# ACCELERATING EIGENVALUE DATASET GENERATION VIA CHEBYSHEV SUBSPACE FILTER

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Eigenvalue problems are among the most important topics in many scientific disciplines. With the recent surge and development of machine learning, neural eigenvalue methods have attracted significant attention as a forward pass of inference requires only a tiny fraction of the computation time compared to traditional solvers. However, a key limitation is the requirement for large amounts of labeled data in training, including operators and their eigenvalues. To tackle this limitation, we propose a novel method, named **S**orting **C**hebyshev **S**ubspace **F**ilter (**SCSF**), which significantly accelerates eigenvalue data generation by leveraging similarities between operators—a factor overlooked by existing methods. Specifically, SCSF employs truncated fast Fourier transform sorting to group operators with similar eigenvalue distributions and constructs a Chebyshev subspace filter that leverages eigenpairs from previously solved problems to assist in solving subsequent ones, reducing redundant computations. To the best of our knowledge, SCSF is the first method to accelerate eigenvalue data generation. Experimental results show that SCSF achieves up to a $3.5\times$ speedup compared to various numerical solvers.

## 1 INTRODUCTION

Solving eigenvalue problems is an important challenge in fields such as quantum physics (Pfau et al., 2023), fluid dynamics (Schmid, 2010), and structural mechanics (Wen et al., 2022). Traditional numerical solvers, such as the Krylov-Schur algorithm (Stewart, 2002), often suffer from prohibitively high computational costs when tackling complex problems. To overcome these computational challenges, recent advancements in deep learning (Schütt et al., 2017; Li et al., 2020; Luo et al.) have demonstrated remarkable success as one forward pass only necessitates a tiny fraction of the computation time compared to numerical solvers, often in milliseconds.

Despite their success, data-driven approaches face a fundamental limitation: the reliance on labeled datasets. Training neural networks requires large-scale labeled data, which is often generated using computationally expensive traditional methods. It usually takes dozens of hours or even days. For example, the QM9 dataset (Ramakrishnan et al., 2014) contains $1.34 \times 10^5$ molecular data points, each produced by solving Hamiltonian operator eigenvalue problems. These calculations typically employ traditional algorithms, whose computational costs can escalate dramatically with increasing problem complexity, like finer grid resolutions or higher accuracy requirements. This scalability issue represents a significant bottleneck for generating the labeled data needed to train deep learning models. Furthermore, the diversity of scientific problems leads to the need for a unique dataset for each scenario, which further intensifies this challenge of computational intractability. As a result, the high computational expense of generating eigenvalue data severely limits the application of data-driven approaches (Zhang et al., 2023).

In particular, the dataset generation process typically involves six key steps, as illustrated in Figure 1 (left). Among these steps, solving the eigenvalue problem is the most computationally demanding (step 4), accounting for $95\%$ of the total processing cost (Hughes, 2012). Existing data generation methods typically compute the eigenvalues of each matrix in the dataset independently. However, operators in the dataset often share similarities, as they describe related physical phenomena, which can largely simplify and accelerate the eigenvalue-solving process. Existing approaches, however, fail to leverage these similarities, leading to significant computational redundancy. Previous works (Wang et al., 2024; Dong et al., 2024) have demonstrated the potential of leveraging similarity to

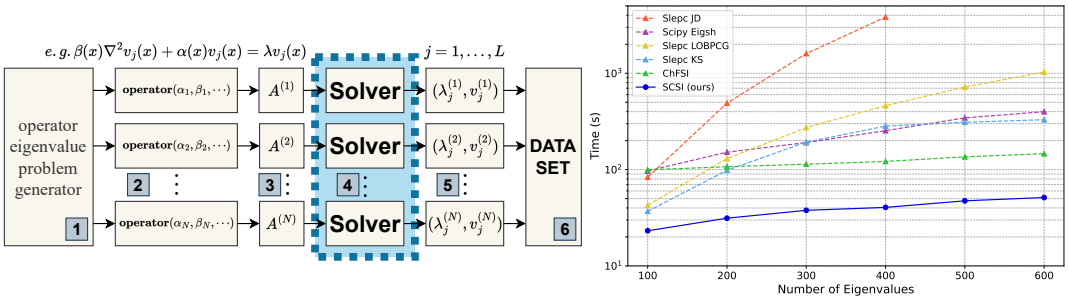


Figure 1: **Left.** Generation process of the eigenvalue dataset: 1. Generate a set of random problem parameters. 2. Derive the corresponding operators based on these parameters. 3. Convert the operators into matrices using discretization methods. 4. Independently solve for the matrix eigenvalues using numerical solvers. 5. Obtain the matrix eigenpairs, converting them into the operator eigenpairs. 6. Assemble the dataset. **Right.** Results of average computation times across various algorithms based on the number of eigenvalues solved on the Helmholtz operator dataset.

significantly reduce generation time of linear system datasets. However, how to effectively exploit matrix similarity to accelerate eigenvalue datasets generating remains an unknown problem.

To address this problem, we introduce a novel data generation approach, named **S**orting **C**hebyshev **S**ubspace **F**ilter (**SCSF**). SCSF is designed to use the eigenpairs of similar problems to reduce redundant computations in the eigenvalue solving process, thereby accelerating eigenvalue dataset generation. Specifically, at the beginning, SCSF employs a sorting algorithm based on truncated Fast Fourier transform (FFT), which arranges these problems efficiently, enhancing the adjacent correlation between problems in the queue and laying the groundwork for sequential solving. Then, SCSF accelerates the convergence of iterations and significantly reduces computation times by constructing a Chebyshev subspace filter, which solves the problem aided by the eigenpairs from previous problem solving. The core design of SCSF is to identify and exploit the close spectral distributions and invariant subspaces within these eigenvalue problems. SCSF coordinates the sequential resolution of these systems rather than treating them as discrete entities. This improved approach not only alleviates the computational demands of the eigenvalue algorithm but also significantly speeds up the generation of training data for data-driven algorithms. We summarize our contributions as follows:

- To the best of our knowledge, SCSF is the first method to accelerate operator eigenvalue dataset generation by exploiting intrinsic operator similarity through a novel sorting and filtering framework.
- By using truncated FFT sorting and the Chebyshev filtered subspace iteration, we introduce a novel approach that transforms dataset generation into sequence eigenvalue problems.
- Comprehensive experiments demonstrate that SCSF substantially reduces the computational cost of eigenvalue dataset generation. As demonstrated in Figure 1 (right), our method achieves up to a $3.5\times$ speedup compared to state-of-the-art solvers.

## 2 Preliminaries

### 2.1 Discretization of Eigenvalue Problem

This section outlines the discretization of operator eigenvalue problems into matrix form. Our method focuses on solving these matrix problems, which is the most time-consuming step in generating the required eigenvalue datasets. As shown in Figure 1 (left), these problems are typically solved by numerical discretization methods such as FDM (Strikwerda, 2004; LeVeque, 2002). These discretization techniques embed the infinite-dimensional Hilbert space of operators into an appropriate finite-dimensional space, thereby transforming operator eigenvalue problems into matrix eigenvalue problems. We provide a simple example to clarify the discussed processes. A detailed process can be found in Appendix C. Specifically, we discuss the case that uses FDM to solve the eigenvalue problem of the two-dimensional Poisson operator, transforming it into a matrix eigenvalue problem:

$$k(x, y)\nabla^2 u(x, y) = \lambda u(x, y). \tag{1}$$

We map the problem onto a $2 \times 2$ grid (i.e., $N_x = N_y = 2$ and $\Delta x = \Delta y$), where both the variable $u_{i,j}$ and the coefficients $k_{i,j}$ follow a row-major order. This setup facilitates the derivation of the matrix eigenvalue equation:

$$\begin{bmatrix} k_{1,1} & 0 & 0 & 0 \\ 0 & k_{1,2} & 0 & 0 \\ 0 & 0 & k_{2,1} & 0 \\ 0 & 0 & 0 & k_{2,2} \end{bmatrix} \begin{bmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \end{bmatrix} = \lambda \begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \end{bmatrix}. \tag{2}$$

By employing various methods to generate the parameter matrices $P = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$. Such as utilizing Gaussian random fields (GRF) or truncated polynomials, we can derive Poisson operators characterized by distinct parameters.

Typically, training a neural network requires a number of data from $10^3$ to $10^5$ (Lu et al., 2019). Such a multitude of eigenvalue systems, derived from the same distribution of operators, naturally exhibit a high similarity (Soodhalter et al., 2020). It is precisely this similarity that is key to the effective acceleration of SCSF. We can conceptualize this as the task of solving a sequential series of matrix eigenvalue problems:

$$A^{(i)} v_j^{(i)} = \lambda_j^{(i)} v_j^{(i)}, \quad j = 1, \cdots, L; \quad i = 1, 2, \cdots, N \tag{3}$$

where $L$ is the number of eigenvalues to be solved, $N$ is the number of eigenvalue problems, the matrix $A^{(i)} \in \mathbb{C}^{n \times n}$, the eigenvector $v_j^{(i)} \in \mathbb{C}^n$,and the eigenvalue $\lambda_j^{(i)} \in \mathbb{C}$ vary depending on the operator. We define the eigenpairs as $(\Lambda^{(i)}, V^{(i)})$, with $\Lambda^{(i)} = \text{diag}(\lambda_1^{(i)}, \ldots, \lambda_L^{(i)})$, $V^{(i)} = [v_1^{(i)} | \cdots | v_L^{(i)}]$, and $|\lambda_1^{(i)}| \leqslant |\lambda_2^{(i)}| \cdots \leqslant |\lambda_L^{(i)}|$.

### 2.2 The Chebyshev Polynomials and Chebyshev Filter

This section details the Chebyshev filter, which is constructed using Chebyshev polynomials. Distinguished by their optimal uniform approximation properties and efficient three-term recurrence relations, these are among the most widely utilized orthogonal polynomials (Mason & Handscomb, 2002; Rivlin, 2020). This filtering technique is particularly effective for our method because it can efficiently construct a polynomial that isolates a target spectral interval, allowing us to use the solution of one eigenvalue problem to significantly accelerate the next.

The Chebyshev polynomials $C_m(t)$ of degree $m$ are defined on $[-1, 1]$ and are expressed as

$$C_m(t) = \cos(m \cos^{-1}(t)), \quad |t| \leq 1. \tag{4}$$

$C_m(t)$ commonly referred to as the Chebyshev polynomial of the first kind, satisfies the following recurrence relation:

$$C_{m+1}(t) = 2tC_m(t) - C_{m-1}(t). \tag{5}$$

For a Hermitian matrix $A \in \mathbb{C}^{n \times n}$ and vectors $Y_0 \in \mathbb{C}^{n \times k}$, we use the three-term recurrence relation that defines Chebyshev polynomials in vector form:

$$C_{m+1}(Y_0) = 2AC_m(Y_0) - C_{m-1}(Y_0), \quad C_m(Y_0) \equiv C_m(A)Y_0. \tag{6}$$

The computation of $C_m(Y_0)$ and the Chebyshev filter is described in Algorithm 1. Let $A'$ denote the previously solved related matrix, with $(\lambda_i', v_i')$ in ascending order, and $\{\lambda_2', \ldots, \lambda_L'\} \in [\alpha, \beta]$. In Algorithm 1, the parameter $\lambda$ is typically approximated by $\lambda_1'$, while $c = \frac{\alpha+\beta}{2}$ and $e = \frac{\beta-\alpha}{2}$ represent the center and half-width of the interval $[\alpha, \beta]$, providing estimates for the spectral distribution of $A$.

**Algorithm 1:** Chebyshev Filter (Berljafa et al., 2015)

**Input:** Matrix $A \in \mathbb{C}^{n \times n}$, vectors $Y_0 \in \mathbb{C}^{n \times k}$, degree $m \in \mathbb{N}$, and parameters $\lambda, c, e \in \mathbb{R}$.
**Output:** Filtered vectors $Y_m = C_m(Y_0)$, where each vector $Y_{m,j}$ is filtered with a Chebyshev polynomial of degree $m$.

$A = (A - cI_n)/e, \quad \sigma_1 = e/(\lambda - c)$;
$Y_1 = \sigma_1 A Y_0$;
**for** $i = 1, \ldots, m-1$ **do**
  $\sigma_{i+1} = 1/(2/\sigma_1 - \sigma_i)$;
  $Y_{i+1,1:m-1} = Y_{i,1:m-1}, \quad Y_{i+1,m:k} = 2\sigma_{i+1} A Y_{i,m:k} - \sigma_{i+1}\sigma_i Y_{i,m:k}$;

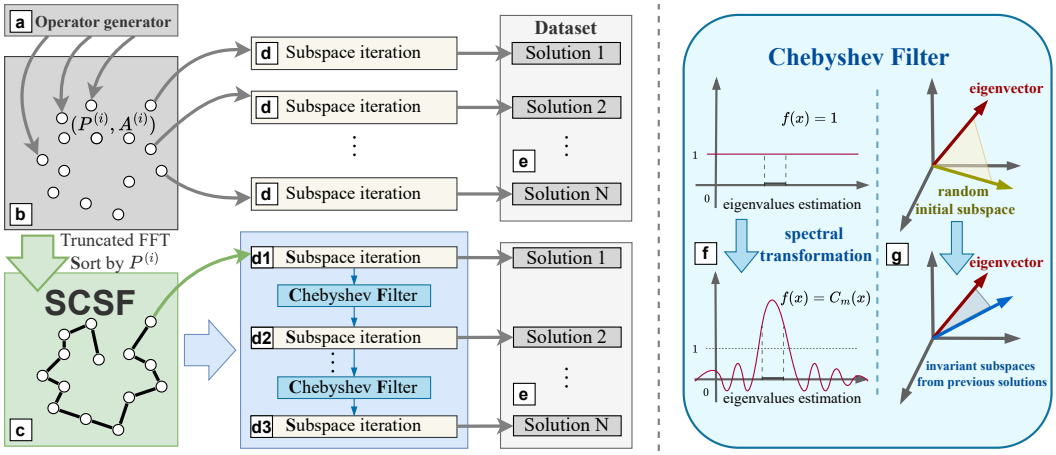


Figure 2: Algorithm Flow Diagram: **a**. Generation of operators to be solved. **b**. Discretization of operators into matrices. **c**. Apply SCSF algorithm to sort matrices, obtaining a sequence with strong correlations. **d**. Other algorithms independently solve eigenvalue problems. **d1, d2, d3**. SCSF algorithm utilizes Chebyshev subspace iterations to sequentially solve the eigenvalue problems. **e**. Assembly of eigenvalue pairs into a dataset. **f**. Amplification of the interval of interest through spectral transformation. **g**. Replacement of initial subspaces with previously solved invariant subspaces.

## 3 METHOD

In this section, we introduce our novel method, named the sorting Chebyshev subspace filter (SCSF), a fast data generation approach that efficiently solves eigenvalue problems by leveraging intrinsic spectral correlations among operators. SCSF incorporates two key components: (1) a truncated fast Fourier transform (FFT)-based approach for efficiently sorting operator eigenvalue problems and (2) the Chebyshev filtered subspace iteration (ChFSI) employed for sequential solving. By integrating these components, SCSF can use spectral information from the previous eigenvalue problem solving to aid the next eigenvalue problem solving, thus accelerating the eigenvalue data generation.

We first introduce the sorting algorithm that leverages the spectral similarities and provides the time complexity analysis in Section 3.1 . Then we give an introduction to the Chebyshev filtered subspace iteration in Section 3.2. Figure 2 shows the overview of our SCSF. Generally, the truncated FFT sorting algorithm ensures that successive matrices in the sequence exhibit close relations. Then ordered sequence enables ChFSI to effectively utilize prior information, thereby accelerating the solution process (Berljafa et al., 2015).

### 3.1 THE SORTING ALGORITHM

To benefit the successive solving sequence of the eigenvalue problem, we need a sorting algorithm that pulls matrices with similar spectral properties, like invariant subspaces, close enough in the solving sequence, so that solving the current matrix in sequence can be easily boosted by the previous solving. Recalling Section 2.1, the matrix $A^{(i)}$ is the operator matrix derived through numerical discretization (e.g., Finite Difference Method) from the parameter matrix $P^{(i)}$ (e.g., the coefficient $k(x, y)$ in the Poisson equation) (Lu et al., 2022; Li et al., 2020). For instance, in the Poisson equation, the values $k_{i,j}$ from $P^{(i)}$ become part of the discretized sparse matrix $A^{(i)}$, as shown in Eq. 2. A naive strategy is to use the Frobenius distance of the parameter matrices $P^{(i)}$ to perform a greedy sort (Wang et al., 2024). And by repeatedly fetching without reservation from the remaining matrix in the dataset, we can reorganize the solving sequence so that the successive solving can benefit from the re-ordered sequence.

However, the main computational cost of such a naive sorting algorithm arises from repeatedly calculating the distances between different matrices $P$, which is directly related to the matrix dimension—that is, the resolution of operators. Existing works (Holmes, 2012; Li et al., 2020) have shown that the key variables that affect operators stem from the low-frequency components of the

**Algorithm 2:** The Truncated FFT Sorting Algorithm

**Input:** Sequence of eigenvalue problems to be solved $A^{(i)} \in \mathbb{C}^{n\times n}$, corresponding parameter matrix $P^{(i)} \in \mathbb{C}^{p\times p}, i = 1, 2, \cdots, N$, $p_0$ is the truncation threshold for low frequencies, and $P_{low}^{(i)} \in \mathbb{C}^{p_0\times p_0}$.

**Output:** Sequence for eigenvalue problems $seq_{mat}$.

```
Initialize the list with sequence seq_0 = {1, 2, ··· , N}, seq_mat is an empty list;
Set i_0 = 1 as the starting point. Remove 1 from seq_0 and append 1 to seq_mat;
for i = 1, ··· , N do
    Let P_low^(i) = Trunc_{p_0}(FFT(P^(i))). Perform truncated FFT on matrix P^(i) to extract
      low-frequency information;
for i = 1, ··· , N − 1 and dis = 1000 do
    for each j in seq_0 do
        dis_j = the Frobenius norm of the difference between P_low^(i_0) and P_low^(j);
        if dis_j < dis then
            dis = dis_j and j_min = j;
    Remove j_min from seq_0, append j_min to seq_mat and set i_0 = j_min;
Get the sequence for eigenvalue problems seq_mat;
```

parameter matrices $P$, while high-frequency components often represent noise or irrelevant data. Based on this insight, to reduce computational overhead during sorting, we first perform a truncated FFT on the parameter matrices to extract the low-frequency information before sorting. We then sort by comparing the Frobenius distances between these low-frequency components.

As shown in Algorithm 2, suppose we have $N$ eigenvalue problems, the parameter matrices $P^{(i)} \in \mathbb{C}^{p\times p}$, and the low-frequency truncated matrices $P_{low}^{(i)} \in \mathbb{C}^{p_0\times p_0}$. The computational complexity of directly using a greedy algorithm is $\mathcal{O}(N^2p^2)$. Our sorting algorithm's complexity consists of two main parts: 1. FFT Computation: The complexity of FFT is $\mathcal{O}(p^2 \log p)$ per matrix. For $N$ matrices, this totals $\mathcal{O}(Np^2 \log p)$. 2. Greedy Sorting: The subsequent greedy sorting algorithm has a complexity of $\mathcal{O}(N^2p_0^2)$. Overall, the total complexity is $\mathcal{O}(N^2p_0^2 + Np^2 \log p)$. Since $p_0 \ll p$ and $p \ll N$, our sorting algorithm effectively reduces computational cost. In addition, we conducted a qualitative theoretical analysis of the truncated FFT to demonstrate its rationality. Details can be found in Appendix F.

### 3.2 Chebyshev Filtered Subspace Iteration

After the sorting algorithm, we obtain a sequence of eigenvalue problems that exhibit strong correlations between consecutive problems. We employ the Chebyshev filtered subspace iteration (Manteuffel, 1977; Saad, 2011; Winkelmann et al., 2019; Berljafa et al., 2015) that leverages the eigenpairs $(\Lambda^{(i-1)}, V^{(i-1)})$ of the previous problem $A^{(i-1)}$ to accelerate the iterative convergence of the subsequent problem $A^{(i)}$, thereby significantly enhancing computational performance. We focus on the most common scenario in eigenvalue problems where the operator is self-adjoint; in this case, the corresponding matrix $A$ is Hermitian.

Algorithm 3 outlines the process of ChFSI for solving the $i$-th eigenvalue problem $A^{(i)}$ $(1 < i \leqslant N)$ where $L$ eigenvalues need to be solved. The initial approximate invariant subspace $V^{(i-1)}$ and spectral distribution $\Lambda^{(i-1)}$ are derived from the eigenvectors and eigenvalues of the previous problem $A^{(i-1)}$ in the sequence. This initialization strategy, illustrated in Figure 2 (g), acts as a "warm start", where a high-quality initial subspace significantly reduces the computational cost of subsequent iterations. The parameter $m$ denotes the polynomial degree in the filter function, e.g., $m = 20$. As depicted in Figure 2 (f), the Chebyshev filter modifies the operator's spectrum to amplify the target region, effectively transforming the problem into an equivalent one that is easier to solve. For the first eigenvalue problem $A^{(1)}$ in the sequence, the initial iterative subspace $\tilde{V}_0$ and initial spectrum $\tilde{\Lambda}_0$ are randomly generated.

**Algorithm 3:** Chebyshev Filtered Subspace Iteration

**Input:** Eigenvalue problem $A^{(i)}$, eigenpairs $(\Lambda^{(i-1)}, V^{(i-1)})$ of the previous eigenvalue problem $A^{(i-1)}$ where $\Lambda^{(i-1)} = \text{diag}(\lambda_1^{(i-1)}, \ldots, \lambda_L^{(i-1)})$, $V^{(i-1)} = [v_1^{(i-1)}| \cdots |v_L^{(i-1)}]$, and filter degree $m$.

**Output:** Wanted eigenpairs $(\Lambda^{(i)}, V^{(i)})$.

1 Initialize empty arrays/matrices $(\tilde{\Lambda}, \tilde{V})$, and set $\tilde{\Lambda}_0 = \Lambda^{(i-1)}$, $\tilde{V}_0 = V^{(i-1)}$;
2 **repeat**
3 Apply Chebyshev filter: $\tilde{V}_0 = C_m(\tilde{V}_0)$;
4 Perform QR orthonormalization on $QR = [\tilde{V}|\tilde{V}_0]$;
5 Compute Rayleigh quotient $G = Q_0^\top A^{(i)} Q$;
6 Solve the reduced problem $GW = W\tilde{\Lambda}_0$, and update $\tilde{V}_0 = \tilde{V}_0 W$;
7 Lock converged eigenpairs into $(\tilde{\Lambda}, \tilde{V})$;
8 **until** *the number of converged eigenpairs* $\geq L$;
9 Return eigenpairs $(\Lambda^{(i)}, V^{(i)}) = (\tilde{\Lambda}, \tilde{V})$ ;

In `line 3`, the Chebyshev filter is applied using the vector form of Chebyshev polynomials; details can be found in the preliminaries Section 2.2. After the Chebyshev filtering step, the vector block $\tilde{V}_0$ spanning the invariant subspace may become linearly dependent. To prevent this, orthonormalization is performed (`line 4`) using QR decomposition based on Householder reflectors. `Line 5` computes the Rayleigh quotient of $A^{(i)}$ using the orthonormalized $\tilde{V}_0$, projecting the eigenvalue problem onto a subspace that approximates the desired eigenspace. In `line 6`, the reduced eigenvalue problem is diagonalized, and the computed eigenvectors are projected back to the original problem. At the end of the Rayleigh-Ritz step, relative residuals of the computed eigenvectors are calculated; converged eigenpairs are locked, and non-converged vectors are set to be filtered again (`line 7`).

Assuming $m$ is the degree of the polynomial, $n$ is the dimension of the matrix $A$, and $L$ is the number of eigenvalues to be solved, the computational complexity per iteration comprises: 1. Chebyshev filter: $\mathcal{O}(mn^2L)$ 2. QR factorization: $\mathcal{O}(nL^2)$ 3. Rayleigh-Ritz procedure: $\mathcal{O}(n^2L + nL^2 + L^3)$ 4. Residuals check: $\mathcal{O}(n^2L)$ . Since $m \gg 1$ and $n \gg L$, the Chebyshev filtering step is the most computationally intensive.

The acceleration of the Chebyshev filtered subspace iteration heavily depends on selecting approximate invariant subspaces and eigenvalues that promote rapid convergence in subsequent iterations. Proper sorting amplifies their impact, reducing the number of iterations required. This underscores the critical importance of the sorting algorithm in our method.

# 4 Experiment

## 4.1 Experimental Settings

To comprehensively assess the performance of our approach SCSF against other algorithms, we conducted extensive experiments, each simulating the generation of an operator eigenvalue dataset. We primarily compared the average computation times across different numbers of eigenvalues solved and various matrix sizes. These tests encompassed four distinct datasets and five mainstream eigenvalue solving algorithms, with SCSF consistently delivering commendable results. The detailed data is provided in Appendix E.1, and the related work is discussed in Appendix B.

**Baseline**. Our focus solves the eigenvalue problem of matrices derived from self-adjoint differential operators, typically consisting of large Hermitian matrices. We benchmarked against the following mainstream algorithms implemented in libraries widely used: 1. Eigsh from SciPy (implicitly restarted Lanczos method) (Virtanen et al., 2020), 2. Locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm from SLEPc (Knyazev, 2001; Hernandez et al., 2009), 3. Krylov-Schur (KS) algorithm from SLEPc (Stewart, 2002), 4. Jacobi-Davidson (JD) algorithm from SLEPc (Sleijpen & Van der Vorst, 2000), 5. Chebyshev filtered subspace iteration (ChFSI) (Berljafa et al., 2015) with random initialization. For detailed information, please refer to Appendix D.1.

Table 1: Comparison of average computation times (in seconds) for eigenvalue problems using various algorithms. The first row lists different algorithms, the first column details the datasets, including matrix dimensions and solution precisions (relative residual), and the second column shows the number of eigenvalues $L$ computed for each matrix. The best algorithm is in **bold**. The symbol '-' denotes the result of a method that fails to converge under the given setting.

| Dataset | $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|---|
| Poisson | 200 | 14.20 | 73.03 | 23.76 | 270.2 | 24.00 | **12.85** |
| 2500 | 300 | 26.27 | 151.5 | 45.95 | 920.8 | 38.03 | **25.61** |
| 1e-12 | 400 | 36.86 | 265.3 | 72.32 | 2691 | 57.41 | **33.91** |
| Ellipse | 200 | 41.82 | 139.2 | 61.77 | 414.3 | 43.90 | **24.08** |
| 4900 | 300 | 62.47 | 264.1 | 110.5 | 1446 | 60.69 | **29.88** |
| 1e-10 | 400 | 87.19 | 459.7 | 188.7 | 3386 | 67.13 | **34.60** |
| Helmholtz | 200 | 151.7 | 129.9 | 98.34 | 489.6 | 107.1 | **31.31** |
| 6400 | 400 | 253.5 | 460.4 | 283.0 | 3829 | 121.5 | **40.52** |
| 1e-8 | 600 | 398.8 | 1031 | 329.6 | - | 146.2 | **51.32** |
| Vibration | 200 | 397.9 | 333.7 | 272.0 | 1230 | 300.8 | **85.70** |
| 10000 | 400 | 635.6 | 1170 | 768.8 | - | 310.5 | **107.2** |
| 1e-8 | 600 | 1037 | 2716 | 857.8 | - | 382.3 | **131.4** |

**Datasets**. To explore the adaptability of the algorithm across different matrix types, we investigate four distinct operator eigenvalue problems: 1. Generalized Poisson operator; 2. Second-order elliptic partial differential operator; 3. Helmholtz operator; 4. Fourth-order vibration equation. For a thorough description of the datasets and their generation, please refer to Appendix D.2.

All experiments focus on computing the smallest $L$ eigenvalues in absolute value and their corresponding eigenvectors. For the runtime environment, experimental parameters, and parallelism setup, please see the Appendixes D.3, D.4, and D.6. The hyperparameter analysis experiments, runtimes for various components of SCSF, the reliability of data generated by traditional algorithms, can be found in Appendixex E.4, E.3, and E.5.

We note that all experiments use relative residual as the metric for solution precision, with its definition provided in Appendix D.5. SCSF is a numerical algebra algorithm that allows for adjustable solution precision as needed. It is purely an acceleration technique and does not alter the solution results at the specified precision. The solution precision for all experiments is set to at least 1e-8, which is significantly higher than the typical relative error range of neural networks (1e-1 to 1e-5), making it effectively a ground truth. Therefore, the datasets generated by different numerical algorithms will not affect the training performance of neural networks.

### 4.2 Main Experiment

Table 1 showcases selected experimental data. From this table, we can infer several conclusions: First, across all settings, our SCSF consistently has the lowest computation cost. The most significant improvements appeared in the Helmholtz dataset, where SCSF demonstrated speedups of $8\times$, $20\times$, $6\times$, $95\times$, and $3.5\times$ compared to Eigsh, LOBPCG, KS, JD, and ChFSI algorithms, respectively. These results confirm that SCSF effectively reduces inherent redundancies in sequential eigenvalue problems, substantially accelerating operator eigenvalue dataset generation.

Moreover, as the number of eigenvalues $L$ solved per matrix increases, the speed advantage of SCSF over other algorithms becomes more pronounced. For instance, on the second-order elliptic operator dataset, when solving for 200 eigenvalues, SCSF is

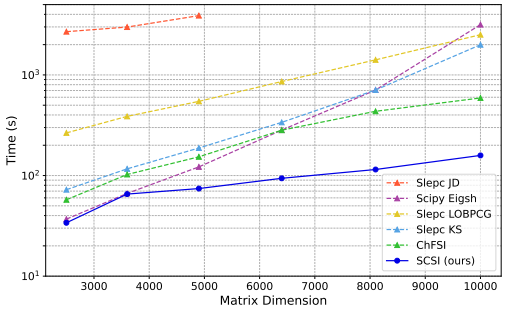


Figure 3: Plot of average computation time versus matrix dimension for solving 400 eigenvalues with a precision of 1e-12 on the generalized Poisson operator dataset.

2.5 times faster than the Krylov-Schur method and 5.5 times faster at 400 eigenvalues. This efficiency stems from SCSF inheriting approximate invariant subspaces from previous solutions, effectively leveraging available information to expand the initial search space. Consequently, SCSF requires minimal additional iterations as $L$ increases, resulting in modest computation time growth.

Besides, the performance disparity across different datasets is significant. For example, on the generalized Poisson operator dataset, SCSF is only about 10% faster than Eigsh, yet it leads by 4-7 times on the Helmholtz dataset. This difference can be attributed to the numerical properties of different operators and the matrix assembly formats, which directly influence algorithmic performance.

We also conducted additional experiments to show that the impact of the matrix dimension is also significant. Results are shown in Figure 3, SCSF performs noticeably better as matrix dimensions increase. Below the matrix dimension of 3600, SCSF and Eigsh show comparable efficiency. However, beyond 5000, SCSF significantly outperforms Eigsh and other algorithms. For more details about matrix dimension influence, we refer to the results in Appendix E.2.

This phenomenon can be explained through operator matrix approximation. A fixed operator has invariant eigenvalues and eigenfunctions. Varying matrix dimensions correspond to embedding the operator in different finite-dimensional linear spaces. For a fixed number of eigenvalues $L$, larger matrices yield more accurate approximations of the true eigenvalues. That is, larger matrices reduce computational noise and enhance operator similarity visibility, enabling SCSF to utilize these similarities more effectively for superior performance. For comparisons with neural networks, similarity impact, and edge-case performance, see Appendices E.6, E.7, and E.8.

### 4.3 Efficacy of Chebyshev Subspace Filter

Table 2: Impact of initial subspace modifications on average computation time (in seconds) for different algorithms. '*' denotes the modified version. The first row lists algorithms, and the first column shows the number of eigenvalues $L$ computed. The best algorithm is in **bold**, and '-' indicates the result of a method that fails to converge under the given setting.

| $L$ | Eigsh | Eigsh* | LOBPCG | LOBPCG* | KS | KS* | JD | JD* | SCSF (ours) |
|---|---|---|---|---|---|---|---|---|---|
| 200 | 151.7 | 150.2 | 129.9 | 95.9 | 98.34 | 100.6 | 489.6 | 760.1 | **31.31** |
| 300 | 208.8 | 206.3 | 270.1 | 199.8 | 179.9 | 185.2 | 1803 | 3101 | **38.67** |
| 400 | 253.5 | 249.1 | 460.4 | 362.1 | 283.0 | 292.2 | 3829 | 6374 | **40.52** |
| 500 | 324.6 | 315.3 | 717.3 | 573.7 | 314.2 | 317.4 | - | - | **46.70** |
| 600 | 398.8 | 394.7 | 1031 | 866.0 | 329.6 | 335.7 | - | - | **51.32** |

To analyze the efficacy of the Chebyshev subspace filter, we conducted the following experiments. After sorting, the initial vector or subspace for the existing algorithms was set to the eigenvectors from the previous problem (the modified version '*'). We compared the computational time across different methods. All experiments were conducted on the Helmholtz operator dataset, with a matrix dimension of 6400 and a tolerance of 1e-8. The results are shown in Table 2.

First, the computation time for SCSF in all experiments was minimal, clearly demonstrating the efficacy of the Chebyshev subspace filter. This also highlights that the Chebyshev subspace filter is the optimal choice for leveraging problem similarity to reduce redundancy.

Second, the impact of initial setup modifications varied across algorithms: 1. LOBPCG accelerated significantly due to its subspace-based logic, similar to SCSF, where initialization strongly influences convergence. 2. Eigsh and KS remained largely unaffected as they rely on initial vectors and Krylov iteration, making problem similarity less impactful. 3. JD showed a performance decline. This is because its performance is sensitive to the size of the initial subspace. Our modification altered the default dimension of the initial subspace.

### 4.4 Efficacy of Sorting Algorithms

We analyze the performance of the sorting algorithm module from two perspectives: 1. Comparing the performance of SCSF algorithm with and without 'sorting' as shown in Table 3. 2. Evaluating the

Table 3: Performance comparison of SCSF with and without sorting. The first column lists the number of eigenvalues $L$ computed, while subsequent columns display average computation times, average iteration counts, total Flop counts, and filter Flop counts. Experiments used the matrix dimension of 2500 and precision 1e-12 on the generalized Poisson operator dataset.

| $L$ | Time (s) | | Iteration | | Flops | | Filter Flops | |
|---|---|---|---|---|---|---|---|---|
| | w/o sort | sort | w/o sort | sort | w/o sort | sort | w/o sort | sort |
| 20 | 8.248 | 2.971 | 19.70 | 9.880 | 519.7 | 298.4 | 485.8 | 280.8 |
| 100 | 14.18 | 9.891 | 18.77 | 15.38 | 1984 | 1332 | 1798 | 970.1 |
| 200 | 18.45 | 12.85 | 36.30 | 33.67 | 4459 | 3944 | 3654 | 3192 |
| 300 | 34.59 | 25.61 | 47.50 | 39.18 | 8967 | 7544 | 6985 | 5702 |
| 400 | 42.60 | 33.91 | 47.43 | 45.18 | 12022 | 11182 | 9087 | 8338 |

effectiveness of different sorting algorithms as detailed in Tables 4 and 5. We note that if the setting is 'w/o sort', SCSF is approximately equivalent to directly using the Chebyshev subspace filter. Unlike the ChFSI used in the main experiments, the initialization of each solve in the 'w/o sort' SCSF is set based on the information obtained from solving the previous problem (following the default unsorted sequence).

Firstly, Table 3 indicates that incorporating sorting can improve SCSF speed to 1.3 to 2.8 times, reduce the number of iterations by 5% to 50%, and decrease total Flops by 7% to 43%. The effect of sorting is more pronounced with smaller numbers of solutions $L$. This is because when $L$ is large, the inherited subspace already contains most of the necessary correlation information, diminishing the impact of sorting. Moreover, the Flops in the Filter component constitute over 70% of SCSF's computational load. A detailed time analysis of different aspects of SCSF can be found in Appendix E.3. Additionally, the 'w/o sort' SCSF achieves a computational speedup of 1.2 to 1.5 times compared to the ChFSI used in the main experiments. The primary difference lies in their initialization strategies: ChFSI uses random initialization for each solve, whereas the 'w/o sort' SCSF leverages information from the previous problem for initialization. This indicates that, even without sorting, there is a certain level of similarity between problems in the dataset. Such similarity can effectively accelerate the solving process.

Table 4: Comparison of average computation times (in seconds) for different sorting algorithms, with the first column indicating dataset size. Experiments used the matrix dimension of 6400 on the Helmholtz dataset.

| Size | Greedy | Truncated FFT Sort (ours) | | |
|---|---|---|---|---|
| | Total | FFT | Greedy | Total |
| $10^2$ | 0.114 | 0.0016 | 0.0147 | 0.0163 |
| $10^3$ | 7.328 | 0.0164 | 1.421 | 1.438 |
| $10^4$ | 592.7 | 0.1658 | 150.9 | 151.1 |

Secondly, as shown in Table 4, our designed truncated FFT sorting algorithm incurs significantly lower time cost compared to the complete greedy sorting in SKR (Wang et al., 2024), with its benefits becoming more pronounced as the dataset size increases. In the truncated FFT sorting algorithm, the FFT contributes minimally to computational overhead but significantly reduces the time required for subsequent greedy sorting. Table 5 validates the accuracy of our efficient sorting against the expensive greedy approach. To ensure fairness, we averaged the results over 100 independent runs to eliminate fluctuations. The results for 'Greedy' and 'Ours' are identical because the sequences produced by both algorithms are over 98% identical. Consequently, the downstream solver incurs exactly the same operation cost, demonstrating that our low-cost sorting achieves the same acceleration performance as the high-cost greedy approach.

Table 5: Comparison of average computation times and iteration counts for different sorting algorithms using SCSF. Experiments used the matrix dimension of 6400 on the Helmholtz dataset, precision 1e-8, and targeting 400 eigenvalues.

| | w/o sort | Greedy | Ours |
|---|---|---|---|
| Time (s) | 66.66 | 40.52 | 40.52 |
| Iteration | 10.4 | 5.5 | 5.5 |

Furthermore, our experiments demonstrate that the choice of the truncation threshold, $p_0$, is robust. We found that excellent performance is achieved with a relatively small $p_0$ (e.g., $p_0 = 20$), a value much smaller than the parameter matrix dimension $p$. This is because performance stabilizes once a

sufficient number of low-frequency components are captured, making it unnecessary to retain a large number of components. For related experiments, please refer to Appendix E.4.3.

## 5 Conclusions and Future Work

**Conclusions** In this paper, we addressed the critical bottleneck of generating large-scale eigenvalue datasets for training neural operators. We introduced SCSF, the first method to accelerate eigenvalue dataset generation by exploiting operator similarity. By integrating a truncated FFT sorting algorithm with a Chebyshev subspace filter, SCSF transforms the generation task into an efficient sequential solving problem. Our method achieves up to a 3.5$\times$ speedup over traditional solvers, significantly reducing computational redundancy. By lowering a key barrier to entry, SCSF provides a valuable tool for advancing research in the AI for Science community.

**Limitations and Future Work** Despite SCSF's demonstrated efficiency, we have identified three key directions for future development. First, we aim to extend the current framework from Hermitian linear operators to non-Hermitian and non-linear operators. Second, the correlation between operators is currently measured using the Frobenius norm of their parameter matrices. We plan to explore more effective, problem-specific metrics rather than relying on a fixed one. Finally, SCSF's effectiveness hinges on a continuous mapping between parameters and operators, rendering it ineffective when this mapping is discontinuous. Future work will involve measuring correlation directly from the operators or their matrix representations to bypass the challenges of discontinuous parameterization.

## REFERENCES


Khaled Akkad and David He. A dynamic mode decomposition based deep learning technique for prognostics. *Journal of Intelligent Manufacturing*, 34(5):2207–2224, 2023.

Daniel J Alford-Lago, Christopher W Curtis, Alexander T Ihler, and Opal Issan. Deep learning enhanced dynamic mode decomposition. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(3), 2022.

Amartya S Banerjee, Lin Lin, Wei Hu, Chao Yang, and John E Pask. Chebyshev polynomial filtered subspace iteration in the discontinuous galerkin method for large-scale electronic structure calculations. *The Journal of chemical physics*, 145(15), 2016.

Albert P Bartók, Sandip De, Carl Poelking, Noam Bernstein, James R Kermode, Gábor Csányi, and Michele Ceriotti. Machine learning unifies the modeling of materials and molecules. *Science advances*, 3(12):e1701816, 2017.

Mario Berljafa, Daniel Wortmann, and Edoardo Di Napoli. An optimized and scalable eigensolver for sequences of eigenvalue problems. *Concurrency and Computation: Practice and Experience*, 27(4):905–922, 2015.

Lipman Bers, Fritz John, and Martin Schechter. *Partial differential equations*. American Mathematical Soc., 1964.

L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732, 2009.

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for pde modeling. *arXiv preprint arXiv:2209.04934*, 2022.

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52(1):477–508, 2020.

Stefan Chmiela, Alexandre Tkatchenko, Huziel E Sauceda, Igor Poltavsky, Kristof T Sch¨utt, and Klaus-Robert M¨uller. Machine learning of accurate energy-conserving molecular force fields. *Science advances*, 3(5):e1603015, 2017.

Zhijie Deng, Jiaxin Shi, and Jun Zhu. Neuralef: Deconstructing kernels by deep neural networks. In *International Conference on Machine Learning*, pp. 4976–4992. PMLR, 2022.

Huanshuo Dong, Hong Wang, Haoyang Liu, Jian Luo, and Jie Wang. Accelerating pde data generation via differential operator action in solution space. *arXiv preprint arXiv:2402.05957*, 2024.

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

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

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.

Trygve Helgaker, Poul Jorgensen, and Jeppe Olsen. *Molecular electronic-structure theory*. John Wiley & Sons, 2013.

V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal. A survey of software for sparse eigenvalue problems. Technical Report STR-6, Universitat Politècnica de València, 2009. Available at https://slepc.upv.es.

Philip Holmes. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

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

Tomoharu Iwata and Yoshinobu Kawahara. Neural dynamic mode decomposition for end-to-end modeling of nonlinear dynamics. *Journal of Computational Dynamics*, 10(2):268–280, 2023.

Anubhav Jain, Joseph Montoya, Shyam Dwaraknath, Nils ER Zimmermann, John Dagdelen, Matthew Horton, Patrick Huck, Donny Winston, Shreyas Cholia, Shyue Ping Ong, et al. The materials project: Accelerating materials design through theory-driven data and tools. *Handbook of Materials Modeling: Methods: Theory and Modeling*, pp. 1751–1784, 2020.

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

Scott Kirklin, James E Saal, Bryce Meredig, Alex Thompson, Jeff W Doak, Muratahan Aykol, Stephan Rühl, and Chris Wolverton. The open quantum materials database (oqmd): assessing the accuracy of dft formation energies. *npj Computational Materials*, 1(1):1–15, 2015.

Charles Kittel and Paul McEuen. *Introduction to solid state physics*. John Wiley & Sons, 2018.

Andrew V Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM journal on scientific computing*, 23(2):517–541, 2001.

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.

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

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.

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

Lu 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.

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.

Jian Luo, Jie Wang, Hong Wang, Zijie Geng, Hanzhu Chen, Yufei Kuang, et al. Neural krylov iteration for accelerating linear system solving. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Jian Luo, Jie Wang, Hong Wang, Zijie Geng, Hanzhu Chen, Yufei Kuang, et al. Neural krylov iteration for accelerating linear system solving. *Advances in Neural Information Processing Systems*, 37:128636–128667, 2024.

Thomas A Manteuffel. The tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik*, 28:307–327, 1977.

John C Mason and David C Handscomb. *Chebyshev polynomials*. Chapman and Hall/CRC, 2002.

Stephan Mohr, William Dawson, Michael Wagner, Damien Caliste, Takahito Nakajima, and Luigi Genovese. Efficient computation of sparse matrix functions for large-scale electronic structure calculations: The chess library. *Journal of Chemical Theory and Computation*, 13(10):4684–4698, 2017.

Takaaki Murata, Kai Fukami, and Koji Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882:A13, 2020.

David Pfau, Simon Axelrod, Halvard Sutterud, Ingrid von Glehn, and James S Spencer. Natural quantum monte carlo computation of excited states. *arXiv preprint arXiv:2308.16848*, 2023.

Andreas Pieper, Moritz Kreutzer, Andreas Alvermann, Martin Galgon, Holger Fehske, Georg Hager, Bruno Lang, and Gerhard Wellein. High-performance implementation of chebyshev filter diagonalization for interior eigenvalue computations. *Journal of Computational Physics*, 325:226–243, 2016.

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

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

Theodore J Rivlin. *Chebyshev polynomials*. Courier Dover Publications, 2020.

Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters*, 108(5):058301, 2012.

J Jon Ryu, Xiangxiang Xu, HS Erol, Yuheng Bu, Lizhong Zheng, and Gregory W Wornell. Operator svd with neural networks via nested low-rank approximation. *arXiv preprint arXiv:2402.03655*, 2024.

Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1): 13890, 2017.

Gerard LG Sleijpen and Henk A Van der Vorst. A jacobi–davidson iteration method for linear eigenvalue problems. *SIAM review*, 42(2):267–293, 2000.

Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8(4):3192–3203, 2017.

Kirk M Soodhalter, Eric de Sturler, and Misha E Kilmer. A survey of subspace recycling iterative methods. *GAMM-Mitteilungen*, 43(4):e202000016, 2020.

Gilbert W Stewart. A krylov–schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, 2002.

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

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

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.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

Jan Winkelmann, Paul Springer, and Edoardo Di Napoli. Chase: Chebyshev accelerated subspace iteration eigensolver for sequences of hermitian eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 45(2):1–34, 2019.

Yaqiang Xue, Guoyong Jin, Hu Ding, and Mingfei Chen. Free vibration analysis of in-plane functionally graded plates using a refined plate theory and isogeometric approach. *Composite Structures*, 192:193–205, 2018.

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.

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.

Yunkai Zhou and Yousef Saad. A chebyshev–davidson algorithm for large symmetric eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 29(3):954–971, 2007.

Yunkai Zhou, Yousef Saad, Murilo L Tiago, and James R Chelikowsky. Parallel self-consistent-field calculations via chebyshev-filtered subspace acceleration. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 74(6):066704, 2006a.

Yunkai Zhou, Yousef Saad, Murilo L Tiago, and James R Chelikowsky. Self-consistent-field calculations using chebyshev-filtered subspace iteration. *Journal of Computational Physics*, 219 (1):172–184, 2006b.

Yunkai Zhou, James R Chelikowsky, and Yousef Saad. Chebyshev-filtered subspace iteration method free of sparse diagonalization for solving the kohn–sham equation. *Journal of Computational Physics*, 274:770–782, 2014.

# A USAGE OF LLMS

Throughout the preparation of this manuscript, Large Language Models (LLMs) were utilized as a writing and editing tool. Specifically, we employed LLMs to improve the clarity and readability of the text, refine sentence structures, and correct grammatical errors. All final content, including the core scientific claims, experimental design, and conclusions, was conceived and written by us, and we take full responsibility for the final version of this paper.

# B RELATED WORK

## B.1 EIGENVALUE DATASETS AND NEURAL EIGENVALUE METHODS

Eigenvalue datasets are widely utilized in neural eigenvalue methods. In molecular chemistry research, eigenvalue algorithms are commonly employed to determine critical molecular properties, such as orbital energy levels (Kittel & McEuen, 2018). These properties form the foundation of datasets and are obtained by solving the eigenvalue problem of the Schrödinger equation and the Hamiltonian operator (Helgaker et al., 2013). Prominent datasets in this domain include QM7 (Blum & Reymond, 2009), QM9 (Ramakrishnan et al., 2014), ANI-1 (Smith et al., 2017), and MD17 (Chmiela et al., 2017). In materials science, eigenvalue algorithms are often applied to solve for electronic band structures and density of states in materials. Representative datasets in this field include the materials project (Jain et al., 2020) and OQMD (Kirklin et al., 2015). These datasets have been extensively used to train and validate neural eigenvalue methods (Schütt et al., 2017; Bartók et al., 2017; Rupp et al., 2012), driving advancements in molecular property prediction and materials design. In fluid dynamics and structural mechanics, eigenvalue algorithms are frequently utilized for modal analysis. Recently, many data-driven modal analysis algorithms have emerged, requiring eigenvalue datasets corresponding to differential operators for training (Murata et al., 2020; Iwata & Kawahara, 2023; Alford-Lago et al., 2022; Brunton et al., 2020; Akkad & He, 2023). Additionally, some studies leverage operator eigenvalue datasets to optimize algorithms. For instance, Luo et al. accelerates the solution of linear systems by predicting the eigenfunctions of operators.

## B.2 EIGENVALUE DATA GENERATION ALGORITHMS

Training data-driven algorithms require a large amount of labeled eigenvalue data. Typically, the generation of these high-precision data is obtained by traditional algorithms. In the field of computational mathematics, solving operator eigenvalue problems often involves utilizing various discretization methods such as finite difference methods (FDM) (Strikwerda, 2004), finite element methods (FEM) (Hughes, 2012; Johnson, 2012). These discretization methods transform operator eigenvalue problems into matrix eigenvalue problems, which are then solved using the corresponding matrix algorithms. For larger matrices, the Krylov-Schur algorithm (Stewart, 2002), Jacobi-Davidson (Sleijpen & Van der Vorst, 2000), and locally optimal block preconditioned conjugate gradient (LOBPCG) (Knyazev, 2001) are among the most frequently employed algorithms (Golub & Van Loan, 2013).

Nonetheless, traditional methods were not designed for dataset generation, resulting in high computational costs, which have become a significant barrier to the advancement of data-driven approaches (Zhang et al., 2023; Hao et al., 2022). Recent data augmentation research (Brandstetter et al., 2022; Liu et al., 2023) has led to the development of methods that preserve symmetries and conservation laws, enhancing model generalization and data efficiency. Wang et al. (2024); Dong et al. (2024) report acceleration in the process of solving linear equations, thereby speeding up the generation of PDE datasets.

However, these improvements largely focus on neural networks or the rapid solution of linear system-based PDEs, without discussing optimizations in the generation of eigenvalue datasets.

## B.3 CHEBYSHEV FILTER TECHNIQUE

The Chebyshev filter technique originates from polynomial approximation theory, where the core concept involves using Chebyshev polynomials to accelerate the convergence of eigenvalues (Zhou & Saad, 2007). This technique constructs a polynomial filter that selectively amplifies spectral components in a specified interval, thereby speeding up the solution of specific eigenvalues. This

technique is particularly effective in dealing with sequence eigenvalue problems (Saad, 2011; Zhou et al., 2006a) and has been applied in various contexts, such as stability analysis in electronic structure (Pieper et al., 2016; Banerjee et al., 2016) and quantum chemical computations (Mohr et al., 2017; Zhou et al., 2014; 2006b).

Due to the chaotic and disordered nature of eigenvalue problems in the dataset, directly applying the Chebyshev filter technique fails to accelerate dataset generation. To further adapt this technique to the generation of operator eigenvalue datasets, we have developed a specialized sorting algorithm that transforms dataset generation into sequence eigenvalue problems. Throughout the solving process, eigenpairs obtained from previous solutions are used to construct Chebyshev filters, accelerating subsequent solutions.

# C From Differential Operator to Matrix Eigenvalue Problem: An Example

## C.1 Overview

The general methodology for solving the eigenvalue problems of differential operators numerically, employing techniques such as Finite Difference Method (FDM), Finite Element Method (FEM), and Spectral Method, can be delineated through the following pivotal steps (Strikwerda, 2004; Hughes, 2012; Johnson, 2012; LeVeque, 2002):

1. Mesh Generation: This step involves dividing the domain, over which the differential operator is defined, into a discrete grid. The grid could be composed of various shapes, including squares, triangles, or more complex forms, depending on the problem's geometry.

2. Operator Discretization: The differential operator is transformed into its discrete counterpart. Essentially, this maps the operator from an infinite-dimensional Hilbert space to a finite-dimensional representation.

3. Matrix Assembly: In this phase, the discretized operator is represented in a matrix form. For linear differential operators, this involves creating a system of matrix eigenvalue problems. For nonlinear operators, iterative methods akin to Newton's iteration are employed, transforming the problem into a sequence of matrix eigenvalue problems.

4. Applying Boundary Conditions: This involves discretizing and applying boundary conditions specific to the differential operator in question, which are then incorporated into the matrix system.

5. Solving the Matrix Eigenvalue Problem: This stage, often the most computationally intensive, entails solving the matrix for its eigenvalues and eigenvectors, which correspond to the eigenvalues and eigenfunctions of the original differential operator.

6. Obtaining the Numerical Solution: The final step involves mapping the obtained numerical solutions back onto the original domain, analyzing them for accuracy and stability, and interpreting them in the context of the initial problem.

## C.2 Example

To illustrate how the FDM can transform the wave equation into a system of matrix eigenvalue problems, let's consider a concrete and straightforward example. Assume we aim to solve a one-dimensional wave equation's operator eigenvalue problem, expressed as

$$-\frac{d^2u}{dx^2} = \lambda u,$$

over the interval $[0, L]$. The boundary conditions are $u(0) = u(L) = 0$, signifying fixed-end conditions. In this context, $u(x)$ denotes the eigenfunction, and $\lambda$ represents the eigenvalue.

1. Mesh Generation: Using the central difference quotient, we divide the interval $[0, L]$ into $N + 1$ evenly spaced points, including the endpoints. The distance between adjacent points is denoted as $\Delta x = \frac{L}{N}$.

2. Operator Discretization: This step involves formulating the difference equation. At each interior node, which excludes the endpoints and totals $N - 1$ points, we apply a central difference

approximation for the second derivative, represented as

$$\frac{d^2u}{dx^2} \approx \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{(\Delta x)^2}$$

3. Matrix Assembly: In this phase, the discretized operator is represented in a matrix form. Following the approximation, we construct the matrix $A$, an $N-1 \times N-1$ tridiagonal matrix, crucial for the computations. The matrix $\boldsymbol{A}$ is constructed as:

$$\boldsymbol{A} = \frac{1}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix}$$

4. Applying Boundary Conditions: For the wave equation with boundary conditions $u(0) = u(L) = 0$, these fixed-end conditions are integrated into the matrix equation. In the FDM framework, the values at the endpoints ($u_0$ and $u_N$) are zero, directly reflecting the boundary conditions. The impact of these conditions is encapsulated in the matrix $\boldsymbol{A}$, affecting the entries related to $u_1$ and $u_{N-1}$ (the grid points adjacent to the boundaries). The tridiagonal matrix $\boldsymbol{A}$ incorporates these boundary conditions, ensuring that the computed eigenfunctions satisfy $u(0) = u(L) = 0$.

5. Solving the Matrix Eigenvalue Problem: The final computational step involves solving the matrix eigenvalue problem, expressed as $\boldsymbol{Au} = \lambda \boldsymbol{u}$. This includes determining the eigenvalues $\lambda$ and corresponding eigenvectors $\boldsymbol{u}$, which are discrete approximations of the eigenfunctions of the original differential equation.

6. Obtaining the Numerical Solution: By solving the eigenvalue problem, we obtain numerical solutions that approximate the behavior of the original differential equation. These solutions reveal the eigenvalues and eigenvectors and provide insights into the physical phenomena modeled by the equation.

# D Details of Experimental Setup

## D.1 Baseline

The baseline algorithms were implemented using the following numerical computing libraries:

- Eigsh: A SciPy (v1.14.1) implementation wrapping ARPACK's SSEUPD and DSEUPD functions, which compute eigenvalues and eigenvectors using the Implicitly Restarted Lanczos Method. The default parameters were used.
- Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG): Implemented in SLEPc (v3.21.1) with default parameters.
- Krylov-Schur (KS): Implemented in SLEPc (v3.21.1) with default parameters.
- Jacobi-Davidson (JD): Implemented in SLEPc (v3.21.1). The implementation uses 'bcgsl' as the linear equation solver, 'bjacobi' as the preconditioner, and sets the linear equation solving precision to 1e-5.
- ChFSI: Implemented in ChASE (v1.6) with default parameters.

## D.2 Dataset

All operators in this paper use Dirichlet boundary conditions.

1. Generalized Poisson Operator

We consider two-dimensional generalized Poisson operators, which can be described by the following equation (Li et al., 2020; Rahman et al., 2022; Kovachki et al., 2021; Lu et al., 2022):

$$-\nabla \cdot (K(x,y)\nabla h(x,y)) = \lambda h(x,y),$$

In our experiment, $K(x, y)$ is derived using the Gaussian Random Field (GRF) method. We convert these operators into matrices using the central difference scheme of FDM. The parameters inherent to the GRF serve as the foundation for our sort scheme.

2. Second-Order Elliptic Partial Differential Operator

We consider general two-dimensional second-order elliptic partial differential operators, which are frequently described by the following generic form (Evans, 2022; Bers et al., 1964):

$$\mathcal{L}u \equiv a_{11}u_{xx} + a_{12}u_{xy} + a_{22}u_{yy} + a_1u_x + a_2u_y + a_0u = \lambda u,$$

where $a_0, a_1, a_2, a_{11}, a_{12}, a_{22}$ are constants, and $f$ represents the source term, depending on $x, y$. The variables $u, u_x, u_y$ are the dependent variables and their partial derivatives. The equation is classified as elliptic if $4a_{11}a_{22} > a_{12}^2$.

In our experiments, $a_{11}, a_{22}, a_1, a_2, a_0$ are uniformly sampled within the range $(-1, 1)$, while the coupling term $a_{12}$ is sampled within $(-0.01, 0.01)$. We then select equations that satisfy the elliptic condition to form our dataset. We convert these operators into matrices using the central difference scheme of FDM. The coefficients $a_0, a_1, a_2, a_{11}, a_{12}, a_{22}$ serve as the foundation for our sort scheme.

3. Helmholtz Operator

We consider two-dimensional Helmholtz operators, which can be described by the following equation (Zhang et al., 2022):

$$\nabla \cdot (p(x, y)\nabla u(x, y)) + k^2(x, y) = \lambda u(x, y),$$

Physical Contexts in which the Helmholtz operator appears: 1. Acoustics; 2. Electromagnetism; 3. Quantum Mechanics.

In Helmholtz operators, $k$ is the wavenumber, related to the frequency of the wave and the properties of the medium in which the wave is propagating. In our experiment, $p(x, y)$ and $k(x, y)$ are derived using the GRF method. The parameters inherent to the GRF serve as the foundation for our sort scheme.

4. Vibration Equation

We consider the vibration equation for thin plates, which can be described by the following eigenvalue problem (Xue et al., 2018):

$$\nabla^2\big(D(x, y)\nabla^2 u(x, y)\big) = \lambda\rho(x, y)u(x, y),$$

Physical contexts in which the vibration equation appears: 1. Structural dynamics of thin plates; 2. Modal analysis in mechanical engineering; 3. Vibrational behavior of elastic materials.

In this equation, $D(x, y)$ represents the flexural rigidity of the plate, $\rho(x, y)$ is the density distribution, and $\lambda$ corresponds to the eigenvalue, which is related to the natural frequencies of the system. The eigenfunction $u(x, y)$ describes the mode shapes of vibration.

In our experiment, $D(x, y)$ and $\rho(x, y)$ are derived using the GRF method. The parameters inherent to the GRF serve as the foundation for our sorting scheme.

### D.3 Environment

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

- Platform: Docker version 4.33.1 (windows 11)
- Operating System: Ubuntu 22.04.3 LTS
- Processor: CPU AMD Ryzen 9 8945HS w, clocked at 4.00 GHz

### D.4 Experimental Parameter Configuration

All baseline methods were implemented using their default parameters from respective libraries.

For SCSF, the following configurations were adopted:

- The size of the inherited subspace varies according to the number of eigenvalues to be computed. Specifically, when calculating 20, 100, 200, 300, and 400 eigenvalues, the corresponding subspace sizes are set to 4, 20, 40, 60, and 80, respectively.
- The filter degree parameter $m$ is consistently set to 20 across all experiments.
- Truncation threshold for low frequencies $p_0$ is consistently set to 20 across all experiments.
- Each experiment generates a dataset consisting of 1,000 samples. In this paper, the Experimental tables report the average solving time for each eigenvalue problem.

### D.5 Error Metrics

- Relative Residual Error:
  To estimate the bias of the eigenpair $(\tilde{v}, \tilde{\lambda})$ predictions, we employ relative residual error as follows:
  $$\text{Relative Residual Error} = \frac{||\mathcal{L}\tilde{v} - \tilde{\lambda}\tilde{v}||_2}{||\mathcal{L}\tilde{v}||_2}.$$
  Here, $\tilde{v}$ represents the eigenfunction predicted by the model, and $\tilde{\lambda}$ denotes the eigenvalue predicted by the model. When $\tilde{\lambda}$ is the true eigenvalue and $\tilde{v}$ is the true eigenfunction, the Relative Residual Error equals $0$.

### D.6 Relationship with Parallelization and Experimental Setting

The SCSF framework is designed to be complementary to parallel computing architectures; the relationship is both orthogonal and synergistic. Fundamentally, SCSF accelerates the serial processing of a sequence of related eigenvalue problems. In a practical, large-scale setting, a total dataset of $N$ problems can be partitioned into $M$ independent chunks. Subsequently, $M$ instances of the SCSF algorithm can be executed in parallel across $M$ computing units, with each computing unit responsible for solving one chunk.

To ensure a fair and direct comparison of algorithmic efficiency under practical, parallelized conditions, all experiments reported in this paper were conducted using the Message Passing Interface (MPI) with 8 cores in parallel.

# E Experimental Data and Supplementary Experiments

## E.1 Main Experimental Data

As shown in Tables 7, 6, 9, SCSF showed the best performance among all tested configurations

Table 6: Comparison of average computation times (in seconds) for eigenvalue problems using various algorithms on the generalized Poisson operator dataset. The first row lists different algorithms, and the first column shows the number of eigenvalues $L$ computed for each matrix. Matrix dimension = 2500, precision = 1e-12.

| $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|
| 150 | 9.15 | 46.8 | 14.9 | 138 | 17.3 | 7.95 |
| 200 | 14.2 | 73.0 | 23.8 | 270 | 24.0 | 12.9 |
| 250 | 19.8 | 109 | 34.3 | 553 | 30.2 | 19.0 |
| 300 | 26.3 | 152 | 45.6 | 921 | 38.0 | 25.7 |
| 350 | 31.5 | 203 | 58.4 | 1732 | 45.8 | 29.8 |
| 400 | 36.9 | 265 | 72.3 | 2691 | 57.4 | 33.9 |
| 450 | 42.8 | 342 | 87.3 | 3708 | 74.2 | 38.3 |

Table 7: Comparison of average computation times (in seconds) for eigenvalue problems using various algorithms on the second-order elliptic operator dataset. The first row lists different algorithms, and the first column shows the number of eigenvalues $L$ computed for each matrix. Matrix dimension = 4900, precision = 1e-10.

| $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|
| 150 | 31.35 | 91.80 | 40.65 | 214.80 | 38.37 | 19.62 |
| 200 | 41.82 | 139.20 | 61.77 | 414.30 | 43.90 | 24.08 |
| 250 | 52.17 | 197.04 | 84.65 | 861.44 | 53.42 | 28.00 |
| 300 | 62.47 | 264.10 | 110.50 | 1446.00 | 60.69 | 29.88 |
| 350 | 74.59 | 355.18 | 147.01 | 2324.88 | 64.94 | 31.52 |
| 400 | 87.19 | 459.70 | 188.70 | 3386.00 | 67.13 | 34.60 |
| 450 | 100.28 | 577.67 | 235.56 | 4629.38 | 76.32 | 40.05 |

Table 8: Comparison of average computation times (in seconds) for eigenvalue problems using various algorithms on the Helmholtz operator dataset. The first row lists different algorithms, and the first column shows the number of eigenvalues $L$ computed for each matrix. Matrix dimension = 6400, precision = 1e-8. The symbol '-' denotes data not recorded due to excessive computation times.

| $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|
| 200 | 151.70 | 129.90 | 98.34 | 489.60 | 107.12 | 31.31 |
| 300 | 190.84 | 273.08 | 192.88 | 1601.08 | 113.73 | 37.78 |
| 400 | 253.50 | 460.40 | 283.00 | 3829.00 | 121.53 | 40.52 |
| 500 | 344.60 | 720.33 | 310.21 | - | 135.73 | 47.41 |
| 600 | 398.80 | 1031.00 | 329.60 | - | 146.24 | 51.32 |

## E.2 Analysis of the Influence of Matrix Dimension

As demonstrated in Table 10, the impact of matrix dimension on algorithm performance reveals several key insights. For matrices below dimension 3600, SCSF and Eigsh show comparable efficiency. However, SCSF's advantages become increasingly pronounced as matrix dimensions grow larger.

Table 9: Comparison of average computation times (in seconds) for eigenvalue problems using various algorithms on the Vibration operator dataset. The first row lists different algorithms, and the first column shows the number of eigenvalues $L$ computed for each matrix. Matrix dimension = 10000, precision = 1e-8. The symbol '-' denotes data not recorded due to excessive computation times.

| $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|
| 200 | 397.9 | 333.7 | 272.0 | 1230 | 300.8 | 85.70 |
| 300 | 516.8 | 750.0 | 520.0 | 3600 | 305.0 | 96.50 |
| 400 | 635.6 | 1170 | 768.8 | - | 310.5 | 107.2 |
| 500 | 820.0 | 1950 | 810.0 | - | 350.0 | 120.0 |
| 600 | 1037 | 2716 | 857.8 | - | 382.3 | 131.4 |

Table 10: Comparison of different algorithm computation time (in seconds) for varying matrix dimensions using the generalized Poisson operator dataset. Results show average computation times for solving 400 eigenvalues with a precision of 1e-12.

| Matrix Dimension | Eigsh | LOBPCG | KS | JD | ChFSI | SCSF (ours) |
|---|---|---|---|---|---|---|
| 2500 | 36.86 | 265.30 | 72.32 | 2691.00 | 57.41 | 33.91 |
| 3600 | 66.41 | 387.20 | 116.50 | 2990.00 | 102.4 | 65.41 |
| 4225 | 89.13 | 467.74 | 151.36 | 3548.13 | 126.2 | 70.79 |
| 4900 | 121.90 | 546.20 | 187.80 | 3886.00 | 153.5 | 74.23 |
| 5625 | 186.21 | 691.83 | 251.19 | - | 216.8 | 85.11 |
| 6400 | 282.80 | 860.00 | 337.70 | - | 282.2 | 93.86 |
| 8100 | 707.95 | 1412.54 | 707.95 | - | 435.1 | 114.82 |
| 10000 | 3162.28 | 2511.89 | 1995.26 | - | 590.3 | 158.49 |

At dimension 10000, SCSF achieves remarkable speedups: 20× faster than Eigsh, 16× faster than LOBPCG, 13× faster than KS, and 3.7× faster than ChFSI. This phenomenon can be attributed to how larger matrix dimensions result in fewer errors and noise in the computed eigenvalues, allowing SCSF to better exploit similarities between problems. Additionally, the JD algorithm becomes computationally intractable at and above dimension 5625, while SCSF maintains stable performance even at high dimensions.

### E.3 Analysis of Computational Times for SCSF Components

Table 11: Analysis of Computational Times (in seconds) for SCSF Components.

| All | Filter (line 3) | QR (line 4) | RR (line 5) | Resid (line 6) | Sort |
|---|---|---|---|---|---|
| 9.89e+0 | 7.41e+0 | 3.12e-1 | 9.76e-1 | 7.95e-1 | 1.51e-2 |

We conducted a statistical analysis of the average time consumption for each component of the SCSF algorithm on the generalized Poisson operator dataset, with a matrix dimension of 2500 and the number of eigenvalues to be solved set to 100. The results are presented in Table 11. The notation 'line x' within parentheses corresponds to line x in Algorithm 3, 'ALL' denotes the total time consumption, and 'sort' represents the average time required by the sorting algorithm. It is evident that the filtering process accounts for over 70% of the total time consumption, which aligns with our theoretical analysis in Section 3.2.

### E.4 Analysis of Hyperparameters

#### E.4.1 Degree Parameter

Table 12: Average Computational Times (in seconds) of SCSF under Different Degree Parameters $m$.

| Deg | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
|---|---|---|---|---|---|---|---|---|
| Time (s) | 43.92 | 39.79 | 40.52 | 40.64 | 40.85 | 41.13 | 41.19 | 43.50 |

We investigated the impact of different degree parameters $m$ on the performance of SCSF. As shown in Table 12, the experiments were conducted on the Helmholtz operator dataset with a matrix dimension of 6400, a solution accuracy of 1e-8, 400 eigenvalues to be solved, and an inherited subspace size of 80. The degree parameter $m$, as described in Algorithm 3, primarily controls the order of the Chebyshev polynomial. The results indicate that varying $m$ within the range of 12 to 40 has a minimal effect on the computation time of SCSF. Therefore, as long as $m$ is chosen within a reasonable range, its specific value does not significantly influence the performance. In the main experiments of this paper, $m$ is fixed at 20.

#### E.4.2 Subspace Dimension

Table 13: Average Computational Times (in seconds) of SCSF under Different Subspace Dimension.

| Dim | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
|---|---|---|---|---|---|---|---|---|
| Time (s) | 43.28 | 44.35 | 42.43 | 40.52 | 39.65 | 37.43 | 38.28 | 38.58 |

We examine the influence of different inherited subspace sizes on the performance of SCSF. As presented in Table 13, the experiments are conducted on the Helmholtz operator dataset with a matrix dimension of 6400, a solution accuracy of 1e-8, 400 eigenvalues to be computed, and a degree parameter $m$ set to 20.

The results demonstrate that as the inherited subspace size increases, the computation time of SCSF initially decreases and then rises, reaching its minimum around a size of 100. The reduction in computation time at the front end is attributed to the enriched initial subspace with more available information as the inherited subspace grows. Conversely, the increase in computation time at the back end is due to the significantly higher overhead of performing Chebyshev filtering with a larger inherited subspace.

Overall, as long as the inherited subspace size is set within a reasonable range, its impact on SCSF remains minimal. In our experiments, we consistently set the inherited subspace size to 20% of the number of eigenvalues to be computed.

### E.4.3 Truncation Threshold for Low Frequencies

Table 14: Average computational times (in seconds) of SCSF under different truncation thresholds.

| | No sort | $p_0 = 10$ | $p_0 = 20$ | $p_0 = 30$ | $p_0 = 40$ | Greedy |
|---|---|---|---|---|---|---|
| One-sided distance | 0.95 | 0.89 | 0.85 | 0.85 | 0.85 | 0.85 |
| Sort time (s) | 0 | 110 | 151 | 193 | 246 | 593 |
| Average solve time (s) | 66.7 | 52.2 | 40.5 | 40.5 | 40.5 | 40.5 |

We measure the similarity between matrices by computing the cosine of the principal angles between their 10-dimensional invariant subspaces (spanned by the smallest 10 eigenvectors in modulus) (one-sided distance). Smaller values indicate higher similarity. As presented in Table 14, the experiments are conducted on the Helmholtz operator dataset with a matrix, a solution accuracy of 1e-8, 400 eigenvalues to be computed, and a degree parameter $m$ set to 20, 10k data problems, parameter matrix $P$ with dimension $p = 80$, and varying truncation frequencies $p_0$

The results demonstrate that sorting significantly increases inter-problem correlation in the dataset (explaining the performance gain). The truncation parameter $p_0$ affects sorting time, sorting quality, and solver time. For $p_0 \geq 20$, solver time becomes stable, showing diminishing returns. This reflects the interplay between sorting and Chebyshev iteration. In the main experiments of this paper, $p_0$ is fixed at 20.

### E.5 Reliability of Generated Data as Ground Truth

A key concern was whether the data generated by our method, which relies on numerical solvers, is a reliable 'ground truth' for training neural networks. To address this, we trained a NeurKItt (Luo et al., 2024) model on generalized Poisson datasets generated by various solvers (including our SCSF) at different matrix dimensions. The precision for all solvers was set to a high tolerance of $10^{-12}$.

Table 15: Impact of data generation method on the performance of a trained NeurKItt model. The consistent final loss indicates that data from all tested solvers serves as a reliable ground truth.

| Generation Method | Matrix Dimension | Generation Time | NeurKItt Principal Angle Loss |
|---|---|---|---|
| Eigsh | 2500 / 6400 / 10000 | 10h / 80h / 800h | 0.06 / 0.06 / 0.06 |
| LOBPCG | 2500 / 6400 / 10000 | 70h / 240h / 700h | 0.06 / 0.06 / 0.06 |
| ChFSI | 2500 / 6400 / 10000 | 16h / 44h / 160h | 0.06 / 0.06 / 0.06 |
| SCSF (ours) | 2500 / 6400 / 10000 | 9h / 26h / 45h | 0.06 / 0.06 / 0.06 |

### E.6 Comparison with Supervised and Unsupervised Neural Network Methods

To clarify the significance of accelerating dataset generation for the dominant supervised learning paradigm, we conducted an experiment comparing the performance and resource trade-offs of different categories of eigensolvers. We evaluated our method (SCSF), a traditional solver (Eigsh), a supervised neural network (NeurKItt Luo et al. (2024)), and two state-of-the-art unsupervised neural networks (NeuralEF Deng et al. (2022) and NeuralSVD Ryu et al. (2024)) on a 2D Helmholtz problem (solving for the smallest 100 eigenvalues, matrix dimension $6400$).

The results, presented in Table 16, highlight the distinct characteristics of each approach. Unsupervised methods obviate the need for pre-generated datasets but require substantial 'solving time' for each new operator, as they essentially perform an optimization from scratch. In contrast, supervised methods offer near-instantaneous inference but demand significant upfront investment in both data generation and model training. Our method, SCSF, dramatically reduces the data generation bottleneck for these powerful supervised models.

As shown in Table 15, the final performance of the trained NeurKItt model (measured by Principal Angle Loss) was identical regardless of which high-precision solver was used for data generation or

Table 16: Comparison of different eigensolver paradigms on a 2D Helmholtz problem. 'Solving Time' for unsupervised methods refers to the entire optimization process required to find the solution for a single operator instance.

| Category | Algorithm | Solving Time | Training Time | Dataset Gen. Time | Relative Residual |
|---|---|---|---|---|---|
| Our Method | SCSF (random init) | 1 min | - | - | $10^{-8}$ |
| Traditional | Eigsh | 1 min | - | - | $10^{-8}$ |
| Supervised NN | NeurKItt | 0.1s | 3h | 20h | $10^{-2}$ |
| Unsupervised NN | NeuralEF | 2h | - | - | $10^{-2}$ |
| Unsupervised NN | NeuralSVD | 3h | - | - | $10^{-2}$ |

the specific matrix dimension (for dimensions $\geq 2500$). This demonstrates that the discretization and solver errors are orders of magnitude smaller than the neural network's approximation error, confirming that the generated data serves as a highly reliable ground truth for training purposes.

### E.7 Performance Bounds and the Impact of Dataset Similarity

To provide theoretical insight into SCSF's performance bounds, we conducted an experiment to quantify the relationship between dataset similarity and acceleration. We generated a sequence of Helmholtz operator problems where each subsequent problem is a slight perturbation of the previous one. The magnitude of this perturbation reflects the dataset's internal similarity. A smaller perturbation size indicates higher similarity. The experiment was run on the Helmholtz dataset (dimension $6400$, $L = 200$ eigenvalues).

Table 17: Average solution time (seconds) as a function of dataset similarity (perturbation size). Lower perturbation implies higher similarity. The experiment was run on the Helmholtz dataset (dim $= 6400, L = 200$).

| Perturbation Size | Eigsh | LOBPCG | ChFSI | SCSF (w/o sort) | SCSF |
|---|---|---|---|---|---|
| 50% | 151 | 130 | 107 | 76 | 27 |
| 10% | 150 | 129 | 107 | 48 | 23 |
| 1% | 152 | 130 | 107 | 14 | 6 |
| 0% (Identical) | 151 | 130 | 107 | 2 | 2 |
| Standard Generation | 152 | 130 | 107 | 82 | 31 |
| Independent Problems | 152 | 130 | 107 | 107 | 107 |

The results in Table 17 show that SCSF's performance is strongly correlated with dataset similarity. As problems become more similar (perturbation size decreases), the speedup increases dramatically. The experiment also highlights the effectiveness of our sorting algorithm; SCSF consistently outperforms SCSF without sorting ('SCSF w/o sort') across various similarity levels. In the theoretical limit of identical problems (0% perturbation), the solution is found in just a few iterations. Conversely, for completely independent problems, SCSF's performance gracefully degrades to that of ChFSI, as expected.

### E.8 Analysis of Failure Cases: Discontinuous Datasets

The core assumption of SCSF is that the dataset is generated from a process with underlying continuity, allowing our sorting algorithm to group similar problems effectively. To investigate the behavior of SCSF when this assumption is violated, We simulated a gradual mixing of the two datasets (Helmholtz and Poisson). We varied the proportion of Helmholtz operators from 100% (a fully continuous dataset) down to 0% (a different fully continuous dataset), with mixed ratios in between representing varying degrees of discontinuity. (dimension $6400$, $L = 200$ eigenvalues) and solved them sequentially.

Table 18 presents the results. As expected, the performance gain of SCSF is reduced in this discontinuous scenario because the inter-problem correlation is disrupted, diminishing the effectiveness of the sorting module. However, even in this challenging case, SCSF still provides a notable speedup over

Table 18: Performance on a discontinuous dataset created by mixing Helmholtz and Poisson problems. All times are in seconds.

| Helmholtz % | Eigsh | LOBPCG | ChFSI | SCSF (w/o sort) | SCSF |
|---|---|---|---|---|---|
| 100% (Helmholtz) | 152 | 130 | 107 | 46 | 31 |
| 75% | 154 | 203 | 116 | 94 | 78 |
| 50% (1:1 mix) | 154 | 280 | 132 | 118 | 98 |
| 25% | 152 | 359 | 141 | 108 | 80 |
| 0% (Poisson) | 149 | 454 | 153 | 81 | 52 |

baseline solvers, demonstrating a degree of robustness. The performance of 'SCSF' approaches that of 'SCSF (w/o sort)', confirming that the sorting component's benefit is tied to dataset continuity.

### E.9 Cost-Benefit Analysis of the Sorting Algorithm

To address the trade-off between the cost of sorting and its benefits, we analyzed its computational overhead. Our analysis shows that the cost of the Truncated FFT Sort is negligible in the context of large-scale dataset generation. For example, as shown in Table 4 of the main paper, sorting a dataset of $10^4$ samples takes approximately 151 seconds. In contrast, solving a single eigenvalue problem from the Helmholtz dataset can take over 250 seconds with a standard solver like Eigsh. For a full dataset of this size, the sorting overhead constitutes less than 0.1% of the total generation time.

The benefit, however, is substantial. As shown in Table 3, sorting reduces the number of solver iterations by 5-50% and total floating-point operations (Flops) by 7-43%. Furthermore, our Truncated FFT Sort is significantly more cost-effective than a naive greedy sort, achieving nearly identical final solver performance at a fraction of the computational cost (see Tables 4 and 5). Given this highly favorable cost-benefit ratio, the sorting step is a crucial and efficient component of the SCSF framework.

### E.10 Sensitivity to Parameterization

To assess the sensitivity of the proposed parameter sorting to different PDE parameterizations, we conducted an additional experiment comparing two distinct discretization methods for the Helmholtz problem: the Finite Difference Method (FDM) (utilized in the main text) and the Finite Element Method (FEM/Galerkin).

As shown in Table 19, SCSF demonstrates consistent speedups across both discretization schemes. For the FEM dataset, we utilized a dimension of 10,000 with a tolerance of 1e-8.

Table 19: Comparison of SCSF performance under different parameterizations (Helmholtz). Times are in seconds. '-' indicates non-convergence or excessive time cost.

| Dataset | $L$ | Eigsh | LOBPCG | KS | JD | ChFSI | **SCSF (ours)** |
|---|---|---|---|---|---|---|---|
| FDM (Central Diff) | 200 | 151.7 | 129.9 | 98.34 | 489.6 | 107.1 | 31.31 |
| | 400 | 253.5 | 460.4 | 283.0 | 3829 | 121.5 | 40.52 |
| Dim: 6400, Tol: 1e-8 | 600 | 398.8 | 1031 | 329.6 | - | 146.2 | 51.32 |
| FEM (Galerkin) | 200 | 623.1 | 515.7 | 365.6 | 1885 | 450.9 | 116.7 |
| | 400 | 938.5 | 1746 | 1166 | - | 481.3 | 142.9 |
| Dim: 10000, Tol: 1e-8 | 600 | 1499 | - | 1299 | - | 599.9 | 196.3 |

These results confirm that SCSF remains highly effective regardless of the discretization scheme, provided the relationship between parameters and operators remains continuous.

### E.11 Generalization of Truncation Level ($p_0$)

To justify this universal constant, we calculated the ratio of the Frobenius norm of high-frequency components (frequency $> 20$) to the total norm for each dataset. The results are presented in Table 20.

Table 20: Ratio of high-frequency component energy (frequency $> p_0 = 20$) to total energy across different PDE datasets.

| Dataset | Poisson | Ellipse | Helmholtz | Vibration |
|---|---|---|---|---|
| High-frequency Ratio ($> p_0$) | 3.4% | 3.7% | 4.4% | 4.8% |

As shown, high-frequency components account for less than 5% of the total energy across all diverse PDE families. This confirms that $p_0 = 20$ consistently captures over 95% of the structural information inherent in the parameter matrices.

# F THEORETICAL ANALYSIS OF TRUNCATED FFT SORTING

Our sorting algorithm approximates the distance between operators to optimize the solving sequence. To analyze the theoretical validity of this approximation, we consider the isometric property of the Discrete Fourier Transform (DFT). Let $P^{(i)}$ and $P^{(j)}$ denote the discretized parameter matrices (e.g., diffusion coefficients) for two distinct problems. By Parseval's identity, the DFT is a unitary transformation (up to a scaling factor), implying that the Frobenius distance in the spatial domain is strictly equivalent to that in the frequency domain:

$$\|P^{(i)} - P^{(j)}\|_F^2 = \|\text{FFT}(P^{(i)}) - \text{FFT}(P^{(j)})\|_F^2,$$

where $\text{FFT}(\cdot)$ denotes the normalized DFT. The SCSF algorithm computes a proxy distance using a truncated spectrum, retaining only frequencies within a low-frequency bandwidth $p_0$. Consequently, the exact distance can be decomposed into the sorting metric and a residual truncation error:

$$\|P^{(i)} - P^{(j)}\|_F^2 = \underbrace{\|\text{Trunc}_{p_0}(\Delta\hat{P})\|_F^2}_{\text{SCSF Metric}} + \underbrace{\|(I - \text{Trunc}_{p_0})(\Delta\hat{P})\|_F^2}_{\text{Truncation Error } \epsilon_{ij}},$$

where $\Delta\hat{P} = \text{FFT}(P^{(i)} - P^{(j)})$ and $\text{Trunc}_{p_0}$ is the truncation operator.

The magnitude of the error term $\epsilon_{ij}$ is governed by the spectral decay properties of the underlying physical operators. The parameter matrices in our dataset, generated via Gaussian Random Fields (GRF) or representing physical media, typically exhibit smoothness or piecewise smoothness corresponding to Sobolev regularity $H^s$ with $s > 0$. Harmonic analysis dictates that the spectral energy of such functions decays polynomially or exponentially with frequency magnitude $|\mathbf{k}|$. Specifically, the Fourier coefficients satisfy $|\hat{P}_\mathbf{k}| \lesssim |\mathbf{k}|^{-s}$. Therefore, the truncation error is bounded by the tail energy of the spectrum:

$$\epsilon_{ij} = \sum_{|\mathbf{k}|>p_0} |\Delta\hat{P}_\mathbf{k}|^2 \approx \mathcal{O}(p_0^{-2s+d}),$$

where $d$ is the spatial dimension. This decay rate implies that the high-frequency components contribute negligibly to the global topological distance between operators. This is also the fundamental reason why truncated FFT sorting is effective.