Neural Krylov Iteration for Accelerating Linear System Solving

Jian Luo[1] **Jie Wang**[1*] **Hong Wang**[1] **Huanshuo Dong**[1]
Zijie Geng[1] **Hanzhu Chen**[1] **Yufei Kuang**[1]
[1]MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition,
University of Science and Technology of China
{jianluo,wanghong1700}@mail.ustc.edu.cn
jiewangx@ustc.edu.cn

Abstract

Solving large-scale sparse linear systems is essential in fields like mathematics, science, and engineering. Traditional numerical solvers, mainly based on the Krylov subspace iteration algorithm, suffer from the low-efficiency problem, which primarily arises from the less-than-ideal iteration. To tackle this problem, we propose a novel method, namely **Neur**al **K**rylov **It**era**t**ion (**NeurKItt**), for accelerating linear system solving. Specifically, NeurKItt employs a neural operator to predict the invariant subspace of the linear system and then leverages the predicted subspace to accelerate linear system solving. To enhance the subspace prediction accuracy, we utilize QR decomposition for the neural operator outputs and introduce a novel projection loss function for training. NeurKItt benefits the solving by using the predicted subspace to guide the iteration process, significantly reducing the number of iterations. We provide extensive experiments and comprehensive theoretical analyses to demonstrate the feasibility and efficiency of NeurKItt. In our main experiments, NeurKItt accelerates the solving of linear systems across various settings and datasets, achieving up to a 5.5× speedup in computation time and a 16.1× speedup in the number of iterations.

1 Introduction

Solving linear systems is the cornerstone of scientific computing, with applications in various fields including mathematics, science, and engineering[28]. Traditional solvers rely on the Krylov subspace iteration algorithm to tackle large-scale sparse linear systems[59]. It starts with a random initial vector, which corresponds to a one-dimensional linear subspace, and progressively expands this subspace to approximate the solution by iteratively minimizing the residual error.

However, the convergence speed and stability of the Krylov subspace iteration algorithms are significantly influenced by the dimensions and characteristics of the employed subspaces in the iterations[3]. Computational inefficiency and instability often arise in scenarios involving high-dimensional problems or large matrices, especially when the matrices exhibit poor conditioning[9]. This inefficiency primarily arises from the less-than-ideal iteration, which leads to a higher number of iterations and consequently longer solving time. To address these challenges, our key insight is that prior knowledge about the linear system's invariant subspace benefits the Krylov subspace iteration by guiding the iteration process, which reduces the required number of iterations[13].

Motivated by this insight, we introduce **Neur**al **K**rylov **It**era**t**ion (**NeurKItt**), a novel method that leverages neural networks to accelerate linear system solving. NeurKItt comprises two modules:

*Corresponding author. Email: jiewangx@ustc.edu.cn

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

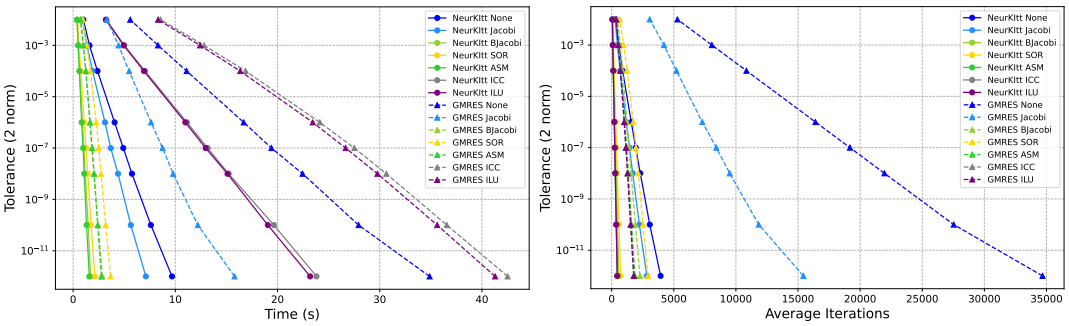

Figure 1: The variation in tolerance for NeurKItt compared to GMRES, where each line represents an experiment under a solving method and a specific preconditioning. Notably, the NeurKItt substantially enhances the efficiency of solving the linear systems, with a reduction in the number of iterations by up to a factor of 16 and achieving a speed-up of up to 5.5 times.

the subspace prediction module and the acceleration module. The subspace prediction module uses a neural network to predict the invariant subspace of linear systems. Inspired by the concept that mapping a linear system to its invariant subspace can be viewed as an operator between two Hilbert spaces, we adopt the neural operator for subspace prediction. We also integrate thin QR decomposition and a projection loss function to optimize training, improving subspace prediction performance. For the acceleration module, it leverages the property that Krylov subspace iteration approximates an invariant subspace of the linear system. When partial information about this subspace is provided, our acceleration module utilizes it to reduce the iterations needed for precise solutions, effectively accelerating the process and addressing the challenges.

We provide comprehensive analyses to demonstrate NeurKItt's efficiency. Furthermore, extensive experiments conducted across various solver settings and different PDE problems validate the effectiveness of our approach. The results show that NeurKItt significantly accelerates linear systems solving, achieving up to a 5.5$\times$ speedup. Both our theoretical analyses and experimental results collectively demonstrate the efficiency of NeuralKItt.

We summarize our contributions as follows:

- To the best of our knowledge, our work is the first to apply the data-driven approach to optimize Krylov iteration algorithms for solving generic non-symmetric linear systems.
- We introduce a novel strategy that predicts the invariant subspace of the linear system to accelerate Krylov iteration. To facilitate the subspace prediction, we design a projection loss for efficient training, in conjunction with QR decomposition for stable outputs.
- Extensive experiments and theoretical analysis demonstrate that NeurKItt reduces the computational cost and the number of iterations for solving linear systems involving non-symmetric matrices.

2 Related Work

2.1 Traditional Numerical Algorithms

In the field of computational mathematics, various algorithms have been devised to address the challenge of solving systems of linear equations. Among these, methods based on the Krylov subspace have garnered attention for their efficacy in handling large matrices[33, 43]. These methods notably mitigate computational complexity by seeking approximate solutions within a constrained, smaller subspace. Within this context, the Generalized Minimal Residual (GMRES) method [45, 14] plays a pivotal role, particularly in addressing non-symmetric matrices.

To enhance the efficiency of these Krylov subspace methods, a range of preconditioning methods are employed[9]. These methods aim to improve matrix conditioning, reduce iteration counts, and boost stability. Preconditioning varieties include Jacobi[46], Additive Schwarz Method (ASM)[64], and

Successive Over-Relaxation (SOR)[42]. They simplify the matrix structure for faster resolution, a significant advantage in large-scale problems.

Our methodology, NeurKItt, utilizes invariant subspaces to hasten convergence in Krylov subspaces, thereby expediting linear system resolutions. It is also engineered to integrate with existing preconditioning techniques, boosting its efficiency and applicability in complex computational scenarios.

2.2 Learning-based Acceleration Algorithms

Although learning-based linear system solvers require training, the training time is negligible compared to the time saved over millions of calls. Thus, there has been a surge in learning-based acceleration efforts in recent years. Research in using neural networks for accelerating linear system solving [30, 23, 60] use neural networks to optimize the Conjugate Gradient algorithm, thereby accelerating the solution of symmetric positive definite linear systems. [37] accelerated the solution of the Poisson equation. [20] focused on accelerating algorithm iterations by learning better algorithm parameters. Recent studies have utilized neural networks for matrix preconditioning. [17, 38, 53] have focused on improving algebraic multigrid preconditioning algorithms. [15, 50] have applied CNNs and machine learning, respectively, to optimize block Jacobi and ILU precondition. Furthermore, specialized preconditioning research is being conducted in areas like physics[2, 7], engineering[47], and climate science[1], demonstrating the breadth of these applications.

However, these studies often face limitations due to specific matrix properties or mainly focus on reducing low-precision solving costs, with fewer advancements in accelerating high-precision solving. In contrast, our approach, NeurKItt, addresses all these limitations. It employs subspace prediction to refine iterative methods universally across Krylov subspace algorithms. This approach not only reduces the initial iterations but also significantly improves the speed and stability of subsequent iterations.

2.3 Neural Operators

Numerical methods for solving PDEs commonly involves solving systems of linear equations, which require extensive computational resources. Recently, Neural operators (NOs), such as the Fourier Neural Operator (FNO)[31, 55, 12] and Deep Operator Network (DeepONet)[35, 36], have shown effectiveness in solving PDEs. Despite their effectiveness, NOs often grapple with challenges like diminished accuracy and stringent boundary condition prerequisites, which limit their applicability as standalone replacements for conventional algorithms in various scientific computing contexts[63, 19]. Given that mapping a matrix to its subspace is fundamentally an operator mapping task, NOs hold significant potential as tools for predicting matrix subspaces.

3 Preliminaries

3.1 Krylov Subspace Iteration

In the realm of large-scale sparse linear systems, Krylov subspace methods are frequently employed as a standard solution strategy [46, 16]. At the heart of this methodology is the principle of utilizing the matrix involved in the linear system to iteratively construct a subspace. This subspace is crafted to approximate the actual solution space, with the iterative process continuing until the generated Krylov subspace sufficiently encompasses the real solution. Such an approach enables the iterative solutions within this confined subspace to progressively approximate the accurate solution vector $\boldsymbol{x}$, thereby efficiently converging towards it.

Suppose we now solve the system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{A} \in \mathbb{C}^{n\times n}$. The m-th Krylov subspace associated with the matrix $\boldsymbol{A}$ and the starting vector $\boldsymbol{r} \in \mathbb{C}^n$ as follows:

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}) = \text{span}\{\boldsymbol{r}, \boldsymbol{A}\boldsymbol{r}, \boldsymbol{A}^2\boldsymbol{r}, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}\}. \tag{2}$$

Generally, $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$ represents the initial residual, where $\boldsymbol{x}_0$ is typically generated randomly or selected as an initial guess for the solution. The iterative process of the Krylov subspace leads to the Arnoldi relation:

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_{m+1}\underline{\boldsymbol{H}}_m = \boldsymbol{V}_m\boldsymbol{H}_m + \boldsymbol{v}_{m+1}h_{m+1,m}\boldsymbol{e}_m^T, \tag{3}$$

where $\boldsymbol{V}_m = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in \mathbb{C}^{n\times m}$ comprises unit vectors $\boldsymbol{v}_i \in \mathbb{C}^n$ for $i = 1, \ldots, m$, mutually orthogonal, and $\boldsymbol{V}_{m+1} = (\boldsymbol{V}_m, \boldsymbol{v}_{m+1})$ extends this set. $\underline{\boldsymbol{H}}_m \in \mathbb{C}^{(m+1)\times m}$ is an upper Hessenberg matrix. $\boldsymbol{H}_m \in \mathbb{C}^{m\times m}$ is the first m rows of $\underline{\boldsymbol{H}}_m$, distinct due to an extra element $h_{m+1,m}$ at the $m+1$-th row, m-th column. $\boldsymbol{e}_m$ is the m-th unit column vector, with its m-th element as 1.

Krylov algorithms reduce the computational effort needed for large linear systems by creating a Krylov subspace. However, starting this process with a random vector can be time-consuming. It's suggested that understanding the matrix's invariant subspace before beginning the iterative process, a warm-starting approach, could notably decrease the required iterations and improve convergence speed and stability.

3.2 Neural Operators

NeurKItt utilizes the framework of neural operator architectures, notably the FNO [26, 31, 27]. We begin by defining a spatial dimension $d \in \mathbb{N}$ and a corresponding domain $D \subset \mathbb{R}^d$. Our focus is on approximating operators $\mathcal{G} : \mathcal{A}(D; \mathbb{R}^{d_a}) \to \mathcal{U}(D; \mathbb{R}^{d_u})$. Here, $a \in \mathcal{A}(D; \mathbb{R}^{d_a})$ and $u \in \mathcal{U}(D; \mathbb{R}^{d_u})$ denote functions that map the domain D to $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$, respectively, where $d_a, d_u \in \mathbb{N}$. Both spaces $\mathcal{A}(D; \mathbb{R}^{d_a})$ and $\mathcal{U}(D; \mathbb{R}^{d_u})$ are identified as Banach spaces.

In line with the definition provided by [32], a neural operator $\mathcal{N} : \mathcal{A}(D; \mathbb{R}^{d_a}) \to \mathcal{U}(D; \mathbb{R}^{d_u})$ is conceptualized as a mapping expressed by

$$\mathcal{N}(a) = \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{R}(a), \tag{4}$$

for a designated depth $L \in \mathbb{N}$. In this formulation, $\mathcal{R} : \mathcal{A}(D; \mathbb{R}^{d_a}) \to \mathcal{U}(D; \mathbb{R}^{d_v})$, with $d_v \geq d_u$, functions as a lifting operator, while $\mathcal{Q} : \mathcal{U}(D; \mathbb{R}^{d_v}) \to \mathcal{U}(D; \mathbb{R}^{d_u})$ serves as a local projection operator.

4 Method

NeurKItt[‡] utilizes a neural operator to predict the invariant subspace $\hat{\mathcal{K}}$ of the linear system, and then uses it to accelerate the Krylov subspace iteration. Generally, NeurKItt is broadly divided into two components.

Subspace Prediction Module: Inspired by operator learning, we employ the neural operator capable of predicting subspaces at a low cost, with the help of QR decomposition. The aim is to precisely predict the invariant subspace $\hat{\mathcal{K}}$ of the linear system.

Acceleration Module: Originating from the mathematical theories of Krylov iteration, we have developed a Krylov algorithm tailored to our problem. Utilizing the invariant subspace, this method expedites Krylov Subspace convergence and enhances stability.

4.1 Subspace Prediction Module

The mapping from the linear system to its corresponding invariant subspace can be considered an operator, i.e., a mapping between two Hilbert spaces. Solving such problems involves finding the corresponding solution operator. In our subspace prediction module, we choose the Fourier Neural Operator (FNO)[31] for subspace prediction. We give a brief introduction to FNO in Appendix C. Generally, for a linear system $Ax = b$ derived from the parametric PDE problem, to predict its invariant subspace $\hat{\mathcal{K}}$, the input to FNO is the input function $a \in \mathbb{R}^{d_a}$ from the given PDE, where $d_a \in \mathbb{N}$. We provide a detailed discussion in Appendix B about how to build a linear system problem from a PDE problem, and what is the input function.

Our task is to learn the mapping between two Hilbert spaces $\mathcal{G} : \mathbb{R}^{d_a} \to \mathbb{C}^{d\times n}$ using FNO. For FNO, the lifting transformation $\mathcal{R}$ first lifts the input a to a higher dimensional representation $v_0 \in \mathbb{C}^{d_a\times c}$, where c is the number of channels. Then we feed the v_0 to Fourier layers. After T Fourier layers forward, we have $v_T \in \mathbb{C}^{d_a\times c}$ from the last Fourier layer, which keeps the same shape as v_0. The FNO's output $X = Q(v_T)$ is the projection of v_T by the transformation $Q : \mathbb{C}^{d_a\times c} \to \mathbb{C}^{d_A\times n}$. NeurKItt then uses QR decomposition to orthogonalize the matrix X, obtaining the predicted

[‡]Our code is available at `https://github.com/smart-JLuo/NeurKItt`

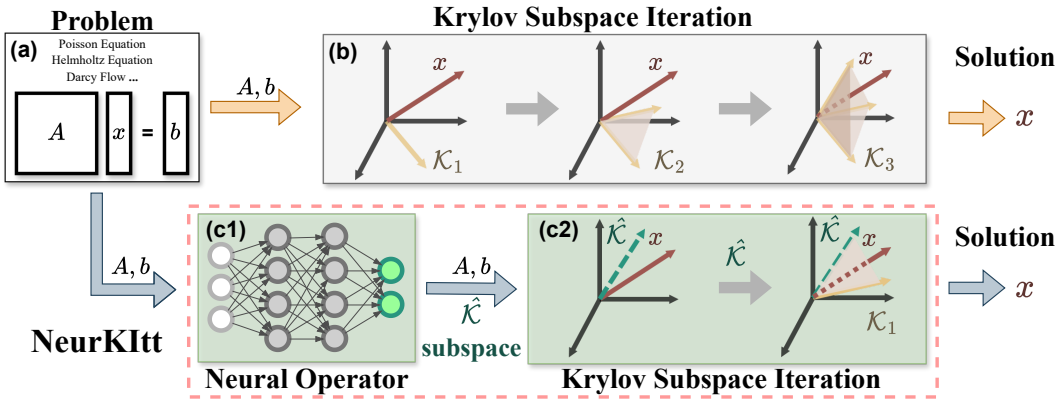

Figure 2: Algorithm Flow Diagram: **(a)** Finding solution $\boldsymbol{x}$ for given matrices $\boldsymbol{A}$ and $\boldsymbol{b}$. **(b)** Traditional Algorithm: Krylov iterations from a random initial vector. **(c1) NeurKItt** Subspace Prediction Module: Utilizing a neural operator for estimating the invariant subspace of matrix $\boldsymbol{A}$. **(c2) NeurKItt** Acceleration Module: Using the predicted invariant subspace of the matrix to guide the iteration, thereby accelerating the Krylov iteration process.

subspace $\hat{\mathcal{K}}$. We provide more details about how to predict the subspace given the 2D Darcy flow problem in Appendix C.

We define the $one-sided\ distance$[51] from the subspace $\mathcal{Q}$ to the subspace $\mathcal{C}$ as

$$\delta(\mathcal{Q}, \mathcal{C}) = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\mathcal{C}})\boldsymbol{\Pi}_{\mathcal{Q}}\|_2, \tag{5}$$

where $\boldsymbol{\Pi}$ represents the projection operator for the associated space. Its mathematical interpretation corresponds to the largest principal angle between the two subspaces $\mathcal{Q}$ and $\mathcal{C}$[3], and defining $\boldsymbol{P}_{\mathcal{Q}}$ as the spectral projector onto $\mathcal{Q}$. The objective of the subspace prediction task is to approximate the invariant subspace of the matrix $\boldsymbol{A}$ with our predicted subspace $\hat{\mathcal{K}}$:

$$\arg\min_{\theta} \delta(\hat{\mathcal{K}}, \mathcal{S}), \tag{6}$$

where θ represents the parameters of NO, and $\delta(\hat{\mathcal{K}}, \mathcal{S})$ is the distance between the two subspaces. Based on subsequent Theoretical Analysis 5.2, we choose $\mathcal{S}$ as the invariant subspace associated with the smallest n eigenvalues, assuming $\mathcal{S} = \text{span}\ \boldsymbol{S} = \text{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n\}$, where s_i is the eigenvector corresponding to the i-th smallest eigenvalue of the given matrix A, $i = 1, 2, \ldots, n$. To reduce computational complexity, considering the norm equivalence theorem in Banach spaces, we optimize using the following loss function:

$$\arg\min_{\theta} \sum_{i=1}^{n} d(\hat{\mathcal{K}}, s_i). \tag{7}$$

This norm transforms the computation of subspace distance into the computation of the distance from a vector to a subspace. According to the properties of the Hilbert space, there exists a unique orthogonal decomposition:

$$s_i = x + y \quad (x \in \hat{\mathcal{K}}, y \in \hat{\mathcal{K}}^{\perp}). \tag{8}$$

We represent the projection operator for $\hat{\mathcal{K}}$ by $\boldsymbol{Q}$, with $\boldsymbol{P}_{\mathcal{Q}} = \boldsymbol{Q}\boldsymbol{Q}^*$, leading to the relationship:

$$d(\hat{\mathcal{K}}, s_i) = \|s_i - \boldsymbol{Q}\boldsymbol{Q}^* s_i\|. \tag{9}$$

Consequently, our projection loss function $l(\hat{\mathcal{K}}, \mathcal{S})$ is defined as follows:

$$l(\hat{\mathcal{K}}, \mathcal{S}) = \sum_{i=1}^{n} \|s_i - \boldsymbol{Q}\boldsymbol{Q}^* s_i\|. \tag{10}$$

We train our subspace module by optimizing the above formula. Experiments have shown that as the projection loss decreases, the principal angle of our predicted subspace also decreases. The principal angle is a mathematical indicator derived in Theoretical Analysis 5.1 that influences the acceleration.

4.2 Acceleration Module

For Krylov algorithms, the iteration count is a critical factor influencing computational load. Drawing inspiration from the Krylov recycling algorithms[40, 13, 6, 58, 41, 24], our approach NeurKItt utilizes the predicted invariant subspace $\hat{\mathcal{K}}$ as the deflation space within the Krylov iteration framework. We provide the pseudocode of the acceleration module in Appendix A.2. The key procedure of our implementation is outlined as follows:

Considering the linear system (1), we obtain a k-dimensional invariant subspace $\hat{\mathcal{K}}$ from the subspace prediction module. Then, NeurKItt computes matrices $\boldsymbol{U}_k, \boldsymbol{C}_k \in \mathbb{C}^{n\times k}$ from $\hat{\mathcal{K}}$ and $\boldsymbol{A}$ such that $\boldsymbol{A}\boldsymbol{U}_k = \boldsymbol{C}_k$ and $\boldsymbol{C}_k^H\boldsymbol{C}_k = \boldsymbol{I}_k$, the specific algorithm can be found in Appendix A.1. We can get the Arnoldi relation of our NeurKItt:

$$(\boldsymbol{I} - \boldsymbol{C}_k\boldsymbol{C}_k^H)\boldsymbol{A}\boldsymbol{V}_{m-k} = \boldsymbol{V}_{m-k+1}\underline{\boldsymbol{H}}_{m-k}. \tag{11}$$

This suggests that when the k-dimensional invariant subspace is given, there is no need to start from scratch for constructing a new Krylov subspace $\mathcal{K}(\boldsymbol{A}, \boldsymbol{r})$. Building upon this foundation, $\mathcal{K}(\boldsymbol{A}, \boldsymbol{r})$ can converge more rapidly to the subspace where the solution $\boldsymbol{x}$ lies. This can significantly reduce the dimensionality of the final Krylov subspace, leading to a marked decrease in the number of iterations and resulting in accelerated performance. Compared to NeurKItt, GMRES can be intuitively conceptualized as the special case of our NeurKItt, where k is initialized at zero[8, 39, 40].

Existing Krylov recycling algorithms share the same Arnoldi relation with NeurKItt. However, they can only accelerate the linear systems solving when there are strongly correlated linear systems provided, which is not common in scientific computing. For example, when solving PDEs with finite element software, matrices vary in size and element positions due to discretization methods and different grids, which makes the recycling invariant subspace from previous matrices impossible. Additionally, storing subspaces from previous solving requires significant storage when the previous linear system is large. Thus, these methods are not widely used in general scenarios. In contrast, NeurKItt, which leverages neural operators to predict subspaces, is not affected by the lack of strongly correlated linear systems.

5 Theoretical Analysis

5.1 Convergence Analysis

Let $\mathcal{Q}$ be an l-dimensional invariant subspace of matrix $\boldsymbol{A}$, and let $\mathcal{C} = \mathrm{range}(\boldsymbol{C}_k)$ be a k-dimensional space ($k \geq l$) selected to approximate $\mathcal{Q}$. We refer to Theorem 3.1 in [40] for an in-depth convergence analysis under NeurKItt.

Theorem 5.1. *[40] Considering a space $\mathcal{C}$, define $\mathcal{V} = \mathrm{range}(\boldsymbol{V}_{m-k+1}\underline{\boldsymbol{H}}_{m-k})$ be the $(m-k)$-dimensional Krylov subspace generated by NeurKItt as in (11). Let $\boldsymbol{r}_0 \in \mathbb{C}^n$, and $\boldsymbol{r}_1 = (\boldsymbol{I} - \boldsymbol{\Pi}_{\mathcal{C}})\boldsymbol{r}_0$. Then, for each $\mathcal{Q}$ such that $\delta(\mathcal{Q}, \mathcal{C}) < 1$,*

$$\begin{aligned}\min_{\boldsymbol{d}_1 \in \mathcal{V}\oplus\mathcal{C}} \|\boldsymbol{r}_0 - \boldsymbol{d}_1\|_2 \leq & \min_{\boldsymbol{d}_2 \in (\boldsymbol{I}-\boldsymbol{P}_{\mathcal{Q}})\mathcal{V}} \|(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{Q}})\boldsymbol{r}_1 - \boldsymbol{d}_2\|_2 \\ & + \frac{\gamma}{1-\delta}\|\boldsymbol{P}_{\mathcal{Q}}\|_2 \cdot \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\mathcal{V}})\boldsymbol{r}_1\|_2,\end{aligned} \tag{12}$$

where $\gamma = \|(\boldsymbol{I} - \boldsymbol{\Pi}_{\mathcal{C}})\boldsymbol{P}_{\mathcal{Q}}\|_2$ and $\delta = \delta(\mathcal{Q}, \mathcal{C})$.

The left-hand side is the residual norm subsequent to $m-k$ iterations of NeurKItt, employing the predictive subspace $\mathcal{C}$. In contrast, on the right-hand side, the initial term epitomizes the convergence of a deflated problem, given that all components within the subspace $\mathcal{Q}$ have been eradicated[39, 49]. The subsequent term on the right embodies a constant multiplied by the residual following $m-k$ iterations of NeurKItt, when solving for $\boldsymbol{r}_1$. Supposed that the predictive space $\mathcal{C}$ encompass an invariant subspace $\mathcal{Q}$, then we have $\delta = \gamma = 0$ for the given $\mathcal{Q}$, which ensures that the convergence rate of NeurKItt matches or surpasses that of the deflated problem. In most cases, $\|\boldsymbol{P}_{\mathcal{Q}}\|_2$ is numerically stable and not large, therefore a reduced value of δ directly correlates with faster convergence in NeurKItt.

Now we compare two approaches to accelerating the linear systems solving[33]:

1. Providing an initial prediction for the solution, allows Krylov algorithms to start from this prediction. The solution involved might come from solutions to similar systems or predictions such as via neural networks[62].
2. Predicting the matrix invariant subspace, using this approximate subspace to speed up Krylov iterations.

For the first approach, particularly with large matrices derived from PDEs, the norm $\|\boldsymbol{A}\|_2$ tends to be large. Our theoretical analysis in Appendix D suggests that FNO's direct solution prediction will not significantly accelerate the linear system solving, possibly only reducing a few Krylov subspace iterations. NeurKItt belongs to the second approach. This crucially speeds up the convergence of Krylov subspace iterations, markedly reducing total iterations and improving stability.

5.2 Subspace Property Analysis

The crucial questions about predicted subspace arise when accelerating Krylov subspace iterations: (1) What kind of subspace should be selected for acceleration? (2) Given the computational cost associated with employing neural networks, is it necessary to expend substantial computational resources to predict a highly accurate subspace?

Regarding the first question: As indicated by our convergence analysis 5.1, effectively accelerating the solution of linear systems only requires a predicted invariant subspace of matrix $\boldsymbol{A}$. A matrix has many invariant subspaces, but which of these are easier to learn by the neural network[40]?

To investigate this problem, we simplify the specific scenario of the linear systems problem. The following definitions and assumptions for Theorem 5.2 are from the reference [24]. Specifically, We deal with a Hermitian positive definite matrix $\boldsymbol{A}$ and a corresponding Hermitian perturbation $\boldsymbol{E}$, allowing $\boldsymbol{A}$ to have the eigendecomposition[24]:

$$\boldsymbol{A} = [\boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_3]\text{diag}(\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2, \boldsymbol{\Lambda}_3)[\boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_3]^H, \tag{13}$$

where $\boldsymbol{Q} = [\boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_3]$ is an orthogonal matrix, $\boldsymbol{\Lambda}_1 = \text{diag}(\lambda_1^{(1)}, \ldots, \lambda_{j_1}^{(1)})$, $\boldsymbol{\Lambda}_2$ and $\boldsymbol{\Lambda}_3$ are defined in a similar fashion. Let the eigenvalues satisfy the following ordering*:

$$\lambda_1^{(1)} \leq \ldots \leq \lambda_{j_1}^{(1)} < \lambda_1^{(2)} \leq \ldots \leq \lambda_{j_2}^{(2)} < \lambda_1^{(3)} \leq \ldots \leq \lambda_{j_3}^{(3)}.$$

We consider the change in the invariant subspace range($\boldsymbol{Q}_1$) under a symmetric perturbation $\boldsymbol{E}$ of $\boldsymbol{A}$. Let $\theta_1(\cdot,\cdot)$ denote the largest canonical angle between two spaces. We do not require that $||\boldsymbol{E}||_F$ be small, but we assume that the projection of $\boldsymbol{E}$ onto the subspace range($[\boldsymbol{Q}_1\boldsymbol{Q}_2]$) is small. We assume that $||[\boldsymbol{Q}_1\boldsymbol{Q}_2]^H\boldsymbol{E}||_F \leq \epsilon$ and that ϵ is small relative to $\lambda_1^{(2)} - \lambda_{j_1}^{(1)}$. We also assume that $\eta = ||\boldsymbol{Q}_3^H\boldsymbol{E}||_F$ is small relative to $\lambda_1^{(3)} - \lambda_{j_1}^{(1)}$. Note that we do not need to assume that $\lambda_{j_2}^{(2)} - \lambda_{j_1}^{(1)}$ is large. Also, let

$$\mu \equiv \min(\lambda_1^{(2)} - \epsilon, \lambda_1^{(3)} - \eta) - 2\epsilon - (\lambda_{j1}^{(1)} + \epsilon) > 2\epsilon,$$

$$\tilde{\mu} \equiv \mu\left(1 - \frac{2\epsilon^2}{\mu^2}\right) + \lambda_{j_1}^{(1)} + \epsilon.$$

Theorem 5.2. *[24] Let $\boldsymbol{A}$ be Hermitian positive definite and have the eigendecomposition given in (13), and let $\boldsymbol{E}$, ϵ, η, μ, and $\tilde{\mu}$ be defined as above. Then there exists a matrix $\tilde{\boldsymbol{Q}}_1$ conforming to $\boldsymbol{Q}_1$ such that range($\tilde{\boldsymbol{Q}}_1$) is a simple invariant subspace of $\boldsymbol{A} + \boldsymbol{E}$, and*

$$\tan\theta_1(\textit{range}(\boldsymbol{Q}_1), \textit{range}(\tilde{\boldsymbol{Q}}_1)) \leq \frac{\epsilon}{\tilde{\mu}}.$$

The specific proof can be found in [24]. A similar bound applies to the perturbation of the eigenvalues associated with $\boldsymbol{Q}_1$. In the context of Theorem 5.1 and our NeurKItt, $\boldsymbol{Q}_1$ corresponds to $\boldsymbol{Q}$, whereas $\boldsymbol{Q}_2$ and $\boldsymbol{Q}_3$ can be chosen to fit the theorem.

This theorem shows that if changes in a matrix occur in the subspace corresponding to larger eigenvalues, the subspace associated with the smallest eigenvalues is minimally affected, as long

*For non-Hermitian matrices, the eigenvalues will be sorted by comparing their modulus, such that $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq |\lambda_4| \leq \cdots \leq |\lambda_n|$, where $\lambda_i \in \mathbb{C}$, $i = 1, 2, \ldots, n$, is the eigenvalue of a given non-Hermitian matrix A.

as the changes are smaller than the gap between the smallest and larger eigenvalues. To facilitate learning of matrix invariant subspaces, we use the subspace composed of eigenvectors corresponding to the smallest eigenvalues as the prediction target for our neural operator.

Regarding the second question: The analysis of the first question, along with the error convergence rate proved in Theorem 5.1, suggests that investing in substantial computational resources for predicting a high-precision subspace is unnecessary. As long as the predicted subspace shows a distinct correlation with the matrix invariant subspace, particularly when δ is small, significant acceleration can be achieved without resorting to high precision[13]. This principle underpins our development of a low-precision, cost-effective subspace prediction framework. Although the subspace prediction may not be precise, the final algorithm has achieved remarkable acceleration results.

6 Experiment

6.1 Experiment Settings

To comprehensively evaluate the performance of NeurKItt, we conducted extensive experiments. Our analysis centers on two primary performance metrics viewed through three perspectives. These tests are conducted on 3 datasets. Specifically, the three Perspectives are: (i) Matrix preconditioning techniques, spanning 7 standard methods. (ii) Accuracy criteria for linear system solutions, emphasizing 8 distinct tolerances. (iii) Different matrix sizes, considering 3 variations. For more details, please refer to the Appendix F.1.

Baselines. NeurKItt focuses on solving linear systems that involve large sparse non-symmetric matrices. The GMRES algorithm is widely used for non-symmetric large sparse linear system solving, which serves as the predominant solution. And we set it as the benchmark for our study. We use GMRES from PETSc (version 3.19.4).

Datasets. To investigate the algorithm's adaptability to various types of matrices, we examined three different linear equation challenges, each rooted in a PDE: 1. Helmholtz Equation [62]. 2. Darcy Flow Problem [31, 44, 27, 36]; 3. Non-uniform Heat Conduction Equation [48, 25, 4, 18]. For an in-depth exposition of the dataset and its generation, kindly refer to the Appendix F.4. For the runtime environment, refer to Appendix F.2.

Metrics. We adopt time speedup for "GMRES solving time / NeurKItt solving time" and iteration speedup for "GMRES iteration count / NeurKItt iteration count". Speedup over 1 denotes better NeurKItt performance. We also provide the average time spent and average iteration count in Appendix H. We use principal angle in Equation 5 to show how close the predicted subspace and the target subspace are.

Table 1: Comparison of our NeurKItt and GMRES computation time and iterations across datasets, preconditioning, and tolerances. The first column lists datasets with matrix size, and the next details tolerances. Results are displayed as "time speedup / iteration speedup".

Dataset	**Tol**	None	Jacobi	BJacobi	SOR	ASM	ICC	ILU
Darcy 160000	1e-2	5.55 / 16.13	3.66 / 11.00	1.87 / 6.82	2.20 / 7.78	1.91 / 6.64	2.64 / 6.35	2.60 / 6.35
	1e-4	4.66 / 12.35	2.89 / 7.66	1.97 / 5.52	2.24 / 6.35	2.05 / 5.58	2.40 / 5.47	2.36 / 5.47
	1e-7	3.95 / 9.93	2.38 / 5.88	1.75 / 4.46	1.98 / 4.97	1.91 / 4.60	2.09 / 4.61	2.06 / 4.62
	1e-10	3.67 / 8.98	2.16 / 5.31	1.65 / 4.08	1.81 / 4.31	1.83 / 4.13	1.86 / 4.16	1.87 / 4.16
	1e-12	3.61 / 8.81	2.22 / 5.45	1.67 / 3.97	1.72 / 4.05	1.74 / 3.94	1.79 / 3.96	1.78 / 3.96
Heat 90000	1e-2	3.15 / 9.48	2.24 / 6.29	1.52 / 5.33	1.54 / 5.96	1.48 / 5.00	1.90 / 4.83	1.97 / 4.83
	1e-4	3.23 / 8.50	2.03 / 5.27	1.47 / 4.79	1.71 / 5.12	1.49 / 4.39	1.84 / 4.30	1.89 / 4.30
	1e-7	2.63 / 7.63	1.55 / 4.35	1.51 / 4.30	1.51 / 4.28	1.45 / 3.81	1.69 / 3.73	1.70 / 3.73
	1e-10	2.93 / 7.34	1.50 / 4.27	1.50 / 4.08	1.42 / 3.90	1.38 / 3.59	1.65 / 3.54	1.64 / 3.54
	1e-12	2.90 / 7.26	1.54 / 4.35	1.45 / 3.96	1.41 / 3.76	1.42 / 3.51	1.61 / 3.46	1.60 / 3.46
Helmholtz 62500	1e-2	2.39 / 7.68	2.37 / 7.68	1.33 / 5.49	1.58 / 5.66	1.32 / 4.83	1.54 / 4.31	1.51 / 4.31
	1e-4	2.69 / 7.99	2.72 / 7.99	1.36 / 4.29	1.45 / 4.36	1.40 / 4.00	1.60 / 3.72	1.59 / 3.72
	1e-7	2.55 / 7.29	2.65 / 7.29	1.52 / 4.08	1.60 / 4.54	1.43 / 3.91	1.68 / 3.76	1.68 / 3.76
	1e-10	2.52 / 6.74	2.52 / 6.74	1.56 / 4.01	1.77 / 4.59	1.56 / 3.95	1.70 / 3.75	1.69 / 3.75
	1e-12	2.53 / 6.56	2.48 / 6.56	1.52 / 3.91	1.72 / 4.56	1.64 / 3.97	1.68 / 3.71	1.67 / 3.71

6.2 Main Experiment

We compare NeurKItt with GMRES under various preconditioners and tolerance, and Table 1 presents main experimental results. More detailed experimental results about time and number of iterations can be found in Appendix H.

The results show that across all tolerances and preconditioning techniques, NeurKIt has been effective in accelerating the linear system solving and reducing the number of iterations required by the Krylov subspace method, achieving up to a 5.5× speedup in computation time and a 16.1× speedup in the number of iterations. These results demonstrate that the invariant subspace predicted by NeurKIt greatly enhances the convergence speed of the Krylov algorithms, thereby reducing the iteration count and accelerating the solving.

With the solution accuracy increasing, the acceleration does not significantly decrease. Though the acceleration effect of NeurKItt varies under different preconditioning, the minimum time speedup is no less than 1.67 in the Darcy flow problem, which suggests NeurKItt can be effectively combined with various preconditioning methods.

6.3 Generalization of Matrix Size

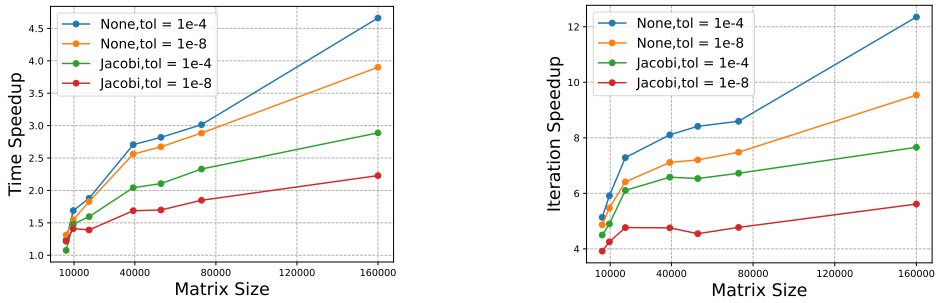

Figure 3: Experiments on the Darcy Flow problem with varying matrix sizes. The results indicate that as the matrix size increases, both time speedup and iteration speedup increase.

We conducted experiments on the Darcy Flow problem with varying matrix sizes under different preconditioning methods and tolerances. We provide the results in Figure 3. The experimental results show that our method consistently improves its acceleration effect as the matrix size increases, across different preconditioning methods and tolerances, which indicates that NeurKItt especially benefits solving large sparse linear systems.

6.4 Ablation Study

We conducted ablation experiments to further validate the performance of FNO and demonstrate the effectiveness of the designed projection loss. The results of the ablation experiments are shown in Table 2. The results demonstrate that FNO significantly outperforms MLP when using the same projection loss. This indicates that FNO is indeed capable of learning a better mapping from the parameter matrix to the subspace, resulting in predicted subspaces that are closer to the target subspaces.

Table 2: Comparison of different settings on principal Angle (rad) for the Darcy flow problem, with a matrix size of 32,400. We replace the FNO with MLP for "w/o FNO" and replace the projection loss with MSE loss for "w/o Projection Loss".

Model	**Principal Angle**↓
NeurKItt	0.07
w/o FNO	0.54
w/o QR decomposition	1.57
w/o Projection Loss	1.57

In addition, we also attempted to utilize the Mean Squared Error (MSE) Loss for subspace prediction. The experimental results show that applying the MSE Loss on subspace learning is merely ineffective. Since we report the principal angle in radians, it implies that the predicted subspace $\hat{\mathcal{K}}$ is merely orthogonal to the target subspace $\mathcal{S}$. Such subspace $\hat{\mathcal{K}}$ cannot be effectively utilized to accelerate linear system solving. Besides, it can be inferred that QR decomposition we have

employed plays an important role in subspace prediction. These results verifies that the projection loss and QR decomposition enable the model to better predict subspace.

7 Limitation and Conclusions

Limitation Our method only considers solving non-symmetric matrices, but for matrices with specific structures, we need to tail our method to achieve faster solving. Furthermore, we have not considered the potential impact of different preconditioning techniques on the subspace and have not optimized for different preconditioning techniques, though the influence caused by preconditioning has little impact based on the observation of our experiments.
Conclusions We propose NeurKItt, a novel method for accelerating linear systems solving through subspace prediction. To the best of our knowledge, this is the first attempt to apply a data-driven approach to optimize the Krylov subspace algorithm for non-symmetric linear systems. Specifically, we employ FNO to predict the invariant subspace and design a projection loss function for this task. By utilizing the predicted invariant subspace, we achieve accelerated Krylov subspace iterations. Theoretical analysis and extensive experiments demonstrate that NeurKItt effectively reduces computational costs and iteration counts.

Acknowledgements

We would like to thank all the anonymous reviewers for their insightful comments. 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 and 62021001.

References

[1] Jan Ackmann, Peter D Düben, Tim N Palmer, and Piotr K Smolarkiewicz. Machine-learned preconditioners for linear solvers in geophysical fluid flows. *arXiv preprint arXiv:2010.02866*, 2020.

[2] Yael Azulay and Eran Treister. Multigrid-augmented deep learning preconditioners for the helmholtz equation. *SIAM Journal on Scientific Computing*, (0):S127–S151, 2022.

[3] Christopher Beattie, Mark Embree, and John Rossi. Convergence of restarted krylov subspaces to invariant subspaces. *SIAM Journal on Matrix Analysis and Applications*, 25(4):1074–1109, 2004.

[4] James V Beck, Ben Blackwell, and Charles R St Clair. *Inverse heat conduction: Ill-posed problems*. James Beck, 1985.

[5] Michele Benzi, Carl D Meyer, and Miroslav Tma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996.

[6] Philipp Birken, Jurjen Duintjer Tebbens, Andreas Meister, and Miroslav Tma. Preconditioner updates applied to cfd model problems. *Applied Numerical Mathematics*, 58(11):1628–1641, 2008.

[7] Salvatore Calì, Daniel C Hackett, Yin Lin, Phiala E Shanahan, and Brian Xiao. Neural-network preconditioners for solving the dirac equation in lattice gauge theory. *Physical Review D*, 107(3):034508, 2023.

[8] Luiz Mariano Carvalho, Serge Gratton, Rafael Lago, and Xavier Vasseur. A flexible generalized conjugate residual method with inner orthogonalization and deflated restarting. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1212–1235, 2011.

[9] Ke Chen. *Matrix preconditioning techniques and applications*, volume 19. Cambridge University Press, 2005.

[10] Lu Cheng and Kuan Xu. Solving time-dependent pdes with the ultraspherical spectral method, 2023.

[11] Eric De Sturler. Truncation strategies for optimal krylov subspace methods. *SIAM Journal on Numerical Analysis*, 36(3):864–889, 1999.

[12] Wenhan Gao, Ruichen Xu, Hong Wang, and Yi Liu. Coordinate transform fourier neural operators for symmetries in physical modelings. *Transactions on Machine Learning Research*, 2024.

[13] André Gaul. Recycling krylov subspace methods for sequences of linear systems. 2014.

[14] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[15] Markus Götz and Hartwig Anzt. 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)*, pages 49–56. IEEE, 2018.

[16] Anne Greenbaum. *Iterative methods for solving linear systems*. SIAM, 1997.

[17] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid pde solvers. In *International Conference on Machine Learning*, pages 2415–2423. PMLR, 2019.

[18] Stefan Güttel and Françoise Tisseur. The nonlinear eigenvalue problem. *Acta Numerica*, 26:1–94, 2017.

[19] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

[20] Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

[21] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[22] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[23] Ayano Kaneda, Osman Akar, Jingyu Chen, Victoria Alicia Trevino Kala, David Hyde, and Joseph Teran. A deep conjugate direction method for iteratively solving linear systems. In *International Conference on Machine Learning*, pages 15720–15736. PMLR, 2023.

[24] Misha E Kilmer and Eric De Sturler. Recycling subspace information for diffuse optical tomography. *SIAM Journal on Scientific Computing*, 27(6):2140–2166, 2006.

[25] Seid Koric and Diab W Abueidda. Data-driven and physics-informed deep learning operators for solution of heat conduction equation with parametric heat source. *International Journal of Heat and Mass Transfer*, 203:123809, 2023.

[26] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *The Journal of Machine Learning Research*, 22(1):13237–13312, 2021.

[27] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[28] Steven J Leon, Lisette G De Pillis, and Lisette G De Pillis. *Linear algebra with applications*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[29] Randall J LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[30] Yichen Li, Peter Yichen Chen, Wojciech Matusik, et al. Learning preconditioner for conjugate gradient pde solvers. *arXiv preprint arXiv:2305.16432*, 2023.

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

[32] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[33] Jörg Liesen and Zdenek Strakos. *Krylov subspace methods: principles and analysis*. Numerical Mathematics and Scie, 2013.

[34] Chih-Jen Lin and Jorge J Moré. Incomplete cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45, 1999.

[35] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[36] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. 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.

[37] Kevin Luna, Katherine Klymko, and Johannes P Blaschke. Accelerating gmres with deep learning in real-time. *arXiv preprint arXiv:2103.10975*, 2021.

[38] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pages 6489–6499. PMLR, 2020.

[39] Ronald B Morgan. Gmres with deflated restarting. *SIAM Journal on Scientific Computing*, 24(1):20–37, 2002.

[40] Michael L Parks, Eric De Sturler, Greg Mackey, Duane D Johnson, and Spandan Maiti. Recycling krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006.

[41] Michael Lawrence Parks. The iterative solution of a sequence of linear systems arising from nonlinear finite element analysis, 2005.

[42] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[43] Ouyuan Qin and Kuan Xu. Solving nonlinear odes with the ultraspherical spectral method. *arXiv preprint arXiv:2306.17688*, 2023.

[44] Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

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

[46] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[47] Johannes Sappl, Laurent Seiler, Matthias Harders, and Wolfgang Rauch. Deep learning of preconditioners for conjugate gradient solvers in urban water related problems. *arXiv preprint arXiv:1906.06925*, 2019.

[48] Rishi Sharma, Amir Barati Farimani, Joe Gomes, Peter Eastman, and Vijay Pande. Weakly-supervised deep learning of heat transport via physics informed loss. *arXiv preprint arXiv:1807.11374*, 2018.

[49] Valeria Simoncini and Daniel B Szyld. On the occurrence of superlinear convergence of exact and inexact krylov subspace methods. *SIAM review*, 47(2):247–272, 2005.

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

[51] G. W. Stewart and Ji guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[52] John C Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.

[53] Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in neural information processing systems*, 34:12129–12140, 2021.

[54] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2004.

[55] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*.

[56] Lloyd N Trefethen. *Spectral methods in MATLAB*. SIAM, 2000.

[57] Lloyd N Trefethen. Spectra and pseudospectra: the behavior of nonnormal matrices and operators. 2020.

[58] Hong Wang, Zhongkai Hao, Jie Wang, Zijie Geng, Zhen Wang, Bin Li, and Feng Wu. Accelerating data generation for neural operators via krylov subspace recycling. *arXiv preprint arXiv:2401.09516*, 2024.

[59] David S Watkins. *The matrix eigenvalue problem: GR and Krylov subspace methods*. SIAM, 2007.

[60] Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

[61] David Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.

[62] Enrui Zhang, Adar Kahana, Eli Turkel, Rishikesh Ranade, Jay Pathak, and George Em Karniadakis. A hybrid iterative numerical transferable solver (hints) for pdes based on deep operator network and relaxation methods. *arXiv preprint arXiv:2208.13273*, 2022.

[63] Xuan Zhang, Limei Wang, Jacob Helwig, Youzhi Luo, Cong Fu, Yaochen Xie, Meng Liu, Yuchao Lin, Zhao Xu, Keqiang Yan, Keir Adams, Maurice Weiler, Xiner Li, Tianfan Fu, Yucheng Wang, Haiyang Yu, YuQing Xie, Xiang Fu, Alex Strasser, Shenglong Xu, Yi Liu, Yuanqi Du, Alexandra Saxton, Hongyi Ling, Hannah Lawrence, Hannes Stärk, Shurui Gui, Carl Edwards, Nicholas Gao, Adriana Ladera, Tailin Wu, Elyssa F. Hofgard, Aria Mansouri Tehrani, Rui Wang, Ameya Daigavane, Montgomery Bohde, Jerry Kurtin, Qian Huang, Tuong Phung, Minkai Xu, Chaitanya K. Joshi, Simon V. Mathis, Kamyar Azizzadenesheli, Ada Fang, Alán Aspuru-Guzik, Erik Bekkers, Michael Bronstein, Marinka Zitnik, Anima Anandkumar, Stefano Ermon, Pietro Liò, Rose Yu, Stephan Günnemann, Jure Leskovec, Heng Ji, Jimeng Sun, Regina Barzilay, Tommi Jaakkola, Connor W. Coley, Xiaoning Qian, Xiaofeng Qian, Tess Smidt, and Shuiwang Ji. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023.

[64] Xuejun Zhang. Multilevel schwarz methods. *Numerische Mathematik*, 63(1):521–539, 1992.

A Algorithmic Details

A.1 computes matrices U_k and C_k

NeurKItt can be modified to solve (1) by carrying over subspace $\hat{\mathcal{K}} = \text{span}\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k\}$ with respect to $\boldsymbol{A}$ as follows [11, 40]:

$$[\boldsymbol{Q}, \boldsymbol{R}] = \text{thin QR decomposition of } \boldsymbol{A}\boldsymbol{Y}_k,$$
$$\boldsymbol{C}_k = \boldsymbol{Q}; \quad \boldsymbol{U}_k = \boldsymbol{Y}_k \boldsymbol{R}^{-1},$$

where the matrix $\boldsymbol{Y}_k = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k]$. The matrices $\boldsymbol{U}_k$ and $\boldsymbol{C}_k$ to satisfy

$$\boldsymbol{A}\boldsymbol{U}_k = \boldsymbol{C}_k; \quad \boldsymbol{C}_k^H \boldsymbol{C}_k = \boldsymbol{I}_k.$$

A.2 NeurKItt Pseudocode

Algorithm 1: NeurKItt Krylov subspace iteration

Select m, the maximum size of the Krylov subspace. Define k as the dimension of the predicted invariant subspace $\hat{\mathcal{K}}$, where $\hat{\mathcal{K}} = \text{span}\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k\}$.

Given matrix A, use subspace prediction module to predict the matrix $\boldsymbol{Y}_k = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k]$ and set tol as the convergence tolerance. Choose an initial guess $\boldsymbol{x}_0$. Calculate $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, and initialize $i = 1$.

Let $[\boldsymbol{Q}, \boldsymbol{R}]$ be the reduced QR-factorization of $\boldsymbol{A}\boldsymbol{Y}_k$.

$\boldsymbol{C}_k = \boldsymbol{Q}$

$\boldsymbol{U}_k = \boldsymbol{Y}_k \boldsymbol{R}^{-1}$

$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{U}_k {\boldsymbol{C}_k}^H \boldsymbol{r}_0$

$\boldsymbol{r}_1 = \boldsymbol{r}_0 - \boldsymbol{C}_k {\boldsymbol{C}_k}^H \boldsymbol{r}_0$

while $\|\boldsymbol{r}_i\|_2 > tol$ **do**

- $i = i + 1$
- Perform $m - k$ Arnoldi steps with the linear operator $(\boldsymbol{I} - \boldsymbol{C}_k \boldsymbol{C}_k^H)\boldsymbol{A}$, letting $\boldsymbol{v}_1 = \boldsymbol{r}_{i-1} / \|\boldsymbol{r}_{i-1}\|_2$ and generating $\boldsymbol{V}_{m-k+1}$, $\underline{\boldsymbol{H}}_{m-k}$ and $\boldsymbol{B}_{m-k}$.
- Let $\boldsymbol{D}_k$ be a diagonal scaling matrix such that $\widetilde{\boldsymbol{U}}_k = \boldsymbol{U}_k \boldsymbol{D}_k$, where the columns of $\widetilde{\boldsymbol{U}}_k$ have unit norm.
- $\widehat{\boldsymbol{V}}_m = [\widetilde{\boldsymbol{U}}_k \;\; \boldsymbol{V}_{m-k}]$
- $\widehat{\boldsymbol{W}}_{m+1} = [\boldsymbol{C}_k \;\; \boldsymbol{V}_{m-k+1}]$
- $\underline{\boldsymbol{G}}_m = \begin{bmatrix} \boldsymbol{D}_k & \boldsymbol{B}_{m-k} \\ 0 & \widetilde{\boldsymbol{H}}_{m-k} \end{bmatrix}$
- Solve min $\|\widehat{\boldsymbol{W}}_{m+1}^H \boldsymbol{r}_{i-1} - \underline{\boldsymbol{G}}_m \boldsymbol{y}\|_2$ for $\boldsymbol{y}$.
- $\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \widehat{\boldsymbol{V}}_m \boldsymbol{y}$
- $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \widehat{\boldsymbol{W}}_{m+1} \underline{\boldsymbol{G}}_m \boldsymbol{y}$
- Compute the k eigenvectors $\widetilde{\boldsymbol{z}}_i$ of $\underline{\boldsymbol{G}}_m^H \underline{\boldsymbol{G}}_m \widetilde{\boldsymbol{z}}_i = \widetilde{\theta}_i \underline{\boldsymbol{G}}_m^H \widehat{\boldsymbol{W}}_{m+1}^H \widehat{\boldsymbol{V}}_m \widetilde{\boldsymbol{z}}_i$ associated with smallest magnitude eigenvalues $\widetilde{\theta}_i$ and store in $\boldsymbol{P}_k$.
- $\widetilde{\boldsymbol{Y}}_k = \widehat{\boldsymbol{V}}_m \boldsymbol{P}_k$
- Let $[\boldsymbol{Q}, \boldsymbol{R}]$ be the reduced QR-factorization of $\underline{\boldsymbol{G}}_m \boldsymbol{P}_k$.
- $\boldsymbol{C}_k = \widehat{\boldsymbol{W}}_{m+1} \boldsymbol{Q}$
- $\boldsymbol{U}_k = \widetilde{\boldsymbol{Y}}_k \boldsymbol{R}^{-1}$

$\boldsymbol{x}_i$ as the numerical solution, $\boldsymbol{r}_i$ as the residual, i as the iteration count.

B From Partial Differential Equation to Linear System

We model a steady-state PDE problem as a corresponding system of linear equations using discrete numerical methods such as FDM, FEM, and FVM [52, 21, 22, 29, 10]. 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.

We take the process of discretizing the two-dimensional inhomogeneous Helmholtz equation using the Finite Difference Method (FDM) as an example to illustrate how to transform a PDE into a system of linear equations:

$$\nabla^2 u(x,y) + k^2(x,y)u(x,y) = f(x,y). \tag{14}$$

This equation could be approximated as:

$$\frac{u(x+\Delta t,y) + u(x-\Delta t,y) + u(x,y+\Delta t) + u(x,y-\Delta t) - 4u(x,y)}{\Delta t^2} + k^2(x,y)u(x,y) = f(x,y), \tag{15}$$

We use 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:

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

In this example, we call the discretization of function $k^2(x,y)$ the input function, corresponding to a 2×2 grid. The function inputs vary from PDE problems.

C A Brief Introduction to Fourier Neural Operator

The Fourier Neural Operator (FNO) aims to map a continuous input function $\mathcal{A}$ to its corresponding continuous output function $\mathcal{U}$ within the Fourier domain. For end-to-end training on FNO, the function pairs $(\mathcal{A}, \mathcal{U})$ are discretized into instance pairs (a, u). The purpose is to learn a mapping $\mathcal{N}$ between (a, u), which can be expressed as:

$$u = \mathcal{N}(a) \tag{16}$$

The mapping $\mathcal{N}$ consists of several sequential steps: first, the input channel is lifted using $\mathcal{R}$; next, the mapping is performed through L Fourier layers $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_L\}$; finally, the output is projected back to the original channel using $\mathcal{Q}$.

$$\mathcal{N}(a) = \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{R}(a), \tag{17}$$

$\mathcal{Q}$ and $\mathcal{R}$ are pixel-wise transformations that can be implemented using models like MLP.

A typical Fourier layer consists of a pixel-wise linear transformation, characterized by weight W and bias b, along with an integral kernel operator $\mathcal{K}$:

$$\mathcal{L}(x) = \sigma(Wx + b + \mathcal{K}(x)), \tag{18}$$

where σ is the nonlinear activation function. The integral kernel operator $\mathcal{K}$ composes three steps: Fast Fourier Transformation (FFT), spectral linear transformation, and inverse FFT.

In the subspace prediction task, we use projection layer $\mathcal{Q}$ mapping the last fno layer output to the target space. Each channel in the FNO output represents one of the basic vector of the predicted invariant subspace. We then apply QR decomposition to the output. Let the input function be $a \in \mathbb{R}^{d_a \times d_a}$ from the Darcy flow problem, where $d_a \in \mathbb{N}$ is the resolution of the input function, which yields a linear system $Ax = b$ for numerical solving, where matrix $A \in \mathbb{R}^{d_A \times d_A}$. We aim to predict a subspace $\hat{\mathcal{K}}$, as a matrix $X \in \mathbb{C}^{d_A \times n}$, for matrix A in the linear system. First, the FNO's

lifting layer $\mathcal{R}$ transforms a to a higher-dimensional representation $v_0 \in \mathbb{C}^{d_a \times d_a \times c}$, while the 3rd dimension is channel dimension. After T Fourier layers forward, we have the $v_T \in \mathbb{C}^{d_a \times d_a \times c}$. Because Fourier layers keep the shape unchanged, we apply a transformation Q to map the v_T to the desired space, $\hat{\mathcal{K}} = Q(v_T)$, with $Q : \mathbb{C}^{d_a \times d_a \times c} \to \mathbb{C}^{d_A \times n}$.

In practice, transformation Q is a stack of transformation layers. It first flattens the first and second dimension of v_T, obtaining $q_0 \in \mathbb{C}^{d_a^2 \times c}$. Then a fully-connected neural network (FNN) applies the mapping $\mathbb{C}^{d_a^2 \times c} \to \mathbb{C}^{d_a^2 \times n}$ to q_0, obtaining $q_1 \in \mathbb{C}^{d_a^2 \times n}$. And another FNN applies the mapping $\mathbb{C}^{d_a^2 \times n} \to \mathbb{C}^{d_A \times n}$ to q_1, obtaining the output $X \in \mathbb{C}^{d_A \times n}$. Finally, we apply QR decomposition to X for orthogonalizing, obtaining $\hat{\mathcal{K}} = \mathrm{span}\{X\}$.

D Analyzing the Efficiency of Using FNO Predictions as Initial Solutions in Linear System Solvers

D.1 Theoretical Analysis

The central premise of this discussion hinges on the potential application of Neural Operators (NOs), such as the Fourier Neural Operator (FNO) [31], in addressing Partial Differential Equation (PDE) problems. Specifically, it explores the feasibility of utilizing these neural operators to predict solutions to PDE problems. Such predictions could serve as preliminary solution vectors for Krylov subspace methods in the resolution of corresponding linear systems. This approach holds promise for expediting the solution process.

However, the practicality of this concept, particularly in contexts where FNO is commonly discussed, warrants careful consideration. This is especially true in scenarios involving large matrix linear systems that emerge from Finite Difference Methods (FDM) [52], spectral methods [56], and similar approaches. A significant obstacle to the application of this method is the disparity between the type of relative error typically discussed in the context of neural operators and the relative error conventionally employed in computational mathematics. This divergence is further complicated by the impact of the norm of the PDE discretization matrix, which exhibits a tendency to increase with the refinement of the grid.

In the context of linear systems derived from PDEs, we consider the following standard formulation:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{19}$$

Within the framework of NOs, the prevalent metric for error assessment is the Mean Squared Error (MSE) of the solution function or solution vector $\boldsymbol{x}$, designated as $rtol_{NO}$:

$$rtol_{NO} = \frac{\|\hat{\boldsymbol{x}}_{NO} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|}, \tag{20}$$

where $\hat{\boldsymbol{x}}_{NO}$ represents the solution as predicted by the NO, while $\boldsymbol{x}$ denotes the precise solution.

In the domain of computational mathematics, the relative residual $rtol$ is commonly utilized as the error metric for resolving linear systems:

$$rtol = \frac{\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|}{\|\boldsymbol{x}\|}, \tag{21}$$

where $\hat{\boldsymbol{x}}$ represents the solution as predicted by the Krylov subspace method.

Utilizing orthogonal polynomial grid theory, specifically Theorem 30.1 in Trefethen's work [57], page 290, offers crucial insights for this analysis.

Theorem D.1. *[57, P. 290] For any N, $\|\boldsymbol{D}_N\|_2 > N^2/3$.*

In this context, N denotes the count of discretization points in a one-dimensional Ordinary Differential Equation (ODE), and $\boldsymbol{D}_N$ represents the discretization matrix constructed using Chebyshev orthogonal polynomial zeros as grid points, with the matrix dimension being N. This assertion holds similarly for other orthogonal polynomial grids, and a uniformly divided grid can be analogized to a grid defined by the zeros of Fourier polynomials.

The theorem suggests that with an increase in the number of grid points, resulting in a larger matrix size, the norm of the matrix $\|\boldsymbol{A}\|_2$ associated with the discretized PDE also escalates. Then We consider the following equation:

$$\frac{rtol_{NO}}{rtol} = \frac{\|\hat{\boldsymbol{x}}_{NO} - \boldsymbol{x}\|}{\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|} = \frac{\|\hat{\boldsymbol{x}}_{NO} - \boldsymbol{x}\|}{\|\boldsymbol{A}(\hat{\boldsymbol{x}} - \boldsymbol{x})\|}. \tag{22}$$

Consequently, when $\|\boldsymbol{A}\|_2$ is substantially large, there emerges a notable discrepancy between $rtol_{NO}$ and $rtol$. Generally, the predictive accuracy $rtol_{NO}$ of the FNO lies in the range of 10^{-2} to 10^{-4}. However, if the norm $\|\boldsymbol{A}\|_2$ approaches the magnitude of 10^5, the relative tolerance $rtol$ as per computational mathematics algorithms may only range between 10^0 and 10^2. In such instances, leveraging the FNO's predicted solution as the initial vector for solving the linear system may not substantially expedite the solution process.

D.2 Experimental Analysis

To empirically corroborate the theoretical framework presented, we delve into the Heat dataset F.4.3 utilized in this study, characterized by a matrix size of 52900. The experimental results underscore several noteworthy observations.

Employing the original FNO for training, we observed $rtol_{NO} \sim 6 * 10^{-3}$, which is in line with the benchmarks reported in the foundational FNO paper. However, because $\|\boldsymbol{A}\|_2 \sim 1.2 * 10^6$, it results in $rtol \sim 60$. This empirical finding is consistent with our theoretical predictions. In such a context, utilizing the FNO-derived solution as the initial vector for the linear system does not markedly expedite the resolution process.

This analysis implicitly suggests that utilizing predicted solutions as initial vectors in iterative algorithms mandates a high level of precision in the neural network model. Nonetheless, it is observed that in standard application scenarios, the impact on accelerating the process is marginal.

In contrast to this method, our algorithm NeurKItt, by predicting the invariant subspace, not only reduces the initial number of iterations but also, more importantly, substantially improves the convergence rate in subsequent iterations, As demonstrated in the convergence analysis 5.1. This marks a pivotal deviation from former techniques that focus on expediting the process by merely predicting initial solutions. Our theoretical analysis 5.2 shows that the predicted invariant subspace does not require high precision to accelerate the solution of linear systems and significantly reduce the number of iterations.

E Thin QR Decomposition

In our research, we utilize the Thin QR decomposition algorithm, a prevalent technique in numerical linear algebra [14]. The essence of Thin QR decomposition lies in its capacity to factorize a matrix into two distinct components: an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{R}$. This factorization is instrumental in diminishing both the computational complexity and the storage demands.

For a matrix $\boldsymbol{A}$ belonging to $\mathbb{C}^{n\times m}$ with the condition $n \geq m$, the Thin QR decomposition is mathematically articulated as:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \tag{23}$$

where $\boldsymbol{Q}$ is an $n \times m$ orthogonal matrix that complies with $\boldsymbol{Q}^*\boldsymbol{Q} = \boldsymbol{I}$, and $\boldsymbol{R}$ is an $m \times m$ upper triangular matrix. In our study, the integration of QR decomposition within the neural network architecture is strategically employed to effectively resolve the orthogonal basis of a subspace.

F Details of Experimental Data and Setup

F.1 Specific parameters of the main experiment

Model Training and Predicting: We employ the 5 FNO layers with modes from {20, 32, 40} and the width from {32, 50, 64}. The learning rate is fix at 1×10^{-3} while the batch size is selected from {16, 32}. The number of subspace dimensions is fixed at 10 in all experiments. For darcy flow and

heat datasets, we generate 8000 samples for training and 1600 samples for testing. For Helmholtz, we generate 1000 samples for training and 200 samples for testing.

Baseline: To achieve maximum efficiency, we initiated our process by creating linear equation systems of PDEs using either Python. These were subsequently processed within the C programming environment. For the GMRES solver, we employed the most recent version of PETSc, specifically 3.19.4.

Three Perspectives:

1. **Precondition**: The act of preconditioning is vital when tackling large matrix equations, as it significantly speeds up the resolution process and augments algorithmic stability. Given the variety of scenarios that may arise, we explored over 10 prevalent preconditioning methods, adapting them to suit different contexts.

2. **Tolerance**: The precision threshold of a solution directly influences both the number of iterations required and the overall computational time. Different algorithms manifest varying rates of convergence, and specific NOs hold unique tolerance standards. Our comprehensive evaluation encompassed a range of tolerances, with a focus on pinpointing 5-8 levels of optimal error precision for our analyses.

3. **Matrix Dimensionality**: The efficiency of an algorithm can vary with changes in matrix size. Furthermore, different NOs often require matrices of various sizes. Therefore, our investigation included the study of 5 distinct matrix sizes.

Two Performance Metrics: 1. **Computational Time**: This metric provides a clear and direct measure of an algorithm's performance. 2. **Iteration Count**: The count of iterations required reflects the algorithm's numerical sensitivity and stability, offering valuable insights into its performance characteristics.

F.2 Environment

To ensure consistency in our evaluations, all comparative experiments were conducted under uniform computing environments. Specifically, the environments used are detailed as follows:

CPU: Intel® Xeon® Gold 6246R CPU @ 3.40GHz
GPU: NVIDIA GeForce RTX 3090

F.3 Precondition

In this study, we implemented a series of preconditioning methods for our experiments, which are outlined below.

1. None: In scenarios where 'none' is selected for preconditioning, the linear system is addressed directly in its original form. This means that no preliminary transformations or adjustments are applied to the system before the iterative solving process.

2. Diagonal Preconditioning (Jacobi) [46]: This technique focuses solely on the diagonal components of the coefficient matrix. The preconditioner in this approach is essentially the inverse of these diagonal elements, offering a straightforward preconditioning solution.

3. Block Jacobi (BJacobi) [5]: A more advanced form of Jacobi preconditioning, Block Jacobi divides the coefficient matrix into smaller, manageable blocks, each representing a specific subdomain or a separate problem. These blocks are then preconditioned individually using their diagonal elements.

4. Successive Over-relaxation (SOR) [61]: SOR, a modification of the Gauss-Seidel method, incorporates a relaxation factor to enhance the process of convergence. This method reformulates the original problem by applying a weight, often improving the convergence rate for certain iterative algorithms.

5. Additive Schwarz Method (ASM) [54]: ASM operates on the principle of domain decomposition, partitioning the main problem into several smaller subdomains. Each subdomain's problem is solved independently, and the local solutions are then aggregated to form the overall solution.

6. Incomplete Cholesky (ICC) [34]: ICC is a preconditioning method based on the Cholesky decomposition, but drops certain off-diagonal elements during the decomposition, making it "incomplete". It's utilized for symmetric positive definite problems.

7. Incomplete LU (ILU) [46]: ILU is based on LU decomposition, but like ICC, drops certain off-diagonal elements during the decomposition. ILU can be applied to nonsymmetric problems.

F.4 Datasets

F.4.1 Helmholtz Equation

We consider a two-dimensional Helmholtz equation, which can be described by the following equation [62]:

$$\Delta u(x,y) + k^2(x,y)u(x,y) = f(x,y), \tag{24}$$

where $u(x,y)$ is the wave function, and $k(x,y)$ denotes the spatially varying wavenumber, with f representing the source term, corresponding to origins of electromagnetic or acoustic waves. This equation is pivotal in several physical realms, notably in Acoustics, Electromagnetism, and Quantum Mechanics. Within these fields, the Helmholtz equation models phenomena such as wave propagation and vibration patterns.

In the context of the Helmholtz equation, the wavenumber $k(x,y)$ is intimately connected to the frequency of the wave and the characteristics of the medium through which the wave propagates. In our experimental setup, the value of $k(x,y)$ is determined using the GRF method [35] while the $f(x,y)$ is the constant.

F.4.2 Darcy Flow problem

We consider two-dimensional Darcy flows, which can be described by the following equation [31, 44, 27, 36]:

$$-\nabla \cdot (K(x,y)\nabla h(x,y)) = f(x,y), \tag{25}$$

where K is the permeability field, h is the pressure, and f is a source term which can be either a constant or a space-dependent function.

In our experiment, $K(x,y)$ is derived using the GRF method [35]. The term $f(x,y)$ is consistently set as a constant f.

F.4.3 Non-uniform Heat Conduction Equation

We consider a two-dimensional heat conduction equation in a non-uniform medium, which is formulated as follows [4, 18]:

$$k(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}) + \frac{\partial k}{\partial x}\frac{\partial T}{\partial x} + \frac{\partial k}{\partial y}\frac{\partial T}{\partial y} + qT = f, \tag{26}$$

where $T(x,y)$ is the temperature, $k(x,y)$ represents the spatially varying thermal conductivity, and $q(x,y)T(x,y)$ models internal or external heat sources/sinks. The term $f(x,y)$ acts as an external heat influence.

This equation is crucial for studying heat distribution in materials with varying properties, such as in the thermal management of electronic devices. Solving this equation helps predict temperature variations, essential for designing effective cooling or heating systems.

In our experiment, $k(x,y)$ is derived using the GRF [35]. And $q(x,y)$ is samples from $U(0,1)$. The term $f(x,y)$ is consistently set as a constant f.

G Analysis of Hyperparameters

G.1 Hyperparameters of Fourier Neural Operator

Fourier Neural Operator has three key hyperparameters: model layers, mode and width for fourier layer. We conduct experiments to investigate the impacts of these hyperparameters. Results in Table 8 suggest that a proper combination of these hyper-parameters will improve the performance.

Table 8: Performance of NeurKItt with changes in layers, modes, and widths. We use Darcy Flow problem with matrix size of 32400 in these experiments, where the tolerance is fixed to 1e-5 and the preconditioning is None. Subspace dimension is fixed to 20.

Layer	2	5	8	10
Principal Angle	0.18	0.12	0.11	0.13
Time Speedup	1.47	1.68	1.69	1.54
Iteration Speedup	8.15	9.86	9.45	8.04

Mode	8	16	32	64
Principal Angle	0.47	0.22	0.13	0.10
Time Speedup	1.19	1.46	1.69	1.71
Iteration Speedup	5.91	7.96	9.86	9.66

Width	8	16	32	64
Principal Angle	0.25	0.14	0.12	0.09
Time Speedup	0.98	1.64	1.69	1.83
Iteration Speedup	4.93	9.23	9.86	10.29

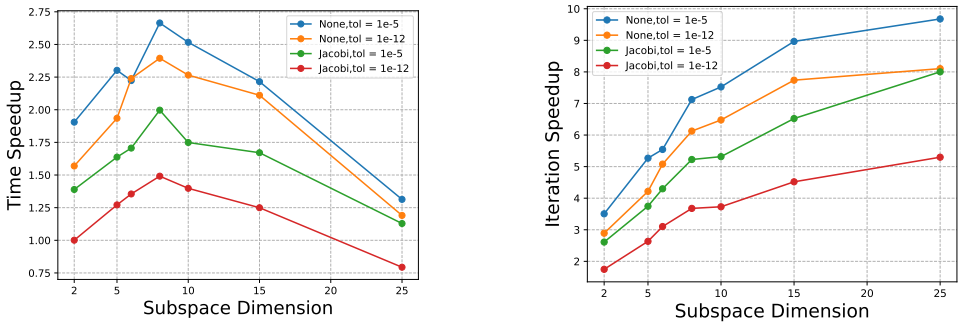

Figure 4: Experiments on the Darcy Flow problem with varying subspace dimensions. The matrix size is fixed to 32400.

G.2 Impact of Invariant Subspace Dimension

We conduct extensive experiments investigate the acceleration impact of subspace dimension. Results in Figure 4 shows that with the subspace dimension increasing, iteration speedup increase as well.

However, the time speedup does not consistently increase with the subspace dimension. As shown in the left figure of Figure 4, the time speedup reaches its maximum when the subspace dimension is 8. When the subspace dimension increase from 8, the time speedup decreases. This may be influenced by the subspace size since the subspaces are dense matrices. Recalling the Arnoldi relation in Section 4.2, larger subspaces mean larger dense matrix multiplications. When the subspace dimension is sufficiently large, the additional time cost of dense matrix operations will offset the time savings from reduced iterations, thereby reducing the overall time speedup. This indicates that a larger subspace dimension is not necessarily better.

H Detailed experimental results

H.1 Time Efficiency Analysis for Subspace Prediction Module

Table 9: The inference time spent of subspace prediction module

Dataset	Helmholtz 62500	Heat 90000	Darcy 160000
Inference Time (ms)	6.16	7.04	8.81

Table 9 shows the time cost of the subspace prediction module for a single pass. The first row lists datasets with matrix side lengths. It's obvious that across various datasets, the computational cost of the subspace prediction module is extremely low, typically taking only a few milliseconds.

For comparison, in Darcy Flow problem, solving a linear system using GMRES without preconditioning at a tolerance of 1e-12 requires 34.8 seconds. While the neural network's inference cost is much more lower, which is negligible.

H.2 Training Time Analysis

Table 10: The 120 epochs training cost for subspace prediction module.

Dataset	Helmholtz 62500	Heat 90000	Darcy 160000
Training Time (h)	0.51	6.48	10.47

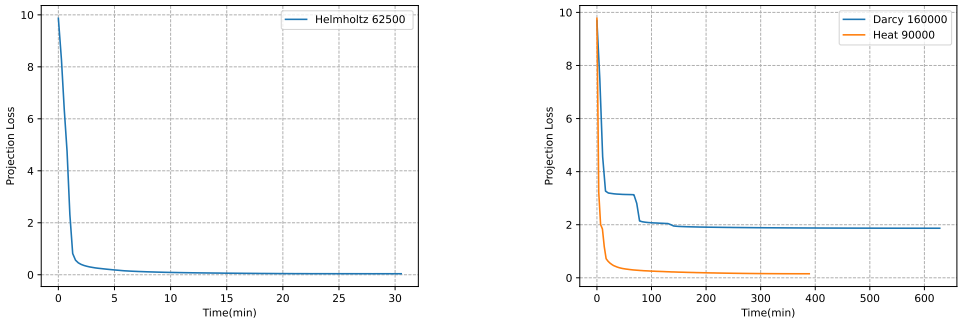

Figure 5: Training time costs for main experiments.

We train each dataset for 120 epochs, and list the training costs in Table 10. The total training time for the Helmholtz dataset is shorter compared to the other two datasets. This is because the problem is simple, allowing us to train on a dataset with fewer samples, and the matrix size involved in the problem is smaller than Darcy Flow and Heat problem.

As shown in the Figure 5, the training typically converges within 100 min. For the Helmholtz problem, it converges in 10 min; for the Darcy Flow dataset and Heat dataset, it converges less than 100 min. However, in scientific computing software, where solving linear systems is frequently required and often time-consuming, the time saved by our approach will add-up to pay off the training costs.

H.3 Darcy Flow problem

Table 11: Comparison of time and iteration counts for two algorithms, with varying preconditioners and tolerances

Preconditioning	Metrics	Methods	1.E-02	1.E-03	1.E-04	1.E-06	1.E-07	1.E-08	1.E-10	1.E-12
None	time(s)	GMRES	5.58 ±0.48	8.31 ±0.71	11.11 ±0.94	16.69 ±1.42	19.41 ±1.65	22.45 ±1.91	27.93 ±2.37	34.88 ±3.14
		NeurKItt	1.01 ±0.26	1.60 ±0.34	2.38 ±0.47	4.06 ±0.64	4.91 ±0.71	5.76 ±0.79	7.61 ±0.94	9.68 ±1.17
	iter	GMRES	5291.44 ±451.50	8068.17 ±686.88	10845.95 ±922.37	16402.62 ±1393.26	19181.24 ±1628.79	21957.72 ±1865.57	27531.72 ±2338.23	34683.57 ±3126.89
		NeurKItt	328.15 ±84.59	573.21 ±121.09	878.11 ±172.60	1576.28 ±250.07	1932.16 ±278.21	2301.87 ±315.52	3065.79 ±377.01	3938.56 ±477.31
Jacobi	time(s)	GMRES	3.30 ±0.48	4.44 ±0.97	5.48 ±1.48	7.62 ±1.99	8.75 ±2.09	9.79 ±2.27	12.20 ±2.63	15.79 ±2.97
		NeurKItt	0.90 ±0.17	1.36 ±0.21	1.90 ±0.26	3.11 ±0.30	3.68 ±0.32	4.39 ±0.35	5.64 ±0.37	7.11 ±0.37
	iter	GMRES	3058.70 ±448.68	4211.90 ±923.94	5213.33 ±1410.40	7292.02 ±1900.35	8426.35 ±2011.34	9503.62 ±2207.99	11837.47 ±2555.77	15432.57 ±2898.05
		NeurKItt	278.29 ±54.04	464.90 ±71.55	680.83 ±93.54	1177.92 ±114.75	1433.17 ±125.27	1693.06 ±135.17	2228.52 ±144.24	2829.14 ±148.50
BJacobi	time(s)	GMRES	0.73 ±0.06	1.00 ±0.12	1.24 ±0.19	1.64 ±0.31	1.83 ±0.36	2.04 ±0.40	2.43 ±0.46	2.86 ±0.52
		NeurKItt	0.39 ±0.03	0.51 ±0.03	0.63 ±0.04	0.91 ±0.05	1.05 ±0.05	1.19 ±0.05	1.47 ±0.05	1.71 ±0.05
	iter	GMRES	470.08 ±37.75	685.41 ±81.05	882.00 ±131.77	1231.23 ±234.76	1396.76 ±277.79	1564.51 ±310.30	1910.46 ±359.60	2276.63 ±414.83
		NeurKItt	68.89 ±4.41	110.98 ±5.43	159.66 ±11.13	262.87 ±13.71	312.86 ±14.51	363.86 ±15.75	468.04 ±17.33	574.13 ±18.32
SOR	time(s)	GMRES	0.94 ±0.05	1.33 ±0.09	1.69 ±0.15	2.27 ±0.30	2.51 ±0.37	2.73 ±0.44	3.19 ±0.59	3.67 ±0.73
		NeurKItt	0.43 ±0.02	0.57 ±0.03	0.76 ±0.06	1.09 ±0.07	1.27 ±0.07	1.43 ±0.07	1.76 ±0.07	2.13 ±0.08
	iter	GMRES	653.60 ±33.76	965.24 ±62.79	1247.77 ±108.19	1732.33 ±228.54	1946.67 ±287.30	2145.13 ±345.91	2524.51 ±467.13	2914.28 ±579.27
		NeurKItt	84.07 ±4.83	135.66 ±8.08	196.51 ±14.86	327.29 ±20.56	391.85 ±20.88	455.74 ±22.03	586.11 ±24.66	720.40 ±27.13
ASM	time(s)	GMRES	0.72 ±0.04	1.00 ±0.06	1.24 ±0.09	1.66 ±0.18	1.87 ±0.23	2.06 ±0.28	2.40 ±0.37	2.76 ±0.47
		NeurKItt	0.38 ±0.02	0.49 ±0.03	0.61 ±0.04	0.83 ±0.04	0.98 ±0.04	1.09 ±0.05	1.31 ±0.05	1.58 ±0.04
	iter	GMRES	385.30 ±19.89	573.78 ±32.55	748.66 ±54.73	1064.04 ±117.68	1208.00 ±150.75	1346.39 ±184.23	1607.47 ±250.70	1873.52 ±318.06
		NeurKItt	58.03 ±3.57	93.66 ±5.01	134.21 ±8.53	220.25 ±11.07	262.54 ±11.53	304.28 ±12.72	388.87 ±13.53	475.30 ±13.40
ICC	time(s)	GMRES	8.56 ±0.39	12.81 ±0.68	16.88 ±1.09	24.14 ±2.44	27.53 ±3.21	30.64 ±3.99	36.56 ±5.51	42.52 ±7.14
		NeurKItt	3.24 ±0.19	5.01 ±0.26	7.04 ±0.45	11.14 ±0.56	13.16 ±0.63	15.22 ±0.66	19.65 ±0.72	23.81 ±0.76
	iter	GMRES	354.05 ±16.03	532.65 ±28.46	701.68 ±45.45	1006.67 ±101.90	1146.12 ±133.70	1278.82 ±166.37	1527.25 ±230.08	1776.70 ±298.47
		NeurKItt	55.75 ±3.22	89.39 ±4.67	128.23 ±8.27	209.24 ±10.52	248.35 ±11.93	287.80 ±12.49	367.34 ±13.44	448.64 ±14.33
ILU	time(s)	GMRES	8.31 ±0.38	12.42 ±0.66	16.35 ±1.06	23.44 ±2.37	26.66 ±3.11	29.77 ±3.87	35.62 ±5.37	41.29 ±6.94
		NeurKItt	3.19 ±0.18	4.95 ±0.26	6.93 ±0.45	10.98 ±0.55	12.96 ±0.62	15.10 ±0.65	19.05 ±0.69	23.19 ±0.73
	iter	GMRES	354.05 ±16.03	532.65 ±28.46	701.68 ±45.45	1006.67 ±101.90	1146.12 ±133.70	1278.82 ±166.37	1527.27 ±230.11	1776.33 ±298.46
		NeurKItt	55.75 ±3.22	89.39 ±4.67	128.23 ±8.27	209.18 ±10.40	248.34 ±11.86	287.72 ±12.46	367.22 ±13.32	448.45 ±14.10

H.4 Non-uniform Heat Conduction Equation

Table 12: Comparison of time and iteration counts for two algorithms, with varying preconditioners and tolerances

Preconditioning	Metrics	Methods	1.E-02	1.E-03	1.E-04	1.E-06	1.E-07	1.E-08	1.E-10	1.E-12
None	time(s)	GMRES	1.83 ±0.15	2.71 ±0.22	3.62 ±0.29	5.43 ±0.44	6.31 ±0.51	7.21 ±0.58	9.10 ±0.73	11.01 ±0.90
		NeurKItt	0.58 ±0.14	0.83 ±0.14	1.12 ±0.17	1.79 ±0.23	2.40 ±0.28	2.74 ±0.31	3.10 ±0.33	3.80 ±0.39
	iter	GMRES	2887.92 ±233.25	4406.10 ±355.17	5925.35 ±476.97	8965.19 ±720.09	10485.41 ±841.59	12005.91 ±963.13	15046.42 ±1205.89	18445.01 ±1510.96
		NeurKItt	304.70 ±70.76	491.21 ±83.34	697.08 ±104.10	1150.39 ±149.72	1374.07 ±160.00	1594.93 ±178.52	2051.06 ±217.49	2539.87 ±263.78
Jacobi	time(s)	GMRES	1.20 ±0.13	1.62 ±0.27	2.00 ±0.44	2.57 ±0.78	2.87 ±0.93	3.22 ±0.96	4.05 ±0.95	5.04 ±1.06
		NeurKItt	0.54 ±0.08	0.76 ±0.09	0.98 ±0.10	1.43 ±0.12	1.85 ±0.14	2.16 ±0.14	2.70 ±0.13	3.27 ±0.14
	iter	GMRES	1797.71 ±199.91	2494.54 ±423.51	3080.30 ±682.55	4100.70 ±1243.05	4580.26 ±1480.60	5186.70 ±1551.22	6579.74 ±1542.97	8193.87 ±1727.63
		NeurKItt	285.97 ±42.54	429.03 ±51.99	584.15 ±61.37	897.17 ±74.56	1053.92 ±77.06	1215.19 ±80.02	1540.85 ±75.40	1882.28 ±82.20
BJacobi	time(s)	GMRES	0.31 ±0.02	0.43 ±0.03	0.55 ±0.03	0.75 ±0.05	0.87 ±0.06	0.95 ±0.07	1.12 ±0.10	1.28 ±0.12
		NeurKItt	0.21 ±0.02	0.27 ±0.02	0.37 ±0.03	0.49 ±0.03	0.57 ±0.03	0.63 ±0.04	0.75 ±0.04	0.88 ±0.03
	iter	GMRES	294.42 ±17.69	452.82 ±28.30	608.86 ±37.22	905.63 ±58.97	1044.87 ±73.22	1180.64 ±88.20	1434.14 ±124.40	1679.72 ±162.88
		NeurKItt	55.22 ±5.27	88.85 ±7.47	127.04 ±10.84	205.67 ±13.44	243.03 ±14.58	278.57 ±16.12	351.18 ±16.59	424.54 ±16.41
SOR	time(s)	GMRES	0.39 ±0.02	0.54 ±0.03	0.66 ±0.05	0.89 ±0.11	1.01 ±0.14	1.10 ±0.16	1.29 ±0.22	1.49 ±0.27
		NeurKItt	0.25 ±0.04	0.31 ±0.04	0.39 ±0.04	0.57 ±0.04	0.67 ±0.05	0.72 ±0.04	0.91 ±0.04	1.06 ±0.04
	iter	GMRES	406.26 ±21.77	606.91 ±38.09	789.00 ±61.21	1113.74 ±131.64	1262.06 ±170.20	1405.04 ±209.51	1677.24 ±286.34	1954.73 ±352.50
		NeurKItt	68.16 ±9.61	108.06 ±13.71	154.12 ±16.66	249.55 ±19.08	295.04 ±20.15	340.18 ±20.86	429.65 ±20.84	519.69 ±21.00
ASM	time(s)	GMRES	0.31 ±0.02	0.41 ±0.02	0.49 ±0.02	0.66 ±0.04	0.75 ±0.06	0.84 ±0.07	0.97 ±0.08	1.13 ±0.10
		NeurKItt	0.21 ±0.02	0.29 ±0.02	0.33 ±0.02	0.45 ±0.03	0.51 ±0.03	0.58 ±0.03	0.70 ±0.03	0.80 ±0.03
	iter	GMRES	235.58 ±14.47	348.55 ±13.33	455.52 ±17.65	658.63 ±44.46	755.72 ±57.18	850.45 ±67.29	1031.84 ±89.71	1212.11 ±111.94
		NeurKItt	47.14 ±3.90	73.28 ±5.16	103.84 ±7.59	167.22 ±10.85	198.55 ±11.42	228.53 ±12.23	287.15 ±12.52	345.50 ±12.87
ICC	time(s)	GMRES	2.84 ±0.14	4.31 ±0.22	5.59 ±0.21	8.00 ±0.46	9.23 ±0.66	10.50 ±0.83	12.72 ±1.06	14.87 ±1.35
		NeurKItt	1.49 ±0.12	2.24 ±0.15	3.03 ±0.22	4.71 ±0.29	5.47 ±0.32	6.23 ±0.32	7.72 ±0.34	9.24 ±0.34
	iter	GMRES	214.30 ±10.71	324.07 ±16.36	423.08 ±16.08	611.08 ±35.00	700.66 ±50.37	787.07 ±62.61	958.47 ±79.77	1125.41 ±102.41
		NeurKItt	44.41 ±3.69	69.62 ±4.52	98.37 ±7.22	158.96 ±9.88	187.98 ±11.13	215.77 ±11.06	270.79 ±11.75	325.39 ±12.05
ILU	time(s)	GMRES	2.85 ±0.14	4.28 ±0.22	5.58 ±0.21	7.97 ±0.46	9.15 ±0.66	10.35 ±0.82	12.51 ±1.04	14.64 ±1.33
		NeurKItt	1.45 ±0.12	2.17 ±0.14	2.96 ±0.22	4.62 ±0.29	5.39 ±0.32	6.12 ±0.31	7.64 ±0.33	9.16 ±0.34
	iter	GMRES	214.30 ±10.71	324.07 ±16.36	423.08 ±16.08	611.08 ±35.00	700.66 ±50.37	787.07 ±62.61	958.47 ±79.77	1125.41 ±102.39
		NeurKItt	44.41 ±3.69	69.62 ±4.52	98.37 ±7.22	158.96 ±9.88	187.98 ±11.13	215.77 ±11.06	270.79 ±11.75	325.38 ±12.06

H.5 Helmholtz Equation

Table 13: Comparison of time and iteration counts for two algorithms, with varying preconditioners and tolerances

Preconditioning	Metrics	Methods	1.E-02	1.E-03	1.E-04	1.E-06	1.E-07	1.E-08	1.E-10	1.E-12
None	time(s)	GMRES	0.68 ±0.01	1.01 ±0.01	1.32 ±0.01	1.94 ±0.02	2.23 ±0.02	2.55 ±0.02	3.18 ±0.02	3.79 ±0.03
		NeurKItt	0.28 ±0.01	0.39 ±0.02	0.49 ±0.02	0.71 ±0.03	0.87 ±0.04	0.98 ±0.04	1.26 ±0.04	1.50 ±0.04
	iter	GMRES	1378.28 ±10.53	2101.40 ±16.11	2825.34 ±21.76	4274.92 ±32.99	5000.12 ±38.60	5725.39 ±44.20	7176.56 ±55.42	8710.98 ±68.68
		NeurKItt	179.49 ±5.28	262.28 ±12.08	353.46 ±17.57	565.67 ±25.06	686.25 ±29.78	811.86 ±32.68	1064.24 ±34.25	1327.98 ±35.47
Jacobi	time(s)	GMRES	0.70 ±0.01	1.01 ±0.01	1.32 ±0.01	1.96 ±0.02	2.26 ±0.02	2.56 ±0.02	3.20 ±0.02	3.86 ±0.03
		NeurKItt	0.30 ±0.01	0.40 ±0.02	0.49 ±0.02	0.72 ±0.03	0.85 ±0.04	1.00 ±0.04	1.27 ±0.04	1.56 ±0.04
	iter	GMRES	1378.28 ±10.53	2101.40 ±16.11	2825.34 ±21.76	4274.92 ±32.99	5000.12 ±38.60	5725.39 ±44.20	7176.56 ±55.41	8711.24 ±68.41
		NeurKItt	179.49 ±5.28	262.28 ±12.08	353.45 ±17.60	565.66 ±25.08	686.26 ±29.78	811.87 ±32.68	1064.27 ±34.27	1327.96 ±35.54
BJacobi	time(s)	GMRES	0.21 ±0.00	0.27 ±0.00	0.34 ±0.00	0.46 ±0.00	0.50 ±0.00	0.56 ±0.00	0.67 ±0.00	0.76 ±0.00
		NeurKItt	0.16 ±0.00	0.18 ±0.00	0.25 ±0.01	0.30 ±0.01	0.33 ±0.01	0.37 ±0.01	0.43 ±0.01	0.50 ±0.01
	iter	GMRES	226.80 ±1.10	345.11 ±2.12	464.77 ±1.64	691.76 ±4.50	794.27 ±3.04	897.95 ±2.73	1109.02 ±3.12	1301.74 ±6.50
		NeurKItt	41.30 ±0.54	71.43 ±1.18	108.44 ±4.10	168.11 ±5.96	194.55 ±6.64	221.47 ±6.80	276.24 ±8.79	332.82 ±9.57
SOR	time(s)	GMRES	0.24 ±0.00	0.31 ±0.00	0.38 ±0.00	0.56 ±0.00	0.63 ±0.01	0.70 ±0.01	0.85 ±0.01	0.98 ±0.01
		NeurKItt	0.15 ±0.00	0.22 ±0.01	0.27 ±0.01	0.34 ±0.01	0.39 ±0.01	0.42 ±0.01	0.48 ±0.01	0.57 ±0.01
	iter	GMRES	278.40 ±2.18	426.14 ±3.14	574.14 ±3.85	873.73 ±6.53	1024.37 ±8.74	1173.98 ±11.35	1472.20 ±16.89	1776.41 ±21.33
		NeurKItt	49.15 ±0.95	92.04 ±5.08	131.78 ±2.67	193.15 ±5.85	225.55 ±7.07	257.19 ±6.26	320.99 ±7.98	389.45 ±9.34
ASM	time(s)	GMRES	0.21 ±0.00	0.27 ±0.00	0.32 ±0.00	0.43 ±0.00	0.48 ±0.00	0.53 ±0.00	0.64 ±0.01	0.76 ±0.01
		NeurKItt	0.16 ±0.00	0.18 ±0.00	0.23 ±0.00	0.28 ±0.00	0.33 ±0.01	0.34 ±0.01	0.41 ±0.01	0.47 ±0.01
	iter	GMRES	173.48 ±1.39	260.62 ±1.99	347.40 ±2.19	521.99 ±3.73	609.52 ±4.11	696.81 ±4.73	875.03 ±8.66	1056.19 ±9.31
		NeurKItt	35.89 ±0.42	55.89 ±0.31	86.75 ±1.73	134.46 ±1.29	155.76 ±4.80	176.42 ±4.71	221.61 ±4.96	266.27 ±6.07
ICC	time(s)	GMRES	1.33 ±0.01	2.00 ±0.02	2.66 ±0.02	4.03 ±0.03	4.71 ±0.03	5.40 ±0.03	6.79 ±0.04	8.18 ±0.05
		NeurKItt	0.87 ±0.01	1.14 ±0.01	1.66 ±0.04	2.58 ±0.02	2.80 ±0.03	3.25 ±0.05	3.99 ±0.06	4.86 ±0.07
	iter	GMRES	146.86 ±1.46	223.38 ±1.83	300.63 ±2.27	455.72 ±3.05	533.56 ±3.21	611.70 ±3.53	768.22 ±4.46	925.17 ±5.65
		NeurKItt	34.04 ±0.37	53.20 ±0.42	80.74 ±1.81	125.56 ±0.75	141.98 ±1.39	164.42 ±2.42	204.95 ±2.97	249.09 ±3.62
ILU	time(s)	GMRES	1.29 ±0.01	1.95 ±0.02	2.61 ±0.02	3.97 ±0.03	4.65 ±0.03	5.32 ±0.03	6.68 ±0.04	8.04 ±0.05
		NeurKItt	0.86 ±0.01	1.13 ±0.01	1.64 ±0.04	2.54 ±0.02	2.78 ±0.03	3.22 ±0.05	3.95 ±0.06	4.82 ±0.07
	iter	GMRES	146.86 ±1.46	223.38 ±1.83	300.63 ±2.27	455.72 ±3.05	533.56 ±3.21	611.70 ±3.53	768.22 ±4.46	925.20 ±5.64
		NeurKItt	34.04 ±0.37	53.20 ±0.42	80.74 ±1.81	125.56 ±0.75	141.98 ±1.39	164.42 ±2.42	204.95 ±2.97	249.08 ±3.61

I Do we need to learn the neural network for matrices of a fixed size?

Table 14: Experiments on using subspace prediction module trained on 9604 Darcy Flow dataset to accelerate the datasets with different matrix sizes. Results indicate that NeurKItt is able to accelerate linear systems with different sizes of the matrix when the parametric PDE is given.

Dataset	Tol	None	Jacobi	BJacobi	SOR	ASM	ICC	ILU
Darcy 39204	1e-2	3.05 / 11.00	2.15 / 8.75	1.27 / 4.85	1.41 / 5.20	1.37 / 4.08	2.15 / 3.44	2.14 / 3.44
	1e-4	2.79 / 8.19	1.99 / 6.16	1.42 / 4.47	1.43 / 4.45	1.19 / 3.51	1.29 / 3.09	1.28 / 3.09
	1e-7	2.65 / 7.21	1.72 / 4.91	1.45 / 3.92	1.33 / 3.86	1.27 / 3.27	1.27 / 2.87	1.26 / 2.87
	1e-10	2.39 / 6.78	1.60 / 4.27	1.27 / 3.65	1.37 / 3.62	1.25 / 3.12	1.26 / 2.80	1.24 / 2.80
	1e-12	2.36 / 6.60	1.54 / 4.10	1.31 / 3.57	1.36 / 3.52	1.27 / 3.06	1.27 / 2.80	1.26 / 2.80
Darcy 158404	1e-2	4.83 / 13.80	2.81 / 8.21	1.92 / 6.59	2.25 / 7.02	1.82 / 6.16	2.24 / 5.95	2.43 / 5.95
	1e-4	4.35 / 11.25	2.42 / 6.49	1.80 / 5.06	2.09 / 5.66	1.80 / 4.96	2.14 / 5.01	2.09 / 5.01
	1e-7	3.50 / 8.92	2.06 / 5.35	1.62 / 4.18	1.87 / 4.67	1.66 / 4.10	1.80 / 4.17	1.78 / 4.17
	1e-10	3.15 / 8.17	1.92 / 4.93	1.57 / 3.89	1.69 / 4.17	1.61 / 3.75	1.66 / 3.82	1.64 / 3.82
	1e-12	3.07 / 7.99	2.01 / 5.13	1.55 / 3.81	1.64 / 3.97	1.56 / 3.63	1.62 / 3.69	1.60 / 3.69

An important question is whether a model trained on a fixed-size dataset can be used to predict other matrices under the same PDE problem, which is common in scientific computing. In this section, we attempt to address this question using Darcy Flow dataset. First, we downsample the input function of the larger Darcy Flow dataset to generate new Darcy Flow datasets according to the discretization method mentioned in B. Specifically, we downsample the original dataset with a matrix size of 158,404 to obtain Darcy Flow datasets with matrix sizes of 39,204 and 9,604.

Next, we use the subspace prediction module trained on the Darcy Flow dataset with a matrix size of 9,604 to predict the subspaces of larger datasets. The input to the module is the input function from the larger matrix dataset, which is downsampled to the same size as the input function of the 9,604 matrix dataset, and the output is the predicted subspace of the original matrix dataset size, recovered using cubic spline interpolation.

In our experiments, the principal angles obtained through this method are generally small. For example, the model trained on the 9,604 matrix dataset predicted a principal angle of 0.07 for the test set of the 39,204 matrix dataset and 0.08 for the test set of the 158,404 matrix dataset. The experimental results in Table 14 are based on the acceleration results using the predicted invariant subspaces.

These results indicate that our method can scale across different matrix sizes for the given parametric PDE, which demonstrates that NeurKItt is practical. Additionally, using this approach for subspace prediction significantly reduces the required training time.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction accurately reflect the contribution and scope of the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We refer to Section 7 for the discussion of our limitations.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We have cite the specific paper with the assumptions and proofs for the Theorems used in our paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the pseudocode for our algorithm in Appendix A, and provide the details about experimental settings and hyperparameter of our model in Appendix F. The dataset generation method is provided in Appendix F.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We refer to Section 4 for code repo.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide details about our experiments and settings in Appendix F

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide the stand deviation of average time cost and average iteration count in Appendix H.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the details about experiments compute recources in Appendix F.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Contents in our paper focus on accelerating the linear system solving, which could have potential help to the industry or other field involving linear system solving.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our data and model focus on linear system solving, especially PDE problems, which have no high risk for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have checked this and confirmed the paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the instruction in supplemental material about our code and data.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.