Journal of Circuits, Systems, and Computers
Vol. 35, No. 9 (2026) 2650020 (26 pages)

DOI: 10.1142/S0218126626500209

World Scientific
www.worldscientific.com

SolverSet: A Large-scale Benchmark Dataset for the Auto-selection of the Optimal Combination of Iterative Solvers and Preconditioners

Hantao Xiong [*,‡] Wangdong Yang [*,§] Shengle Lin [*,¶]
Weiqing He [*,‖] Keqin Li [†,**] and Kenli Li [*,††]

[*]*College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410082, China*

[†]*Department of Computer Science, State University of New York, New Paltz, New York 12561, USA*

[‡]*xionghantao512@hnu.edu.cn*
[§]*yangwangdong@hnu.edu.cn*
[¶]*lsl036@hnu.edu.cn*
[‖]*heweiqing@hnu.edu.cn*
[**]*lkl@hnu.edu.cn*
[††]*lik@newpaltz.edu*

Received 15 June 2025
Accepted 26 September 2025
Published 19 November 2025

The combination of iterative solvers and preconditioners is the mainstream approach for solving sparse linear systems of the form $Ax = b$, which are fundamental to many scientific and engineering applications. However, the automatic selection (auto-selection) of the optimal solver–preconditioner combination remains a challenging task. Although machine learning or deep learning methods have been explored for this purpose, their performance has been limited due to the lack of standardized, large-scale datasets. To address this issue, we introduce a large-scale benchmark dataset called SolverSet, designed for the auto-selection of the optimal combination of iterative solvers and preconditioners. We develop a matrix generation tool to produce a wide variety of large-scale sparse matrices, based on which we construct the SolverSet dataset. This dataset now comprises 12,651 large-scale matrices and their corresponding sparse linear systems, effectively overcoming the limitations of prior work — namely, limited size and a small number of systems — making it highly suitable for automatic selection modeling. We evaluate several baseline methods on SolverSet to validate its effectiveness and usability. Furthermore, we analyze key features of sparse linear systems and propose a deep learning based model named DL-Solver to predict the optimal solver–preconditioner combination for a given system. Experimental results demonstrate that SolverSet serves as a valuable benchmark for auto-selection research, and DL-Solver outperforms state-of-the-art method in predictive performance on the test set, achieving improvements of 2.14%, 3.47%, 2.71% and 3.16% in accuracy, macro-precision, macro-recall and macro-F1 score, respectively.

[§]Corresponding author.

Keywords: Auto-selection; benchmark dataset; deep learning; iterative solver; preconditioner; sparse linear system.

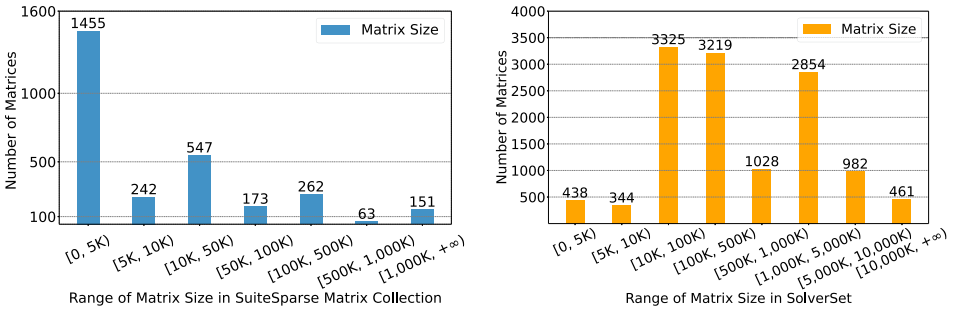
1. Introduction

Iterative solvers are the preferred methods for addressing large-scale sparse linear systems of the form $Ax = b$, which is the vital step in numerically solving partial differential equations (PDEs) from various scientific and engineering problems.[1] Preconditioners are primarily used to optimize the eigenvalue distribution of the coefficient matrix in sparse linear systems, thereby accelerating the convergence rate of iterative solvers by solving the corresponding preconditioned system.[2] Therefore, combinations of iterative solvers and preconditioners have become the most widely used strategy for solving large-scale sparse linear systems.

The development of numerical linear algebra has yielded numerous iterative solvers and preconditioners for solving sparse linear systems.[3,4] Moreover, the selection of iterative solver and preconditioner combinations is a crucial factor determining the convergence properties and speed. For a given sparse linear system, different solver–preconditioner combinations can lead to significant variations in convergence speed. Given the large number of available combinations (over 100 in PETSc[5]), selecting the optimal one can be a significant challenge for ordinary users without much expertise in numerical computation. Hence, the automatic selection (auto-selection) of the optimal combination of iterative solvers and preconditioners has emerged as a critical research problem for efficiently solving large-scale sparse linear systems.

Some work has endeavored to view the task of the automatic selection of iterative solvers and preconditioners as a classification problem in machine learning.[6–15] These methods typically employ traditional machine learning models such as support vector machine (SVM)[16] and AdaBoost[17] to predict the optimal iterative solver and preconditioner for a given sparse linear system. In recent years, with the advancement of deep learning, some studies have begun to explore the use of deep learning methods for predicting the optimal iterative solver and preconditioner.[18–23] However, the aforementioned work merely represents an initial exploration of the task of automatic selection and has not achieved satisfactory prediction performance. One of the most critical reasons for this problem is **the lack of a standardized large-scale benchmark dataset for modeling the auto-selection of the optimal combination of iterative solvers and preconditioners.**

Recently, the critical role of datasets in the automatic selection of iterative solvers and preconditioners has been highlighted by researchers. Zou *et al.*[24] conduct a comprehensive survey on the datasets used in existing studies, which shows that most of them utilize matrices from the SuiteSparse Matrix Collection[25] to construct spare linear systems, with some using linear systems obtained by numerically solving PDEs in several applications. One study uses the Matrix Market[26] to build sparse linear systems. These studies fail to establish a standardized benchmark dataset for



(a) (b)

Fig. 1. (a) The range of matrix size in the most widely used SuiteSparse Matrix Collection. The x-axis represents the matrix size range (in thousands, where $K = 1{,}000$), with matrix size defined as the number of rows. The y-axis represents the number of matrices within each size range. (b) The range of matrix size in our proposed SolverSet. The x-axis represents the matrix size range (in thousands, where $K = 1,000$), with matrix size defined as the number of rows. The y-axis represents the number of matrices within each size range. It is evident that SolverSet has a larger number of matrices and larger matrix sizes compared to SuiteSparse Matrix Collection.

automatic selection modeling, making it difficult to evaluate the results consistently. Moreover, the most popular SuiteSparse Matrix Collection, currently contains only 2,893 matrices, with a portion of them (e.g., matrices of pattern type) not suitable for constructing sparse linear systems. The Matrix Market contains only about 500 matrices. Additionally, due to the increasing scale of applications,[27–30] public matrix datasets are relatively small in scale and do not adequately represent the size of sparse linear systems in real applications. For example, we conduct a statistical analysis of the matrix size in the SuiteSparse Matrix Collection, and the results are shown in Fig. 1(a), from which we discover that 50.29% $(1,455/2,893)$ and 58.66% $(1,697/2,893)$ of the matrices have a size smaller than 5,000 and 10,000, respectively. Only 5.21% $(151/2,893)$ of the matrices have a size larger than 1 million. Apparently, the SuiteSparse Matrix Collection contains a limited number of matrices, and most of them are also relatively small in size. It is not suitable as a standard large-scale benchmark dataset for automatic selection modeling, in which "large-scale" should carry two distinct meanings: the large number of sparse linear systems and the large size of the linear systems themselves.

To overcome the aforementioned issues, we construct **a large-scale benchmark dataset named SolverSet**. Specifically, we build a matrix generation tool based on many curated real-world sample programs involving the solution of PDEs. Using this tool, we produce a sparse matrix set consisting of 12,651 large-scale sparse matrices. Based on these matrices, we build sparse linear systems, and employ candidate combinations in solver package PETSc[5] to solve them, establishing the optimal combinations as labels, which leads to the proposed SolverSet, containing 12,651 samples. The matrix size distribution in SolverSet is shown in Fig. 1(b), from which we observe that 93.82% $(11,869/12,651)$ of the matrices in SolverSet have size larger

than 10,000, and 33.97% $(4,297/12,651)$ of matrices have size larger than 1 million. Compared to currently available public datasets, SolverSet contains the largest number of matrices, and these matrices are also larger in scale.

To verify the validity and usability of SolverSet, we evaluate several baseline models on it. Additionally, by analyzing the features of sparse linear systems, we propose a deep learning model called DL-Solver. Experimental results demonstrate that the SolverSet has excellent properties and can enhance the performance of state-of-the-art methods. Moreover, our proposed DL-Solver model outperforms baseline models in terms of prediction performance.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries on iterative solvers and precoditioners for solving sparse linear systems. Section 3 presents the construction of SolverSet. Section 4 demonstrates the automatic selection model DL-Solver. Section 5 presents the experimental evaluation results of this work. Section 6 describes related work. Finally, Sec. 7 concludes this work.

2. Preliminaries

This section provides the basic knowledge of iterative solvers and preconditioners utilized for solving sparse linear systems represented as

$$Ax = b, \tag{1}$$

where A is usually an $n \times n$ nonsingular sparse coefficient matrix, and b and x represent the given right-hand side vector and the vector to be solved, respectively.

2.1. *Iterative solvers*

As the problem scale increases and parallel computing advances, iterative solvers have emerged as the preferred methods for solving sparse linear systems. These solvers can be roughly categorized into two main groups: stationary iterative solvers and nonstationary iterative solvers. Stationary iterative solvers (e.g., Jacobi and Gauss–Seidel methods) follow a simple iterative formula:

$$x^{k+1} = Tx^k + c, \quad k = 0, 1, \ldots, \tag{2}$$

where T represents the iteration matrix, which remains constant throughout the iterative process and c is a constant vector. Currently, stationary iterative methods are commonly used as preconditioners.[4]

The iteration process of nonstationary iterative solvers is more complicated and cannot be straightforwardly expressed by a matrix. They include approaches such as the algebraic multigrid method (AMG),[31] which is usually employed as the preconditioner for iterative solvers. Krylov subspace methods are recognized as the most effective and widely used among nonstationary iterative solvers. Therefore, in this study, when referring to iterative solvers, we primarily denote Krylov subspace methods, which fall within the framework of the general projection method.[4]

In this portion, we elaborate on the general projection method. Let A in Eq. (1) be an $n \times n$ sparse real matrix, and let K_m and L_m be two m-dimensional subspaces of $\mathbb{R}_n$ ($m < n$). Given an initial guess x_0 for the solution x, the projection method onto the subspace K_m and orthogonal to L_m is aimed at finding an approximate solution $\tilde{x}$ in the affine space $x_0 + K_m$ by enforcing the condition that the new residual vector be orthogonal to L_m. In other words, the general framework of projection based methods can be expressed as

$$r_0 = b - Ax_0, \tag{3}$$

$$\tilde{x} = x_0 + \delta, \quad \delta \in K_m, \tag{4}$$

$$(r_0 - A\delta, w) = 0, \quad \forall w \in L_m. \tag{5}$$

There are two kinds of projection methods: the orthogonal projection and the oblique projection methods. In the orthogonal method, the subspace L_m is the same as K_m, whereas in the latter one, L_m differs from K_m and might not have any connection to it.

Algorithm 1 shows basic procedures of the projection method.

Krylov subspace methods are essentially projection based methods, with

$$K_m = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}, \tag{6}$$

where $r_0 = b - Ax_0$, and the subspace K_m is known as the Krylov subspace. Depending on how the constraint subspace L_m is constructed, Krylov subspace methods can be broadly categorized into three groups:

- $L_m = K_m$. In this scenario, the constraint subspace L_m coincides with K_m, representing orthogonal projection based methods. Typical methods include the conjugate gradient method (CG),[32] which is applicable for symmetric positive definite linear systems.
- $L_m = AK_m$. In this context, the constraint subspace L_m is formed by applying a linear transformation to K_m. Representative methods include the generalized minimum residual method (GMRES)[33] and the conjugate residual method (CR).[34]
- $L_m = A^TK_m$. Similar to the previous case, but the linear transformation between L_m and K_m is effected by the matrix A^T, rather than A itself. Widely used

Algorithm 1. Procedures of projection method

1: Select an initial guess for x
2: **repeat**
3: Select subspaces K_m and L_m, respectively
4: Choose bases $V = [v_1, ..., v_m]$ and $W = [w_1, ..., w_m]$ for K_m and L_m
5: $r =: b - Ax$
6: $y =: (W^TAV)^{-1}W^Tr$
7: $x =: x + Vy$
8: **until** Convergence

algorithms falling into this category include the biconjugate gradient method (BiCG),[35] the quasi-minimal residual method (QMR),[36] etc.

2.2. *Preconditioning methods*

The convergence behavior of Krylov subspace methods is predominantly influenced by the eigenvalue distribution of the coefficient matrix.[2] For example, a smaller condition number of the matrix leads to better convergence properties for symmetric positive definite systems. Preconditioning methods are employed to optimize the eigenvalue distribution of the coefficient matrix, which involves several steps.[4] At first, a matrix M, known as the preconditioner, is identified. Then, the original linear system $Ax = b$ is transformed into a new system $M^{-1}Ax = M^{-1}b$. Similarly, the preconditioner M can be applied on the other side, resulting in $AM^{-1}u = b$, where $x = M^{-1}u$. If the preconditioner is available in a factored form as $M = M_L M_R$, the new linear system after preconditioning takes the form $M_L^{-1}AM_R^{-1}u = M_L^{-1}b$, with $x = M_R^{-1}u$. Finally, these preconditioned linear systems are solved using various Krylov subspace methods, such as CG or GMRES, resulting in corresponding Preconditioned Krylov subspace methods.

As numerical computation progresses, various methods have emerged for constructing the preconditioner M and these preconditioners are primarily represented by the following types:

- **Incomplete Factorization Preconditioners**
  These methods rely on the incomplete factorization of the coefficient matrix A. Examples include incomplete LU preconditioner (ILU), incomplete Cholesky preconditioner (ICC), etc.[4]
- **Preconditioners from Stationary Iteration**
  These preconditioners are derived from stationary iteration methods, and are usually built from specific parts of coefficient matrices such as the diagonal, upper triangular, or lower triangular part. Examples include the Jacobi preconditioner, successive over-relaxation preconditioner (SOR), etc.[4]
- **Approximate Inverse (AI) Preconditioners**
  These methods seek to find the matrix M that directly approximates the inverse of the coefficient matrix A by solving a minimization problem.[2] Then M is used to directly transform the original linear system into the form $MAx = Mb$, rather than using M^{-1}.
- **Preconditioners Based on Multigrid**
  These preconditioners utilize multigrid techniques. One of the most popular examples is the algebraic multigrid (AMG),[31] which is developed to solve matrix equations using the principles of usual multigrid methods.[37]
- **Preconditioners Based on Domain Decomposition**
  Domain decomposition methods involve dividing the problem domain into subdomains and solving them iteratively. Prevalent examples include the additive Schwarz method (ASM)[38] and block Jacobi preconditioner.[4]

PETSc[5] is currently the most powerful library for solving sparse linear systems, covering nearly all available iterative solvers and preconditioners, with over 100 possible combinations of the two. We utilize PETSc for solving all sparse linear systems in SolverSet.

3. Dataset Construction

In this section, we begin by providing an overview of the SuiteSparse Matrix Collection[25] and explaining the rationale behind creating a new benchmark dataset. Next, we present the matrix generation tool and how we use the tool to produce matrices, along with their properties. Finally, we elaborate on the development of SolverSet, the benchmark dataset for facilitating the modeling of the automatic selection of the optimal iterative solver and preconditioner for large-scale sparse linear systems.

3.1. *Overview of SuiteSparse Matrix Collection*

The SuiteSparse Matrix Collection is widely acknowledged as the standard repository for sparse matrices. Table 1 illustrates several representative domains along with their corresponding number of matrices, including computational fluid dynamics (CFD), structural engineering and electromagnetics. Matrices in these domains are usually generated from the numerical solution of PDEs. Thus they are well-suited for constructing sparse linear systems. However, matrices from several domains are not suitable for building sparse linear systems such as those found in linear programming problem. Furthermore, when considering matrices themselves from the SuiteSparse Matrix Collection, many of them are inappropriate for constructing sparse linear systems. These include nonsparse matrices (of array type), nonsquare matrices, or matrices of pattern type. By excluding such matrices, the subset suitable for constructing sparse linear systems is significantly reduced, with

Table 1. Representative domains and the number of matrices within these domains in SuiteSparse Matrix Collection. 2D/3D Geometry refers to whether the problem's domain is in 2-dimensional or 3-dimensional space.

Domains	Matrix number	2D/3D geometry
Linear programming problem	342	No
Combinatorial problem	299	No
Undirected graph	267	No
Circuit simulation problem	259	No
Directed graph	156	No
Optimization problem	138	No
Random graph	98	No
Optimal control	91	No
Structural problem	302	Yes
Computational fluid dynamics problem	185	Yes
Electromagnetics problem	53	Yes

the number of matrices dropping below 1,700. Additionally, the majority of matrices in the SuiteSparse Matrix Collection are of small size, as is shown in Fig. 1(a).

In summary, the SuiteSparse Matrix Collection only offers a small number of matrices suitable for building sparse linear systems, and these matrices are also of small size, making them inappropriate for modeling the automatic selection of the optimal iterative solvers and preconditioners. Hence, it is essential to develop a new large-scale benchmark dataset for the automatic selection of the optimal combination of iterative solvers and preconditioners for large-scale sparse linear systems.

3.2. *Matrix generation tool*

The numerical solution to PDEs from scientific and engineering applications serves as the primary source of large-scale sparse matrices.[39,40] Therefore, in this work, we build the matrix generation tool based on the numerical solution to PDEs. General numerical methods for solving these PDEs include the finite difference method (FDM), finite element method (FEM) and finite volume method (FVM).[41] These methods typically involve solving sparse linear systems of the form $Ax = b$. Since FDM is primarily for simpler and less frequent applications, we in this work rely on FEM and FVM to solve PDEs and produce sparse matrices in the matrix generation tool. The architecture of the proposed matrix generation tool is illustrated in Fig. 2, which comprises three main components: **Basic Sample Programs**, **Config Module** and **Code Generation Module**.

Basic Sample Programs are well-defined programs that solve various types of PDEs. During the execution of these programs, sparse linear systems can be constructed and solved. Since we produce sparse matrices through solving PDEs by using FEM and FVM, these sample programs stem from FEM based software

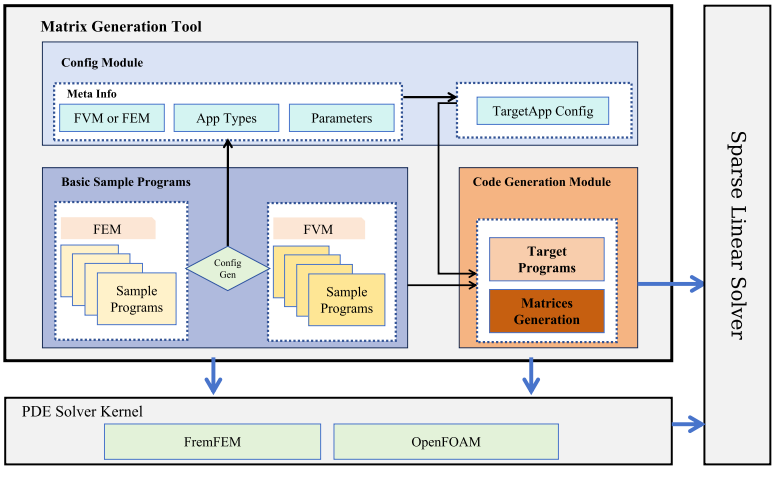


Fig. 2. The architecture of the proposed matrix generation tool. It utilizes sample programs and application configures to generate matrices of different scales and sparsity patterns.

FreeFEM[42] and FVM based software OpenFOAM.[43] FreeFEM is a leading PDE solver developed in C++, which provides programming interfaces for the definition and resolution of PDEs through FEM. It excels in solving nonlinear multi-physics PDE systems across different dimensions and boundary domains. OpenFOAM stands out as a widely utilized open source software based on FVM, offering an extensive array of functionalities to address a wide spectrum of problems, ranging from intricate fluid dynamics involving chemical reactions, turbulence and heat transfer, to disciplines such as acoustics, solid mechanics and electromagnetics. Both OpenFOAM and FreeFEM offer abundant sample programs derived from different real-world applications, which we incorporate as basic sample programs in the matrix generation tool.

Config Module contains the Meta Info of all basic sample programs, which includes the list of programs in use, application types, configurable parameters, etc. The Meta Info is generated by the Config Gen program, which retrieves all basic sample programs and extracts relevant information to build it. Moreover, the Config Module contains the TargetApp Config. Before beginning the matrix generation process, users utilize the TargetApp Config to set the generation method (FVM or FEM) and the target basic sample programs upon which the generation process is based, as well as configurable parameters such as the problem domain definition and grid resolution.

Code Generation Module is employed to generate target programs and produce matrices by executing these programs. It first retrieves TargetApp Config in the Config Module. Then, based on the specified configuration information and the basic sample programs upon which it relies, it executes code generation process to produce Target Programs. Code Generation Module is able to generate many new target programs with different parameters and configurations. Utilizing the Matrices Generation interface, we can export numerous large-scale sparse matrices by running these target programs based on packages FreeFEM and OpenFOAM.

Table 2. Descriptions of several curated basic sample programs.

Sample program	Source	Brief description
EpPoisson.edp	FreeFEM	A solver for the Poisson equation, which is a PDE for describing phenomena such as static electric fields or steady-state heat conduction, commonly used to relate the gradient of a field to a source, such as charge density or heat source.
Laplace3d.edp	FreeFEM	A solver for the three-dimensional Laplace equation, which is a PDE describing the relationship between the Laplacian operator of a field and the field itself in the absence of any source terms.
Stokes.edp	FreeFEM	A solver for the Stokes equations, which are a set of PDEs that describe the relationship between the velocity field and pressure field of fluid flow.

Table 2. (*Continued*)

Sample program	Source	Brief description
icoFoam.C	OpenFOAM	A simple steady-state iterative coupling algorithm for solving incompressible flow problems, suitable for handling simple flow problems involving incompressible fluids.
potentialFoam.C	OpenFOAM	A solver for the potential flow equations that describe the irrotational nature of fluid flow and are applicable to flow situations characterized by low Mach numbers and high Reynolds numbers.
scalarTransportFoam.C	OpenFOAM	A solver for the scalar transport equation that describes the transport and diffusion of scalars, such as concentration and temperature, in fluid flow. It is commonly employed for simulating the transport of various substances in fluids.

The matrix generation tool incorporates 21 sample programs from FreeFEM and 19 sample programs from OpenFOAM as its basic sample programs, including those governed by well-known equations such as the Stokes equation and Poisson equation. In total, there are 8,482 sparse matrices by FEM methods, and 4,169 sparse matrices from FVM methods. These matrices constitute a new matrix collection, currently comprising 12,651 large-scale sparse matrices, the scale of which can be seen in Fig. 1 (b). If further requirement in both the quantity and diversity of matrix properties is desired, more sample programs from various domains can be integrated into the matrix generation tool in the future. Table 2 presents a selection of the basic sample programs utilized in the matrix generation tool, along with their corresponding brief descriptions.

3.3. *Properties of matrices*

In this work, we employ the number of nonzero (nnz) elements, sparsity ratio, pattern and value symmetry and diagonal dominance ratio to demonstrate properties of generated matrices.

Figure 3 showcases the range of nnz elements for all matrices, which mainly determines the computational load of different kinds of sparse matrix computation tasks. It is evident that the majority (93.7%) of matrices contain over 100 K nnz elements, a quantity substantial for various sparse computation tasks. Figure 4 shows the sparsity distribution of our generated matrices. We can observe that the sparsity degree of our generated matrices relatively large, generally above 99.0%, which conforms to the typical sparsity degree of sparse matrices in the field of scientific and engineering computing, and further demonstrates the reliability and rationality of the matrix generation tool.

Pattern symmetry and value symmetry are two important metrics employed for assessing the symmetry of sparse matrices. They significantly influence the storage and computational optimization of sparse matrices. Figure 5 illustrates the pattern symmetry and value symmetry of all matrices of generated matrices. We can see that

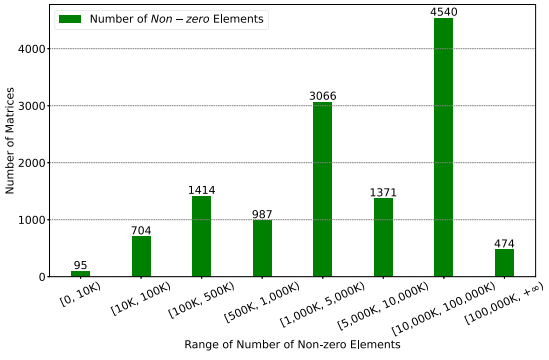


Fig. 3. The nonzero element distribution of generated matrices. The x-axis represents the range of the number of nonzero elements (in thousands, where $K = 1,000$). The y-axis represents the number of matrices within each range.

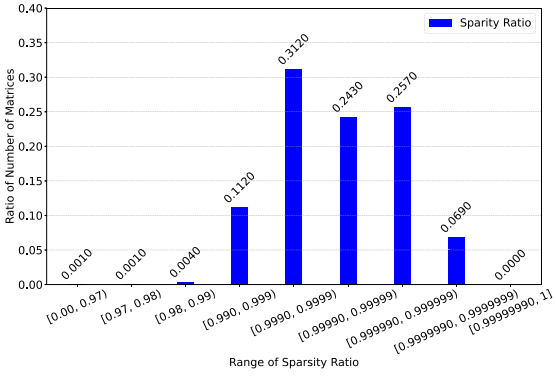


Fig. 4. The sparsity ratio distribution of generated matrices. The sparsity ratio, defined as the ratio of zero elements to the total number of elements, is shown on the x-axis, grouped into specific ranges. The y-axis represents the proportion of matrices in each sparsity ratio range relative to the total number of matrices.

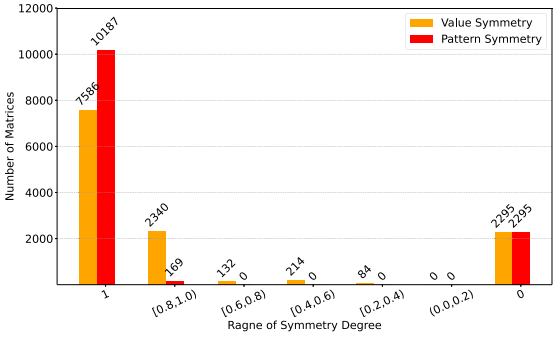


Fig. 5. The symmetry degree distribution of generated matrices. The x-axis represents the ranges of numerical and pattern symmetry ratios. The numerical symmetry ratio is the proportion of elements where $A_{ij} = A_{ji}$, while the pattern symmetry ratio is the proportion of nonzero positions where both A_{ij} and A_{ji} are nonzero. The y-axis shows the number of matrices in each range.

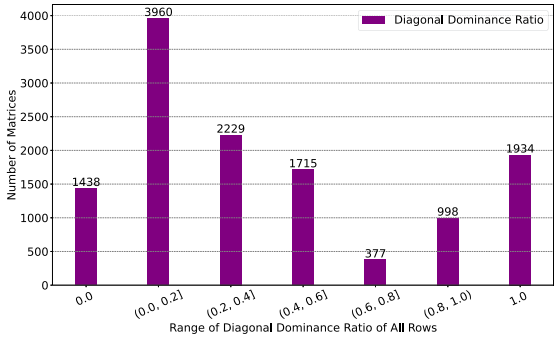


Fig. 6. The diagonal dominance ratio distribution of generated matrices. The x-axis represents the ranges of the diagonal dominance ratio, defined as the proportion of rows in a matrix that satisfy the diagonal dominance condition. The y-axis represents the number of matrices within each range.

a significant portion of matrices exhibit nice symmetry, while many others are asymmetric. These symmetric and asymmetric matrices are highly suitable for studying various symmetric and nonsymmetric sparse matrix algorithms.

For each row in a matrix, if the absolute value of the diagonal element is greater than the sum of the absolute values of all off-diagonal elements in that row, the row is said to satisfy diagonal dominance.[44] To calculate the diagonal dominance ratio for a given matrix, we count the number of rows that satisfy this condition and then divide that number by the total number of rows in the matrix. Diagonal dominance ratio significantly impacts the convergence difficulty of iterative solvers. Sparse linear systems with small diagonal dominance ratio tend to have poor convergence, whereas those with larger diagonal dominance exhibit better convergence properties. The matrix is called diagonally dominant if the diagonal dominance ratio is 1.0, which can be defined as follows:

$$|a_{ii}| - \sum_{j=1, j\neq i}^{n} |a_{ij}| > 0, \quad i = 1, 2, \ldots, n. \tag{7}$$

The diagonal dominance ratio for all matrices in SolverSet is shown in Fig. 6. It can be observed that SolverSet contains matrices with both high and low ratios of diagonal dominance. Overall, the distribution of diagonal dominance ratios is relatively balanced. This indicates that the convergence behavior of the generated sparse linear systems varies significantly, which numerically demonstrates the diversity of the SolverSet samples.

3.4. *Building SolverSet*

In this part, we describe the construction process of SolverSet based on matrices produced by the matrix generation tool. To construct SolverSet, the initial step involves generating the right-hand side vector b for each matrix (denoted as A) that we produce. For each sparse matrix, this is achieved by randomly assigning values

within the interval $[0, 1)$ to all elements in solution vector x, which is then multiplied by the sparse matrix A to yield the right-hand side vector b. By this method, we have built 12,651 sparse linear systems.

PETSc[5] refers to the Portable, Extensible Toolkit for Scientific Computation, offering a variety of iterative solvers and preconditioners for solving large-scale sparse linear systems. We utilize PETSc to solve all sparse linear systems in SolverSet. In terms of candidate combination of iterative solvers and preconditioners, we consider all feasible combinations available in PETSc (version 3.17.3), excluding those derived from third-party libraries. Incompatible combinations of solvers and preconditioners are disregarded.

In PETSc, the convergence test is based on the l_2-norm of the residual. Convergence is detected at iteration k if

$$\|r_k\|_2 < \max(\text{rtol} \times \|b\|_2, \text{atol}), \tag{8}$$

where $r_k = b - Ax_k$, rtol is the relevant tolerance, and atol is the absolute tolerance. After referring to sample program settings in OpenFOAM, FreeFEM and PETSc, we set rtol and atol to 1e−6 and 1e−50, respectively. The maximum number of iterations is set to 1,000, ensuring that all iterative solvers undergo a maximum of 1,000 iterations when solving sparse linear systems.

The Huawei TaiShan server is utilized as the hardware platform for solving all sparse linear systems, and Table 3 gives a brief description of it. To ensure a fair comparison of convergence speed at the algorithmic level, default parameters are set for all solvers and preconditioners uniformly.

After determining the software and hardware platforms, as well as the solution settings, we employ candidate combinations to solve 12,651 sparse linear systems, which is also the most time-consuming step of constructing the SolverSet dataset, spanning several months.

Upon completing all the solving processes, we collect solution data for each sparse linear system and its solution results of different combinations, which include convergence status, final accuracy, solution time and iteration count. Based on the data, the optimal combination of iterative solvers and preconditioners can be identified for each sparse linear system. Specifically, for each sparse linear system, we examine the results of all candidate combinations. Among those that successfully converge (i.e., meet the predefined convergence criteria), we select the combination with the

Table 3. Descriptions of Huawei Taishan server.

Parameters	Huawei Taishan server
Processor	ARM v8.2 architecture, 2 Kunpeng 920
Clock cycle	2.6 GHz
Operating system	CentOS Linux 4.18.0.aarch64
Memory	256 GB
NUMA node(s)	4 NUMA, 128 cores
Theoretical throughput	1331.2 Gflops

shortest solution time as the optimal one. By enumerating optimal combinations across all sparse linear systems, a set of optimal combinations can be obtained. In this work, the optimal combination refers to the one meeting the convergence criteria while achieving the shortest solution time. After examining all 12,651 linear systems, we find that a total of 19 distinct combinations have been selected as the optimal solution at least once. This set of 19 combinations constitutes the complete set of labels for the classification task. Finally, through these procedures, we obtain a labeled dataset consisting of 12,651 data samples named SolverSet. The sparse matrices, right-hand side vectors and labels from SolverSet will be open-sourced to improve dataset reproducibility and impact.

The automatic selection of the optimal combination for a given sparse linear system can be viewed as a multi-classification machine learning problem. Each sparse linear system and its corresponding optimal combination form a piece of sample data. The total number of distinct optimal combinations that have appeared across all sample data is 19, which means that the automatic selection of the optimal combination is regarded as a 19-class classification problem.

4. DL-Solver

4.1. *Feature extractor*

For a given sparse linear system $Ax = b$, with the software, hardware and related parameters determined, the iterative solution time depends on the number of iterations and the time consumed per iteration. Algorithm 2 demonstrates the solution process of the widely used preconditioned conjugate gradient (PCG)[4] method for symmetric positive definite linear systems.

We observe that in each iteration of PCG, the primary operations involve sparse matrix-vector multiplication (SpMV), vector inner product, preconditioning and AXPY ($aX + Y$), which determine the execution time of each single iteration. The time required for these operations is primarily determined by the number and distribution of nonzero elements and the matrix size. It is worth noting that, apart

Algorithm 2. Preconditioned conjugate gradient

1: Compute $r_0 := b - Ax_0, z_0 = M^{-1}r_0$, and $p_0 := z_0$
2: **for** $j = 0, 1, ...,$ until convergence **do**
3: $\alpha_j := (r_j, z_j)/(Ap_j, p_j)$
4: $x_{j+1} := x_j + \alpha_j p_j$
5: $r_{j+1} := r_j - \alpha_j Ap_j$
6: $z_{j+1} := M^{-1}r_{j+1}$
7: $\beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$
8: $p_{j+1} := z_{j+1} + \beta_j p_j$
9: **end for**

from some complicated preconditioning methods (e.g., geometric AMG (GAMG) and ASM), some preconditioning operations are essentially an SpMV operation.

The determinants of the iteration number in iterative methods are not entirely certain, but from a general numerical computation perspective, they are typically related to factors such as the distribution of eigenvalues, diagonal dominance, symmetry and other similar properties, which are fundamentally related to values of nonzero elements.

The automatic selection of the optimal combination of iterative solvers and preconditioners aims to predict the combination that minimizes the execution time and meets the convergence criterion. From the above analysis, it is evident that predicting this involves considering both the number of iterations and the time taken per iteration. Based on the previous analysis, we extract relevant features from the coefficient matrix A, as shown in Table 4. These features can be divided into two categories: **features based on the pattern of nonzero elements** (Numbers 1–9) and **features based on values of nonzero elements** (Numbers 10–17). The former determines the amount of floating point operations in each single iteration process, while the latter is closely related to the number of iterations required for the solution of sparse linear systems. Furthermore, the computational complexity of these features is all linear with the nnz elements ($O(nnz)$), resulting in low computational overhead and facilitating the low-cost modeling.

Table 4. Features of sparse linear systems for the proposed model DL-Solver.

Number	Feature name	Brief description
1	row_num	The row (and column) number of the coefficient matrix
2	nnz	The number of the nonzero elements
3	nnz_ratio	The ratio of nonzero elements to the total number of elements
4	nnz_lower	The number of nonzero elements in the lower triangular part of the matrix
5	nnz_upper	The number of nonzero elements in the upper triangular part of the matrix
6	nnz_diagonal	The number of nonzero elements in the diagonal
7	ave_nnz_each_row	The average number of nonzero elements per row
8	max_nnz_each_row	The maximum number of nonzero elements per row
9	arr_nnz_each_row	The variance of the number of nonzero elements per row
10	max_value	The maximum absolute value of nondiagonal elements
11	max_value_diagonal	The maximum absolute value of nonzero elements on the diagonal
12	diagonal_dominant_ratio	The ratio of the number of rows exhibiting diagonal dominance to the total number of rows
13	is_symmetric	Is the matrix symmetric
14	pattern_symm	A ratio to measure the degree of symmetry in a sparse matrix pattern
15	value_symm	A ratio to measure the degree of symmetry in the values of a sparse matrix
16	row_variability	Logarithm of the maximum row-wise max-to-min absolute ratio
17	col_variability	Logarithm of the maximum column-wise max-to-min absolute ratio

4.2. *Model design*

The automatic selection of the optimal combination of iterative solvers and preconditioners aims to build a machine learning or deep learning model for predicting the optimal one for a given sparse linear system, which is a multi-class classification task. From the aforementioned feature analysis, it is evident that the model needs to predict the optimal combination through learning information from both pattern based features and value based features. The combination of these two types of features further enhances the representation capacity.

Inspired by the model (denoted as MLP-Solver) proposed by Funk *et al.*,[22] we redesign a multi-layer perceptron (MLP) based deep learning model named **DL-Solver** to predict the optimal combination of iterative solvers and preconditioners for a given sparse linear system. The input features of DL-Solver are shown in Table 4, and the architecture of the proposed DL-Solver is illustrated in Fig. 7.

DL-Solver operates as a four-stage pipeline. First, the input layer ingests the sparse linear system and standardizes it into the compressed row storage (CSR) format for efficient processing. Second, the feature extractor layer analyzes the CSR-formatted matrix to compute the 17 handcrafted features (Table 4), categorizing them into the two fundamental types: value-based features (e.g., condition number) that govern convergence speed and pattern-based features (e.g., nonzero count) that influence per-iteration cost. Third, the feature learning and crossing layer processes these 17 features through a multi-layer perceptron, applying nonlinear transformations

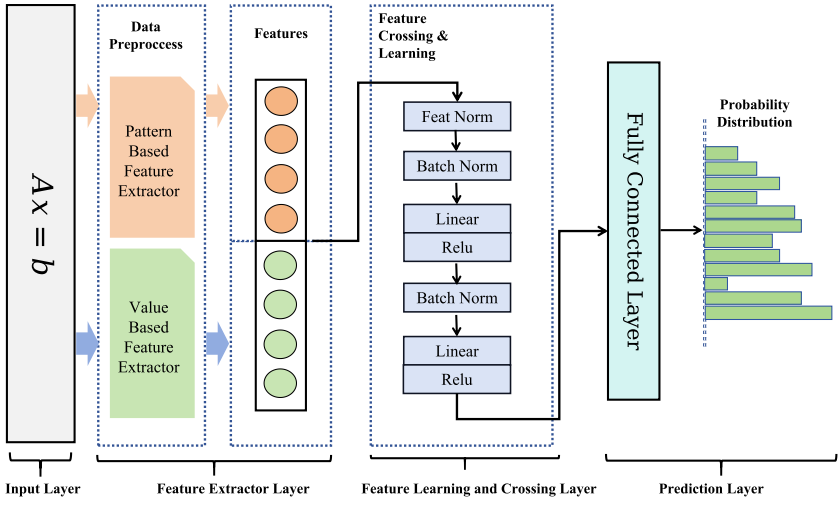


Fig. 7. The architecture of the proposed DL-Solver model. The model consists of four main components: (1) Input layer: reads the input sparse linear system and converts it into the compressed sparse row (CSR) format for efficient processing; (2) Feature extractor layer: extracts 17 handcrafted features from the CSR-formatted matrix; (3) Feature learning and crossing layer: a multi-layer perceptron (MLP) that learns nonlinear mappings and interactions between the extracted features; (4) Prediction layer: applies a softmax function to convert the learned features into a probability distribution over the 19 possible solver–preconditioner combinations, with the highest probability indicating the predicted optimal choice.

to learn complex, high-level interactions between the input features. Finally, the prediction layer applies a softmax function to convert the learned high-level representation into a probability distribution over all possible solver–preconditioner combinations, with the highest probability indicating the predicted optimal choice.

This sequential flow — from raw data ingestion to feature extraction, deep feature learning and final probabilistic prediction — forms the core workflow of DL-Solver. The use of the logarithmic loss function during training guides the model to refine this entire pipeline to maximize the accuracy of its final prediction.

The major advantage of the DL-Solver proposed in this work lies in its use of computationally efficient features (as shown in Table 4) and relatively simple model architecture, which will greatly enhance the practicality of the model. Despite its simplicity, DL-Solver achieves better performance than state-of-the-art methods. The details of the performance evaluation will be presented in the subsequent experimentation section.

5. Experimentation

5.1. *Model implementation*

The DL-Solver is implemented by Tensorflow 2.14.0,[45] based on CUDA 11.3. It is trained and evaluated on a heterogeneous computing system comprising an Intel(R) Xeon(R) Gold 5120 CPU and an Nvidia A100 GPU. The hyperparamters for the proposed DL-Solver are illustrated in Table 5.

5.2. *Datasets*

In this work, we describe the dataset for modeling the automatic selection of the optimal combination of iterative solvers and preconditioners for sparse linear systems. It consists of all 12,651 sparse linear systems from SolverSet. All of them are paired with their optimal combinations of iterative solvers and preconditioners, which serve as labels. We divide the dataset into the training set and the test set in a ratio of 10:1. Therefore, the training set contains a total of 11,386 samples, and the test set contains 1,265 samples. The label distribution of the training set and test set is shown in Table 6.

Table 5. Hyper parameter settings of DL-Solver.

Hyper parameter	Values
Epoch	256
Batch size	512
Optimizer	Adam
Learning rate	0.001
Cost function	Cross entropy
Dimension of input layer	17
Dimension of first linear layer	1024
Dimension of second linear layer	128
Dimension of output layer	19

Table 6. The label distribution of the training set and test set in SolverSet.

Label (Combination)	Training set	Test set	Total number
cg_eisenstat	208	26	234
symmlq_jacobi	878	92	970
symmlq_sor	567	59	626
bcgsl_none	2,040	233	2,273
cg_ilu	268	41	309
symmlq_icc	1,213	113	1,326
cr_jacobi	295	37	332
fgmres_gamg	232	25	257
fcg_gamg	341	34	375
gmres_gamg	633	71	704
minres_gamg	506	61	567
dgmres_none	621	81	702
cgs_gamg	37	6	43
cr_ilu	63	10	73
fbcgs_ilu	533	55	588
cg_bjacobi	203	19	222
bcgsl_asm	30	3	33
cr_eisenstat	601	60	661
fbcgs_jacobi	2,117	239	2,356
Sum	11,386	1,265	12,651

It is worth noting that some of the combinations we used may not be theoretically reasonable from a numerical computation perspective, yet they achieve good solution results in practice. For example, in the combination cg_ilu, CG is an iterative solver designed for symmetric positive definite systems, while the ILU preconditioner is essentially suited for nonsymmetric linear systems. However, experimental results show that cg_ilu yields the best solving performance for some sparse linear systems.

From Table 6, we can observe that the distribution of labels is imbalanced. Some combinations, for example, fbcgs_jacobi, bcgsl_none and symmlq_icc, correspond to a large number of training and test samples. In contrast, some combinations, such as bcgsl_asm, cgs_gamg, correspond to only a few dozen samples. The imbalance issue reflects, on one hand, that some combinations exhibit universality and generality in solving sparse linear systems. On the other hand, it also poses a challenge to the accuracy of automatic selection. In the future, we will also focus on labels with small number of samples, and use the matrix generation tool to create more matrices and sparse linear systems of these labels.

5.3. *Evaluation metrics*

Since the prediction of the optimal combination is modeled as a multi-class classification task, we adopt four metrics commonly used in this task, including Accuracy (Acc), Macro Precision (MP), Macro Recall (MR) and Macro-F1-score (F1). The MP, MR and Macro-F1 are calculated by the macro average of them respectively,

with all classes being equally important. The Acc is able to reflect the overall prediction performance, but might be dominated by classes with a large number of training data. Hence, MP, MR and Macro-F1 metrics are also employed for a comprehensive evaluation of prediction performance on the test set.

5.4. *Baseline models*

Several baseline models are employed to evaluate the validity and usability of the proposed SolverSet, and they also serve as comparison objects for the DL-Solver model that we put forward in this work.

- **XGBoost**. XGBoost (eXtreme Gradient Boosting)[46] is a powerful tree based ensemble learning algorithm, which can be seen as an improved version of AdaBoost method[17] and used to perform multi-class classification tasks.
- **MLP-Based Method**. Funk *et al.*[22] proposed a model (denoted as MLP-Solver in this work), which is based on multi-layer perceptron (MLP), to predict the optimal solver for sparse linear systems.
- **CNN-Based Method**. Inspired by the convolutional neural network (CNN) used by Yamada *et al.*,[21] we implement a CNN based model denoted as CNN-Solver, as one of the baseline models.

Previous work has attempted more complex graph neural network (GNN) for the automatic selection of iterative solvers and preconditioners. However, due to the GNN not demonstrating satisfactory prediction performance and the significant overhead associated with handling large-scale matrices in SolverSet, it is not considered as a baseline model.

5.5. *Experimental results*

We describe the predictive performance of the DL-Solver proposed in this work, as well as the baseline methods, on the test set from SolverSet. As can be seen in Table 7, there are two parts of performance results in it. The first part demonstrates

Table 7. The overall performance of different methods for predicting the optimal combination of iterative solvers and preconditioners.

Method	Acc (%)	MP (%)	MR (%)	F1 (%)
XGBoost	47.19	34.54	24.65	24.96
CNN-Solver	66.09	47.66	43.53	43.33
MLP-Solver	76.60	59.72	56.41	57.09
DL-Solver	78.74	63.19	59.12	60.25
Ablation study				
w/o value-based features	51.46	34.06	29.46	29.49
w/o pattern-based features	68.62	45.61	42.17	41.70

the performance of the proposed model DL-Solver and three baseline models: XGBoost, CNN-Solver and MLP-Solver. The second part shows the result of the ablation study.

From Table 7, we can observe that the proposed DL-Solver achieves 78.74%, 63.19%, 59.12% and 60.25% in Acc, MP, MR and F1, outperforming three baseline methods in all four metrics. Specifically, DL-Solver achieves improvements of 2.14%, 3.47%, 2.71% and 3.16% in Acc, MP, MR and F1, respectively, compared MLP-Solver, which is considered as the best method available. Notably, the MLP-Solver introduced by Funk *et al.*[22] achieved approximately 57% predictive accuracy in classifying seven categories in their study by using the SuiteSparse Matrix Collection. This demonstrates that by utilizing the new large-scale dataset SolverSet, MLP-Solver shows a significant improvement in prediction performance, highlighting SolverSet's advantage in enhancing state-of-the-art methods compared to other datasets. CNN-Solver achieves 66.09%, 47.66%, 43.53% and 43.33% in four metrics, which can also be considered a reasonably good prediction result in terms of 19 labels. The relatively poor performance achieved by XGBoost in Table 7 indicates that traditional machine learning models might not be capable of effectively predicting the optimal combinations of iterative solvers and preconditioners, especially with 19 candidate combinations.

DL-Solver's key advantage over current state-of-the-art models lies in its fine-grained feature engineering and selection. It systematically categorizes input features into two distinct types: features based on the pattern of nonzero elements (Numbers 1–9) and features based on values of nonzero elements (Numbers 10–17). This categorization is grounded in numerical theory, as value-based features are closely related to the total number of iterations required for convergence, while pattern-based features directly influence the computational cost of a single iteration. Since the overall solver performance is determined by the product of these two factors ($T_{\text{totaltime}} = \text{iterations} \times T_{\text{timeperiteration}}$), DL-Solver's ability to explicitly learn and jointly model both aspects provides a more comprehensive and physically meaningful representation of the problem. Furthermore, DL-Solver adopts an architecture similar to the MLP-Solver, utilizing a relatively simple two-layer MLP structure. This choice, combined with optimized hyperparameters, allows the model to efficiently learn the nonlinear relationships within and between these two critical feature categories, leading to enhanced predictive performance.

5.6. *Ablation study*

In this section, we demonstrate the enhancement of model performance of DL-Solver by eliminating value based features and pattern based features.

DL-Solver utilizes pattern based features (numbered from 1 to 9 in Table 4) and value based features (numbered from 10 to 17 in Table 4) for modeling. We argue that these two types of features respectively affect the the time consumption per iteration and number of iterations in iterative solution. Therefore, we remove one of

these two types of features to train the models and validate their effectiveness. As shown in Table 7, w/o pattern-based features denotes that we remove features 1–9 within Table 4 and w/o value-based features denotes the model where features 10–17 within Table 4 are removed. From the experimental results, removing either values based features or pattern based features exhibits a significant performance decrease compared to DL-Solver. Among them, the decline in model performance of removing value based features is larger than that without pattern based features. The results suggest that in the prediction of the combination of iterative solvers and preconditioners, features related to values of nonzero elements play a more significant role, as they directly influence the number of iterations in iterative solution.

6. Related Work

The advancement of machine learning, particularly deep learning, offers the potential to automate the process of selecting iterative solvers and preconditioners for solving sparse linear systems. This section presents an overview of research exploring the utilization of machine learning or deep learning techniques for the automatic selection of iterative solvers and preconditioners.

Bhowmick *et al.*[6] first introduced machine learning for the automatic selection of solvers for sparse linear systems. This work meticulously demonstrated the process of organizing datasets, identifying pertinent features, defining selection criteria for various iterative solvers and training machine learning models such as Adaboost. Subsequently, Bhowmick *et al.*[7] continued to advance a cost-effective approach for computing and selecting features, enabling the efficient construction of traditional machine learning models (such as Nearest Neighbor, Naive Bayes and SVM) that were employed for predicting the optimal iterative solver and preconditioner. Eijkhout *et al.*[8] introduced a theoretical framework for recommending numerical methods, along with libraries that have been developed to embody concepts in the framework. Eller *et al.*[9] employed machine learning algorithms to forecast the optimal combination of iterative solvers and preconditioners for transient simulations, typically dealing with larger-scale linear systems compared to those found in the SuiteSparse Matrix Collection. Sakurai *et al.*[10] introduced a technique which utilized past residuals to forecast the most effective combination of iterative methods and preconditioners. Leveraging the attributes of sparse linear systems alongside various combinations of iterative solvers and preconditioners, Jessup *et al.*[11–14] initiated the Lighthouse project that employed machine learning models to forecast the efficacy of specific combinations of iterative solvers and preconditioners, and it also proposed a performance model to characterize the communication overhead of different iterative methods, aiming to select the most suitable one on distributed platforms. Recently, Sun *et al.*[15] proposed a feature selection method for enhancing machine learning models that predict Krylov subspace solvers. Zabegaev *et al.*[47] proposed an adaptive linear solver selection framework for transient multiphysics simulations. Its core innovation is treating the linear solve at each time step as an independent decision

task, significantly shortening the feedback loop. The aforementioned studies concentrate on investigating traditional statistical machine learning algorithms to forecast iterative solvers and preconditioners. These techniques offer strong interpretability and can depict the connection between sparse linear systems and iterative solvers or preconditioners to a certain extent. For instance, the symmetry of the coefficient matrix holds significance in the choice of iterative solvers and preconditioners.

To enhance the utilization of the sparsity distribution inherent in the coefficient matrix, researchers have commenced integrating deep learning models for the prediction of iterative solvers and preconditioners. Holloway and Chen[18] proposed using neural networks to predict the behavior of preconditioned iterative solvers, i.e., predicting whether they can solve the problem. Kuefler and Chen[19] proposed reinforcement learning for solving sparse linear systems, emphasizing that reinforcement learning could be used not only to recommend a solver but also solve sparse linear systems, without requiring label information. Yeom *et al.*[20] drew inspiration from the field of natural language processing and constructed a performance vector space to map combinations of iterative solvers and preconditioners into this space. During prediction, the linear equation system is projected into this space, and the nearest neighbor algorithm is employed to determine the optimal combination. Yamada *et al.*[21] utilized CNN to extract information directly from the coefficient matrix and predict the optimal preconditioner for a given sparse linear system. More specifically, it transformed the coefficient matrix into a fixed-size full-color image consisting of three channels. Subsequently, CNN operations were applied to the color image to gather more information to aid in the prediction of the preconditioner. Funk *et al.*[22] investigated various deep learning models, such as MLP and fully connected neural network (FCNN), to predict the optimal iterative solver for sparse linear systems. Tang *et al.*[23] employed GNNs to forecast the optimal combination of iterative solver and preconditioner. This approach was inspired by the recognition that the nonzero elements of coefficient matrices can be accurately represented by a graph, and GNNs possess the capability to glean domain knowledge from such graphs.

7. Conclusion

In this paper, we propose to solve the problem of the automatic selection of the optimal combination of iterative solvers and preconditioners for sparse linear systems by introducing a benchmark dataset named SolverSet. In particular, we develop a matrix generation tool to generate large-scale matrices. Using these matrices, we create the large-scale dataset SolverSet for modeling the automatic selection of the optimal combination of iterative solvers and preconditioners. Then we conduct an analysis to identify relevant matrix features and propose a deep learning-based model DL-Solver to predict the optimal combination of iterative solvers and preconditioners for a given sparse linear system. Finally, we evaluate our proposed model DL-Solver and several baseline models through extensive experiments.

The experimental evaluation results demonstrate that SolverSet is a valuable benchmark dataset for modeling the automatic selection of the optimal combination of iterative solvers and preconditioners. The results also show that the proposed DL-Solver outperforms current baseline methods in prediction performance on the test set, and has the potential to be applied in real-world applications. In summary, this work makes three main contributions: (1) the construction of the large-scale SolverSet dataset, which addresses the data scarcity challenge in this field; (2) the proposal of the DL-Solver model, which demonstrates the effectiveness of deep learning for this task; and (3) the systematic evaluation framework established for future research. This study lays a solid foundation for data-driven approaches in iterative solver selection. Future work will focus on exploring more advanced deep learning architectures and expanding the dataset to cover a wider range of problem domains.

Acknowledgments

The research was partially funded by the Major Projects of Xiangjiang Laboratory (No. 22xj01011) and the Key Program of National Natural Science Foundation of China (grant nos. U21A20461, 92055213 and 62227808).

ORCID

Hantao Xiong https://orcid.org/0009-0001-5972-4199
Wangdong Yang https://orcid.org/0000-0003-2681-7898
Shengle Lin https://orcid.org/0000-0003-3329-0924
Weiqing He https://orcid.org/0009-0006-5219-6289
Keqin Li https://orcid.org/0000-0001-5224-4048
Kenli Li https://orcid.org/0000-0002-2635-7716

References

1. W. F. Ames, *Numerical Methods for Partial Differential Equations* (Academic Press, 2014).
2. M. Benzi, Preconditioning techniques for large linear systems: A survey, *J. Comput. Phys.* **182** (2002) 418–477.
3. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, 1994).
4. Y. Saad, *Iterative Methods for Sparse Linear Systems* (SIAM, 2003).
5. S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang and J. Zhang, PETSc Web page, 2023, https://petsc.org/.
6. S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes and D. Keyes, Application of machine learning to the selection of sparse linear solvers, *Int. J. High Perf. Comput. Appl.* (2006).

7. S. Bhowmick, B. Toth and P. Raghavan, Towards low-cost, high-accuracy classifiers for linear solver selection, *Computational Science — ICCS 2009*, eds. G. Allen *et al.*, *Lecture Notes in Computer Science*, Vol. **5544** (Springer, Berlin, Heidelberg, 2009), pp. 463–472.
8. V. Eijkhout and E. Fuentes, Machine learning for multi-stage selection of numerical methods, *New Advances in Machine Learning* (Intech, 2010), pp. 117–136.
9. P. R. Eller, J.-R. C. Cheng and R. S. Maier, Dynamic linear solver selection for transient simulations using machine learning on distributed systems, *2012 IEEE 26th Int. Parallel and Distributed Processing Symp. Workshops & PhD Forum* (IEEE, 2012), pp. 1915–1924.
10. T. Sakurai, T. Katagiri, H. Kuroda, K. Naono, M. Igai and S. Ohshima, A sparse matrix library with automatic selection of iterative solvers and preconditioners, *Procedia Comput. Sci.* **18** (2013) 1332–1341.
11. P. Motter, K. Sood, E. Jessup and B. Norris, *Lighthouse: An Automated Solver Selection Tool* (ACM, New York, 2015), pp. 16–24.
12. E. Jessup, P. Motter, B. Norris and K. Sood, Performance-based numerical solver selection in the lighthouse framework, *SIAM J. Sci. Comput.* **38** (2016) S750–S771.
13. K. Sood, B. Norris and E. Jessup, Comparative performance modeling of parallel preconditioned krylov methods, *2017 IEEE 19th Int. Conf. on High Performance Computing and Communications; IEEE 15th Int. Conf. on Smart City; IEEE 3rd Int. Conf. on Data Science and Systems (HPCC/SmartCity/DSS)* (IEEE, 2017), pp. 26–33.
14. K. Sood, Iterative solver selection techniques for sparse linear systems, thesis, University of Oregan (2019).
15. H.-B. Sun, Y.-F. Jing and X.-W. Xu, A new matrix feature selection strategy in machine learning models for certain krylov solver prediction, *J. Classif.* **42** (2024) 72–89.
16. C. Cortes and V. Vapnik, Support-vector networks, *Mach. Learn.* **20** (1995) 273–297.
17. Y. Freund and R. E. Schapire, A short introduction to boosting, *J. Jpn. Soc. Artif. Intell.* **14** (1999) 771–780.
18. A. Holloway and T.-Y. Chen, Neural networks for predicting the behavior of preconditioned iterative solvers, *Computational Science — ICCS 2007. ICCS 2007*, eds. Y. Shi *et al.*, *Lecture Notes in Computer Science*, Vol. **4487** (Springer, Berlin, Heidelberg, 2007), pp. 302–309.
19. E. Kuefler and T.-Y. Chen, On using reinforcement learning to solve sparse linear systems, *Computational Science — ICCS 2008. ICCS 2008*, eds. M. Bubak *et al.*, *Lecture Notes in Computer Science*, Vol. **5101** (Springer, Berlin, Heidelberg, 2008), pp. 955–964.
20. J.-S. Yeom, J. J. Thiagarajan, A. Bhatele, G. Bronevetsky and T. Kolev, Data-driven performance modeling of linear solvers for sparse matrices, *2016 7th Int. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)* (IEEE, 2016), pp. 32–42.
21. K. Yamada, T. Katagiri, H. Takizawa, K. Minami, M. Yokokawa, T. Nagai and M. Ogino, Preconditioner auto-tuning using deep learning for sparse iterative algorithms, *2018 Sixth Int. Symp. on Computing and Networking Workshops (CANDARW)* (IEEE, 2018), pp. 257–262.
22. Y. Funk, M. Götz and H. Anzt, Prediction of optimal solvers for sparse linear systems using deep learning, *Proc. 2022 SIAM Conf. Parallel Processing for Scientific Computing* (SIAM, 2022), pp. 14–24.
23. Z. Tang, H. Zhang and J. Chen, Graph neural networks for selection of preconditioners and krylov solvers, *NeurIPS 2022 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2022)* (The NeurIPS Foundation (proceedings published by Curran Associates, Inc.), 2022).
24. H. Zou, X. Xu and C.-S. Zhang, A survey on intelligent iterative methods for solving sparse linear algebraic equations, preprint (2023), arXiv:2310.06630.

25. T. A. Davis and Y. Hu, The university of florida sparse matrix collection, *ACM Trans. Math. Softw.* **38** (2011) 1–25.
26. R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett and J. J. Dongarra, Matrix market: A web resource for test matrix collections, *Quality of Numerical Software: Assessment and Enhancement* (Springer, Boston, 1997), pp. 125–137.
27. A. Musaelian, S. Batzner, A. Johansson and B. Kozinsky, Scaling the leading accuracy of deep equivariant models to biomolecular simulations of realistic size, *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (ACM, New York, 2023), pp. 1–12.
28. E. Merzari *et al.*, Exascale multiphysics nuclear reactor simulations for advanced designs, *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (ACM, New York, 2023), pp. 1–11.
29. Y. Fu *et al.*, Towards exascale computation for turbomachinery flows, preprint (2023), arXiv:2308.06605.
30. W. Wan *et al.*, 69.7-pflops extreme scale earthquake simulation with crossing multi-faults and topography on sunway, *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (ACM, New York, 2023), pp. 1–15.
31. J. W. Ruge and K. Stüben, Algebraic multigrid, in *Multigrid Methods* (SIAM, 1987), pp. 73–130.
32. M. R. Hestenes and E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems* (NBS Washington, DC, 1952).
33. Y. Saad and M. H. Schultz, Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7** (1986) 856–869.
34. D. G. Luenberger, The conjugate residual method for constrained minimization problems, *SIAM J. Numer. Anal.* **7** (1970) 390–398.
35. R. Fletcher, Conjugate gradient methods for indefinite systems, *Numerical Analysis*, ed. G. A. Watson, *Lecture Notes in Mathematics*, Vol. **506** (Springer, Berlin, Heidelberg, 2006), pp. 73–89.
36. R. W. Freund and N. M. Nachtigal, QMR: A quasi-minimal residual method for non-hermitian linear systems, *Numer. Math.* **60** (1991) 315–339.
37. S. F. McCormick, *Multigrid Methods* (SIAM, 1987).
38. L. F. Pavarino, Additive Schwarz methods for the p-version finite element method, *Numer. Math.* **66** (1993) 493–515.
39. P. Grigoraş, P. Burovskiy, W. Luk and S. Sherwin, Optimising sparse matrix vector multiplication for large scale FEM problems on FPGA, *2016 26th Int. Conf. Field Programmable Logic and Applications (FPL)* (IEEE, 2016), pp. 1–9.
40. J. Williams, C. Sarofeen, H. Shan and M. Conley, An accelerated iterative linear solver with GPUS for CFD calculations of unstructured grids, *Procedia Comput. Sci.* **80** (2016) 1291–1300.
41. S. Mazumder, *Numerical Methods for Partial Differential Equations: Finite Difference and Finite Volume Methods* (Academic Press, 2015).
42. F. Hecht, New development in freefem++, *J. Numer. Math.* **20** (2012) 251–265.
43. H. Jasak *et al.*, OpenFOAM: A c++ library for complex physics simulations, **1000** (2007) 1–20.
44. D. J. Limebeer, The application of generalized diagonal dominance to linear system stability theory, *Int. J. Control* **36** (1982) 185–212.
45. M. Abadi *et al.*, TensorFlow: A system for large-scale machine learning, *Proc. 12th USENIX Conf. on Operating Systems Design and Implementation* (USENIX, USA, 2016), pp. 265–283.

46. T. Chen and C. Guestrin, XGBoost: A scalable tree boosting system, *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining* (ACM, New York, 2016), pp. 785–794.
47. Y. Zabegaev, E. Keilegavlen, E. Iversen and I. Berre, Automated linear solver selection for simulation of multiphysics processes in porous media, *Comput. Methods Appl. Mech. Eng.* **426** (2024) 117031.