SymMaP: Improving Computational Efficiency in Linear Solvers through Symbolic Preconditioning

Anonymous Authors¹

Abstract

Matrix preconditioning is a critical technique to accelerate the solving of linear systems, where performance heavily depends on the selection of preconditioning parameters. Traditional parameter selection approaches often define fixed constants for specific scenarios. However, they rely on domain expertise and fail to consider the instance-wise features for individual problems, limiting their performance. In contrast, machine learning (ML) approaches, though promising, are hindered by high inference costs and limited interpretability. To combine the strengths of both approaches, we propose a symbolic discovery framework-namely, Symbolic Matrix Preconditioning (SymMaP)-to learn efficient symbolic expressions for preconditioning parameters. Specifically, we employ a large neural network to search the high-dimensional discrete space for expressions that can accurately predict the optimal parameters. The learned expression allows for high inference efficiency and excellent interpretability (expressed in concise 034 symbolic formulas), making it simple and reli-035 able for deployment. Experimental results show that SymMaP consistently outperforms traditional strategies across various benchmarks. 038

1. Introduction

039

041

Linear systems are foundational in the machine learning, physics, engineering, and other scientific fields (Leon et al., 043 2006; LeVeque, 2007). Since analytical solutions are often unavailable, efficient numerical algorithms become es-045 sential (Demmel, 1997). Matrix preconditioning, a key 046 technique in this domain, accelerates iterative solvers and 047 improves computational stability (Trefethen & Bau, 2022; Chen, 2005). For instance, the successive over-relaxation 049 (SOR) method optimizes convergence by integrating Gauss-050 Seidel iterations with a weighted update scheme governed 051 by the over-relaxation factor ω (Golub & Van Loan, 2013). 052

053 The effectiveness of matrix preconditioning depends on 054

key parameters, such as ω in SOR. Selecting $\omega > 1$ can accelerate convergence, while $\omega < 1$ may stabilize the process. This trade-off makes ω a critical parameter, directly influencing the performance of preconditioning.

Traditional parameter selection strategies often rely on domain expertise to define fixed constants for specific scenarios. However, (challenge 1) different problem parameters often require distinct optimal preconditioning parameters. Traditional strategies ignore instance-wise features-specific characteristics of individual problems, such as equation coefficients. This limits their adaptability to varying problem instances and tasks.

In contrast, machine learning (ML) approaches hold great promise but come with other challenges. First, (challenge 2) ML inference, while efficient on GPUs, performs poorly in CPU-only environments due to limited parallel processing capabilities. This is particularly problematic in linear system solver deployments, where GPU resources are often unavailable. Second, (challenge 3) the "black-box" nature of many ML techniques hinders a deeper understanding of the learned policies, raising concerns about their reliability.

In light of this, a natural solution is to combine the reliability and superior performance of these two paradigms. We propose a symbolic discovery framework-namely, Symbolic Matrix Preconditioning (SymMaP)-to learn efficient symbolic expressions for preconditioning parameters. The framework consists of three main steps. SymMaP first begins by applying grid search to identify the optimal preconditioning parameters based on task-specific performance metrics. Next, the framework performs a risk-seeking search in the high-dimensional discrete space of symbolic expressions, evaluating the best-found symbolic expression using a risk-seeking strategy. Finally, these symbolic expressions can be directly integrated into the modern solvers for linear systems, significantly improving computational efficiency.

The key contributions and advantages of SymMaP are summarized as follows:

• We propose a symbolic discovery framework, SymMaP, to learn efficient symbolic expressions for preconditioning parameters.

• SymMaP exhibits excellent generalization, making it adaptable to a wide range of preconditioning methods and optimization objectives.

 The symbolic expressions derived by SymMaP are both interpretable and easy to integrate into solver environments, offering a practical and transparent approach to enhancing the performance of linear system solvers.

2. Preliminaries

2.1. Matrix Preconditioning Technique

Matrix preconditioning is a technique employed to accelerate the convergence of iterative solvers and enhance the stability of algorithms. It is generally employed in solving linear systems (Chen, 2005; Golub & Van Loan, 2013), which are typically expressed in the form:

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

The fundamental idea of preconditioning is to transform the original problem into an equivalent one with better numerical properties. Specifically, this technique involves finding a preconditioner M that approximates either the inverse of A or some form conducive to iterative solutions (Chen, 2005). Consequently, the original (1) is transformed into

$$MAx = Mb. \tag{2}$$

There are generally two optimization objectives: 1. to accelerate the convergence of iterations by altering the spectral distribution of the matrix *A*. 2. to reduce the condition number of the matrix *A*, thereby lessening its ill-conditioning and enhancing the stability of iteration. Some common preconditioning techniques include the Jacobi, Gauss-Seidel, SOR (Young, 1954), algebraic multigrid (AMG) (Trottenberg et al., 2000), etc.

2.2. Prefix Notation

6 Prefix notation is a mathematical format where every operator precedes its operands, eliminating the need for parentheses required in conventional infix notation and simplifying 9 symbolic manipulation. This representation is particularly 1 advantageous in symbolic regression, as it allows mathemat-1 ical expressions to be expressed as sequences of tokens that 2 can be easily processed by neural networks.

In this notation, operators can be unary (e.g., sin, cos) or binary (e.g., $+, -, \times, \div$), while operands can be constants or variables (Landajuela et al., 2021). Each prefix expression uniquely corresponds to a symbolic tree structure, facilitating the conversion back to the original mathematical expression (Lample & Charton, 2019).



Figure 1: Variation in iteration counts and computation times under different SOR preconditioning parameters applied to a linear system from a second-order elliptic PDE.

The sequential nature of prefix notation aligns well with the architecture of recurrent neural networks (RNNs), which process information step by step. Unlike infix notation which may require variable-length look-ahead to determine the next valid token, prefix notation allows RNN to generate expressions through an auto-regressive process where each decision is well-defined based on previous tokens, and by removing the need for parentheses, it reduces the vocabulary size of possible tokens, which greatly enhances model training efficiency.

3. Motivation

The selection of matrix preconditioning parameters significantly affects their effectiveness (Chen, 2005). To design appropriate algorithmic prediction parameters, we first analyze the optimization space for preconditioning parameter selection and investigate the existence of optimal parameters. Next, we analyze the unique challenges present in this scenario. Finally, to address these challenges, we design a symbolic discovery framework to select the parameters.

3.1. Motivation for Optimizing Preconditioning Parameters

As illustrated in figure 1, the choice of relaxation factors ω significantly impacts the iteration count and computation time, when solving a second-order elliptic partial differential equation (PDE) (Evans, 2022) using SOR preconditioning (Golub & Van Loan, 2013). There exists an optimal parameter ω_{op} that minimizes the computation time, with specific details available in the Appendix B.2.

To further analyze the optimization space of preconditioning parameters, we evaluate the impact of various parameter selection strategies on preconditioning performance.



Figure 2: Ratio of average computation times at various tolerances to default parameter times under different SOR parameter selection schemes, evaluated on the second-order elliptic PDE dataset.

127

128

129

130

131 As shown in Figure 2, the 'Optimal Parameter ω_{op} ' rep-132 resents the parameter that minimizes computation time in 133 each experiment, serving as the theoretical upper limit of 134 our optimization. The 'Optimal Fixed Constant' refers to 135 a fixed constant that minimizes average computation time, 136 and 'Default Parameter' corresponds to the default setting 137 of $\omega = 1$ in the portable extensible toolkit for scientific com-138 putation (PETSc) (Balay et al., 2024). The gap between the 139 optimal fixed constant and the optimal parameter highlights 140 significant potential for optimizing preconditioning param-141 eter selection, motivating this paper. The performance of 142 our SymMaP algorithm approaches the optimal parameter, 143 demonstrating its accuracy in learning the optimal parameter 144 expression. 145

146 147 148 **3.2. Challenges in Predicting Efficient Preconditioning** Parameters

We aim to develop a universal framework for predictingefficient parameters. However, the context of solving linearsystems imposes specific challenges:

152 (C1) Strong Generalization Capability: Real-world sci-153 entific computing scenarios vary significantly. For instance, 154 the choice of PDE grid form can lead to significant variations 155 in matrix structure (Johnson, 2009), resulting in distinct 156 optimal parameters. Moreover, preconditioning addresses 157 various optimization goals, such as reducing computational 158 time, the number of iterations, and lowering condition num-159 bers (Chen, 2005). This necessitates that parameter predic-160 tion algorithms possess robust generalization capabilities: 161 they should take problem scenarios and features as inputs 162 while applying them to different preconditioning methods 163 and optimization goals. 164

(C2) Computational Efficiency: linear system solver typically relies on Krylov subspace methods implemented in low-level libraries optimized for CPU architectures, such as PETSc (Balay et al., 2024), LAPACK (Anderson et al., 1999). Algorithms like generalized minimal residual method (GMRES) (Saad & Schultz, 1986) and conjugate gradient (CG) (Greenbaum, 1997) iteratively compute the matrix's invariant subspace, favoring single-threaded or limited multithreaded execution modes. Preconditioning techniques aim to accelerate these solvers without significant additional computational overhead, often adopting implicit iterative formats (e.g., SOR (Chen, 2005)) or utilizing low-cost matrix decompositions (e.g., AMG (Trottenberg et al., 2000)). Therefore, any parameter prediction algorithms must be compatible with CPU environments and seamlessly integrated into existing algorithm libraries. At the same time, it must maintain low computational costs to preserve the performance benefits of preconditioning.

(C3) Algorithmic Transparency: Algorithms in scientific computing often require rigorous analysis under mathematical theories. Opaque prediction algorithms could confuse researchers. For instance, the relaxation factor ω in SOR needs to avoid being too close to 0 or 2 in some scenarios (Agarwal, 2000). This is an issue that opaque algorithms cannot avoid in advance. Moreover, interpretable algorithms can guide researchers to conduct further studies and reveal the underlying mathematical structures of problems. Therefore, these pose challenges to the transparency and interpretability of the parameter prediction algorithms.

3.3. Symbolic Discovery to Preconditioning Parameter Selection

Symbolic discovery extracts mathematical expressions from data, establishing relationships between problem features and optimal preconditioning parameters. Its integration into matrix preconditioning overcomes parameter selection challenges through a generalizable, efficient, and transparent approach.

Firstly, symbolic discovery can accommodate various types of input parameters and can specifically tailor symbolic expression learning for different preconditioning methods and optimization goals (Cranmer et al., 2020), thereby meeting the requirement for broad applicability in scientific computing tasks (C1).

Secondly, the explicit expressions derived are computationally lightweight and can be quickly evaluated at runtime. They integrate seamlessly into existing CPU-based algorithm libraries like PETSc (Balay et al., 2024) with almost no overhead (**C2**).

Thirdly, the symbolic discovery provides transparent and interpretable expressions (Rudin, 2019), allowing researchers



Figure 3: Illustration of how SymMaP discovers efficient symbolic expressions for preconditioning parameters. Part 1 demonstrates the acquisition of optimal parameters and dataset generation; Part 2 illustrates the training process of the RL-based deep symbolic discovery framework; Part 3 shows how the sequential model generates symbolic policies; Part 4 presents the deployment of symbolic expressions.

to understand the influence of parameters within existing theoretical frameworks and identify potential numerical stability issues. This interpretability fosters trust in the algorithm's predictions and supports further theoretical exploration (C3).

4. Method

185

186

187

188 189 190

191

193

195

196

197

This study focuses on enhancing the performance of parameterized preconditioners in solving linear systems derived from parameterized PDEs. Specifically, we investigate preconditioners with continuous parameters, such as the relaxation factor in SOR, while excluding those with discrete parameters like the level of fill-in ICC or ILU factorization.

205 We introduce a novel framework, SymMaP, for symbolic dis-206 covery in matrix preconditioning. As shown in Figure 3, we first obtain the optimal preconditioning parameters through 208 a grid search to construct a training dataset. Then we em-209 ploy an RNN to generate symbolic expressions in prefix 210 notation, which are then evaluated for their fitness. The 211 RNN is trained using a reward function based on the performance of the generated expressions. By optimizing the 212 RNN parameters to maximize this reward function, we gen-213 214 erate symbolic expressions that approximate the relationship 215 between the problem's feature parameters (PDE parameters) and the optimal preconditioning parameters. Finally, we 216 217 deploy the learned symbolic expressions into linear system 218 solvers. The detailed steps are as follows and pseudocode is 219

provided in the Appendix C.

4.1. Input Features and Training Data Generation

Input Features. In the context of solving parameterized PDEs, which frequently arise in linear systems, we consider feature parameters that characterize the equations. For instance, a second-order elliptic PDE can be expressed as: $a_{11}u_{xx} + a_{12}u_{xy} + a_{22}u_{yy} + a_1u_x + a_2u_y + a_0u = f$, where the coefficients $a_{11}, a_{12}, a_{22}, \ldots$ represent the feature parameters of PDE (see Appendix D.1 for details). These feature parameters, denoted as x_i , serve as input features for the symbolic discovery process in SymMaP.

Training Data Generation. For each linear system, we determine optimal preconditioning parameters through an adaptive grid search. Using SOR preconditioning as an example, we optimize the relaxation factor ω within [0, 2] to minimize computation time (or condition number). The search process involves: 1. Conducting an initial coarse grid search (step size: 0.01) to evaluate computation time (or condition number) for each ω . 2. Identifying candidate regions with optimal performance. 3. Performing a refined grid search (step size: 0.001) within these regions.

This process yields the required training dataset, where each data point contains: 1. the problem feature parameters x_i ; 2. the optimal preconditioning parameters y_i . i = 1, 2, ..., n, and n is the number of data, typically set to n = 1200, with 1000 allocated for the training set and 200 for the test set.

4.2. The Generation of Symbolic Expressions

Token Library. For SymMaP, we define the library \mathcal{L} of mathematical operators and operands as $\{+, -, \times, \div, \text{sqrt}, \exp, \log, \text{pow}, 1.0\}$. Although other operators such as poly, sin and cos are frequently used (Udrescu & Tegmark, 2020), we decided to exclude them because they offer limited explanatory power in matrix preconditioning and significantly increase the time and memory consumption during training.

After converting the mathematical expressions into prefix notation, we use this tokenized representation as a preorder traversal of the expression tree (Zaremba & Sutskever, 2014). In each iteration, the RNN receives a pair consisting of a parent node and a sibling node as inputs. Then the RNN outputs a categorical distribution over all possible next tokens. The parent node refers to the last incomplete operator that requires additional operands to form a complete expression. The sibling node, in the context of a binary operator, represents the operand that has already been processed and incorporated into the expression. In cases where no parent or sibling node is applicable, they are designated as empty nodes. This structured input method enables the RNN to maintain contextual awareness and effectively predict the sequence of tokens that form valid mathematical expressions.

The Sequential Model. During the generation of a single symbolic expression, the RNN emits a categorical distribution for each "next token" at each step. This distribution is represented as a vector $\psi_{\theta}^{(i)}$, where *i* denotes the *i*-th step and θ represents the parameters of the RNN. The elements of the vector correspond to the probabilities of each token, conditioned on the previously selected tokens in the traversal (Petersen et al., 2019):

$$\psi_{\boldsymbol{\theta}}^{(i)}(\boldsymbol{\tau}_i) = \boldsymbol{p}(\boldsymbol{\tau}_i | \boldsymbol{\tau}_{1:i-1}; \boldsymbol{\theta}).$$
(3)

Here, τ_i denotes the index of the token selected at the *i*-th step. The probability of generating the entire symbolic expression τ is then the product of the conditional probabilities of all tokens (Petersen et al., 2019; Landajuela et al., 2021):

$$\boldsymbol{p}(\boldsymbol{\tau}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \psi_{\boldsymbol{\theta}}^{(i)}(\boldsymbol{\tau}_i). \tag{4}$$

Optimization of Constants: The library \mathcal{L} incorporates a 'constant token,' which allows for the inclusion of various constant placeholders within sampled expressions. These placeholders serve as the parameters ξ in the symbolic expression. We seek to find the optimal values

of these parameters by maximizing the reward function: $\xi^* = \arg \max_{\xi} R(\tau; \xi)$, utilizing a nonlinear optimization method. This optimization is executed within each sampled expression as an integral part of computing the reward, prior to each training iteration.

4.3. The Reward Function

Once a symbolic expression is fully generated (i.e., the symbolic tree reaches all its leaf nodes), we evaluate its fitness by calculating the normalized root-mean-square error (NRMSE), a metric frequently used in genetic programming symbolic discovery (Schmidt & Lipson, 2009). It is defined as

NRMSE =
$$\frac{1}{\sigma_y} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2},$$
 (5)

where $\hat{y}_i = \boldsymbol{\tau}(x_i)$ is the predicted value for the *i*-th sample, x_i is the problem feature parameter, y_i is the optimal preconditioning parameter, σ_y is the standard deviation of the target values y, and n is the number of data. To bound this fitness value between 0 and 1, we apply a squashing function:

$$R(\boldsymbol{\tau}) = \frac{1}{1 + \text{NRMSE}}.$$
(6)

Our objective is to maximize $R(\tau)$, thereby minimizing the NRMSE and improving the accuracy of the generated expressions.

4.4. The Training Algorithm

Although the objective function is well-defined, it is important to note that $R(\tau)$ is not a deterministic value but a random variable dependent on the RNN's parameters θ . Therefore, the key challenge is to establish an appropriate criterion for evaluating this random variable, and then apply gradient-based optimization methods accordingly.

Risk-seeking Policy. It is natural to consider the expectation of the reward function, i.e., $\mathbb{E}_{\tau \sim p(\tau;\theta)}[R(\tau)]$, as the objective function to optimize. We can apply the 'log-integral' trick (Williams, 1992) and obtain

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{p}(\boldsymbol{\tau}; \boldsymbol{\theta})}[R(\boldsymbol{\tau})] = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{p}(\boldsymbol{\tau}; \boldsymbol{\theta})}[R(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{p}(\boldsymbol{\tau}; \boldsymbol{\theta})].$$
(7)

Thus, even though the expectation of the reward function is not directly differentiable with respect to θ , we can approximate the gradient using the sample mean.

Table 1: Comparison of average computation times (seconds) for SOR with different ω selections, and tolerance is 1e-7. SymMaP 1 and 2 are the two learned expressions that achieved the highest reward function scores, with the best-performing method highlighted in bold.

Dataset	Matrix size	No precondition	PETSc default $\omega = 1$	Fixed constant $\omega = 0.2$	Fixed constant $\omega = 1.8$	Optimal constant	SymMaP 1	SymMaP 2
Biharmonic	4.2×10^3	7.67	2.04	4.86	1.60	1.31	1.24	1.26
Darcy Flow	1.0×10^4	33.1	13.5	17.5	9.91	9.54	8.50	8.60
Elliptic PDE	4.0×10^4	31.3	21.0	21.4	17.5	16.6	15.8	16.3
Poisson	2.3×10^3	4.12×10^{-2}	1.95×10^{-2}	2.15×10^{-2}	1.95×10^{-2}	1.38×10^{-2}	1.35×10^{-2}	$1.36{\times}10^{-2}$
Thermal	2.8×10^3	2.23×10^{-1}	5.98×10^{-2}	2.07×10^{-1}	$1.18{\times}10^{-1}$	5.94×10^{-2}	5.76×10^{-2}	$5.91{\times}10^{-2}$

In the context of symbolic regression, model performance is often driven by a few exceptional results that outperform others by a significant margin (Petersen et al., 2019; Tamar et al., 2015). With this in mind, we adopt a risk-seeking policy, which aims to maximize:

$$J(\boldsymbol{\theta},\varepsilon) = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{p}(\boldsymbol{\tau};\boldsymbol{\theta})}[R(\boldsymbol{\tau})|R(\boldsymbol{\tau}) > Q(\boldsymbol{\theta},\varepsilon)]. \quad (8)$$

Here, ε is the risk factor, typically $\varepsilon = 0.05$, $Q(\theta, \varepsilon)$ is the $(1 - \varepsilon)$ -quantile of the reward distribution under parameter θ , i.e.

$$Q(\boldsymbol{\theta}, \varepsilon) = \inf\{q \in \mathbb{R} | \text{CDF}(R(\boldsymbol{\tau}); \boldsymbol{\theta}) \ge 1 - \varepsilon\}, \quad (9)$$

where $\text{CDF}(R(\tau); \theta)$ refers to the cumulative distribution function. From this, the gradient of $J(\theta, \varepsilon)$ can be derived as (Petersen et al., 2019):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \varepsilon) = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{p}(\boldsymbol{\tau}; \boldsymbol{\theta})} \left[\left(R(\boldsymbol{\tau}) - Q(\boldsymbol{\theta}, \varepsilon) \right) \\ \cdot \nabla_{\boldsymbol{\theta}} \log \boldsymbol{p}(\boldsymbol{\tau}; \boldsymbol{\theta}) \middle| R(\boldsymbol{\tau}) > Q(\boldsymbol{\theta}, \varepsilon) \right].$$
(10)

This gradient can be estimated using Monte Carlo sampling:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \varepsilon) \approx \hat{g} \triangleq \tag{11}$$

$$\frac{1}{\varepsilon N} \sum_{i=1}^{N} (R(\boldsymbol{\tau}^{(i)}) - \tilde{Q}(\boldsymbol{\theta}, \varepsilon)) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{p} \cdot \mathbb{1}_{R(\boldsymbol{\tau}^{(i)}) > \tilde{Q}(\boldsymbol{\theta}, \varepsilon)},$$

 $\tilde{Q}(\theta, \varepsilon)$ is the empirical $(1 - \varepsilon)$ -quantile of the reward function. By concentrating on the top ε percentile of samples, SymMaP emphasizes optimizing the best-performing solutions in preconditioning, thereby obtaining the optimal symbolic expressions for preconditioning parameters.

4.5. Deployment in Linear System Solver

After the training process, we obtained a symbolic expression for predicting the preconditioning parameter. The learned formula is exceptionally concise and incurs minimal computational cost. Therefore, we directly compile the learned policy into a lightweight shared object using a simple script and then integrate it into the linear system solver package (e.g., PETSc).

5. Experiments

We conducted comprehensive experiments to evaluate the SymMaP framework, organized into three primary sections: 1. Assessment of three different preconditioners and optimization goals across various datasets to determine the effectiveness of SymMaP algorithm, 2. Analysis of associated computational cost and the interpretability of the learned symbolic expressions, 3. Ablation studies of SymMaP.

Preconditioners: We considered three different preconditioners and various optimization metrics: 1. SOR preconditioner with the relaxation factor ω (Golub & Van Loan, 2013); 2. SSOR preconditioner with the relaxation factor ω (Golub & Van Loan, 2013); 3. AMG preconditioner with the threshold parameters θ_T (Trottenberg et al., 2000).

Datasets: We investigated linear systems derived from five distinct PDE classes: 1. Darcy Flow Problems (Li et al., 2020), 2. Second-order Elliptic PDEs (Evans, 2022), 3. Biharmonic Equations (Barrata et al., 2023), 4. Thermal Problems (Wang et al., 2024). 5. Poisson Equations (Wang et al., 2024). All cases except biharmonic equations yield symmetric matrices. Notably, the non-symmetric matrices from biharmonic equations are incompatible with SSOR and AMG preconditioning techniques.

Baselines: We compared SymMaP against various parameter selection methods for preconditioning. Specifically, the comparison involved the following scenarios: 1. No matrix preconditioning, 2. Default parameters in PETSc (Balay et al., 2024), 3. Fixed constants, 4. Optimized fixed constants.

Experiment Settings: All preconditioning procedures

Table 2: Comparison of average computation times (seconds) for SSOR with different ω selections, and tolerance is 1e-7. SymMaP 1 and 2 are the two learned expressions that achieved the highest reward function scores, with the best-performing method highlighted in bold.

+ 5 5	Dataset	Matrix size	No precondition	PETSc default $\omega = 1$	Fixed constant $\omega = 0.2$	Fixed constant $\omega = 1.8$	Optimal constant	SymMaP 1	SymMaP 2
	Darcy Flow	$4.9 imes 10^3$	4.18	0.488	0.757	1.09	0.448	0.412	0.523
	Elliptic PDE	4.0×10^4	23.9	10.5	14.7	8.72	8.68	7.70	7.74
	Poisson	2.3×10^3	$2.12{\times}10^{-2}$	1.02×10^{-2}	1.93×10^{-2}	9.91×10^{-3}	9.89×10^{-3}	9.10×10^{-3}	9.92×10^{-3}
	Thermal	2.8×10^3	2.34×10^{-1}	2.69×10^{-2}	5.08×10^{-2}	9.87×10^{-2}	2.24×10^{-2}	2.13×10^{-2}	2.14×10^{-2}

Table 3: Comparison of average condition numbers for preconditioned matrices using different threshold parameter θ_T selections in AMG. SymMaP 1 and 2 are the two learned expressions that achieved the highest reward function scores, with the best-performing method highlighted in bold.

Dataset	Matrix size	No precondition	PETSc default $\theta_T = 0$	Fixed constant $\theta_T = 0.2$	Fixed constant $\theta_T = 0.8$	Optimal constant	SymMaP 1	SymMaP 2
Darcy Flow	1.0×10^4	752862	8204	19146	11426	7184	4824	5786
Elliptic PDE	4.0×10^4	6792	184.6	205.4	212.5	182.8	168.8	170.3
Poisson	1.0×10^4	1242	4.55	68.85	68.85	4.55	3.72	3.72
Thermal	2.8×10^3	7325	11.9	627.2	627.2	9.91	9.71	9.71

were uniformly implemented using the C-based PETSc library (Balay et al., 2024) to maintain evaluation consistency.
The experiments were conducted within PETSc's GMRES
linear solver framework (Golub & Van Loan, 2013), with
condition numbers computed through the built-in function
KSPComputeExtremeSingularValues.

Details on preconditioners, the mathematical forms of datasets, and the runtime environment are available in Appendices B, D.1, and D.2, respectively. Information on the generation of training datasets for the following experiments and parameters of the SymMaP algorithm are outlined in Appendices D.3 and D.4. The generated dataset and training time are available in Appendix D.5. For an introduction to related work, see Appendix A.

5.1. Main Experiments

333

342 343

344

345

346

355 356

372

373

374 In these experiments, as shown in Tables 1, 2, 3, we opti-375 mized relaxation factors ω in both SOR and SSOR precon-376 ditioning, and threshold parameters θ_T in AMG precondi-377 tioning. For SOR and SSOR, we identified ω values that 378 minimize computation time, forming the training dataset 379 for SymMaP to learn symbolic expressions that optimize 380 computational times for solutions. Similarly, for AMG, we 381 selected θ_T values that minimize the condition number of 382 preconditioned matrices. Partial symbolic expressions can 383 be found in Appendix E.1. 384

Experimental results indicate that SymMaP consistently outperforms all others across all experimental tasks. For SOR, Table 1 shows that SymMaP reduces computation times by up to 40% compared to PETSc's default settings and by 10% against the optimal constants. In SSOR, Table 2 shows that it cuts computation time and iteration counts by up to 27%, over PETSc's defaults, and achieves reductions of 11% in time compared to optimal constants. For AMG, Table 3 shows that SymMaP lowers the condition number by up to 40% relative to PETSc's defaults and 32% against the optimal constants.

These results highlight SymMaP's ability to effectively derive high-performance symbolic expressions for various preconditioning parameters, showcasing its broad applicability and strong generalization across different preconditioning tasks.

5.2. Comparison with Neural Network Performance

To evaluate the deployment overhead and prediction performance of SymMaP, we compared it with a basic multilayer perceptron (MLP) architecture. The MLP implementation consists of three fully connected layers, taking PDE parameters as input and generating preconditioning parameters as output. We employed ReLU activation functions and trained the model using mean squared error (MSE) between predicted and optimal parameters as the loss function. Both Table 4: Comparison of the runtime required for symbolic expression and MLP to predict the SOR relaxation factor and the subsequent average solution time for linear systems, using the Darcy flow dataset with a matrix size of 10^3 and tolerance is 1e-5.

390

395

398

399

400

409

421

422

	Runtime (s)	Solution time (s)
MLP	5.1e-5	7.1e-1
Symbol	1.1e-5	7.1e-1

396 symbolic expression and MLP were executed in a CPU397 environment to simulate a modern solver environment.

Table 5: Partial symbolic expressions learned from some of the main experiments

Precondition	Dataset	Symbolic expression
SOR	Biharmonic	$1.0 + 1.0/(4.0 + 1.0/x_2) +$
SOR	Elliptic PDE	$1.0 + 1.0/(x_2 + 1.0 + 1.0/(x_2$
SOR	Darcy Flow	$1.0 + 1.0/(x_4 + 1.0)$
SSOR	Elliptic PDE	$1.0 + 1.0/(x_2 + 1.2)$
AMG	Elliptic PDE	$(x_1x_3+1)/7$

410 As shown in Table 4, the runtime of symbolic expressions 411 learned by SymMaP was only 20% of that of the MLP, pri-412 marily due to the poor performance of neural networks in a 413 pure CPU environment, highlighting SymMaP's computa-414 tional efficiency. Furthermore, the average solution times 415 for parameters predicted by both symbolic expressions and 416 MLP were closely matched. This demonstrates that sym-417 bolic expressions possess equivalent expressive capabilities 418 to neural networks in this scenario, effectively approximat-419 ing the optimal parameter expressions. 420

5.3. Interpretable analysis

In Table 5, we report a subset of the learned symbolic expressions, with the mathematical significance of the related
symbols detailed in Appendix E.2. More symbolic expressions can be found in Appendix E.1. These symbolic expressions are notably more concise and selective, not utilizing all
candidate parameters and symbols, which aids researchers
in analyzing their underlying relationships.

430 For instance, in the context of SOR and SSOR precondi-431 tioning, empirical evidence suggests that smaller relaxation 432 factors should be chosen when diagonal components are 433 relatively small. Our experimental findings corroborate this: 434 for the second-order elliptical PDE dataset, the symbolic 435 expressions derived for SOR and SSOR preconditioning de-436 pend solely on x_2 , with larger x_2 values leading to smaller 437 predicted relaxation factors, exemplified by $1.0 + \frac{1.0}{(x_2+1.2)}$ 438 Here, x_2 represents the coupling coefficient of the elliptical 439

Table 6: Ablation study examining the selection of mathematical operators, comparing the effects on preconditioning and training times. The first column lists the selected operators, the second column shows the condition numbers of preconditioned matrices derived from AMG parameter predictions on the Darcy flow dataset (lower is better), and the third column displays SymMaP training times.

Functionset	Condition number	time(s)
$+, -, \times, \div, poly$	6803.8	15351
$+,-, imes, \div, ext{sqrt}, ext{exp}, ext{log}, ext{pow}, 1.0$	7086.9	703.17
$+,-, imes, \div, ext{sqrt}, ext{exp}, ext{log}, ext{sin}, ext{cos}, ext{pow}, 1.0$	7172.6	635.82
$+,-,\div,1.0,$ pow	7241.8	703.26
$+,-, imes, \div, ext{sqrt}, ext{pow}, 1.0$	7271.1	746.80
$+,-, imes, \div, \mathrm{pow}, 1.0$	7301.4	702.46

PDE, which directly influences the relative size of the nondiagonal components of the generated matrix, whereas other coefficients have minimal impact. As the coupling coefficient increases, the relative numerical of the non-diagonal components increases, and the diagonal components reduce correspondingly, aligning with empirical observations.

These experimental outcomes demonstrate that SymMaP can derive interpretable and efficient symbolic expressions for parameters, further aiding researchers in understanding and exploring the underlying mathematical principles.

5.4. Ablation Experiments

We conducted an ablation study using SymMaP to evaluate the impact of different mathematical operator selections, as described in Table 6. In the main experiments, We utilized the operator set $\{+, -, \times, \div, \text{sqrt}, \exp, \log, pow, 1.0\}$ listed in the second row.

The results indicate that this selection of operators achieves a balance between predictive performance and training time efficiency, meeting our expectations. Furthermore, experiments detailing the performance of SymMaP about variations in learning rate, batch size, and dataset size are documented in Appendix E.3.

6. Conclusions

In this paper, we propose SymMaP, a deep symbolic discovery framework designed for predicting efficient matrix preconditioning parameters. Experiments show that SymMaP can predict high-performance parameters and is applicable across a variety of preconditioning and optimization objectives. Additionally, SymMaP is easy to deploy with virtually no computational cost. We are confident in the symbolic model's immense potential for broad real-world applications, especially in matrix preconditioning.

Impact Statement 440

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

474

475

476

477

478

479

480

481

482

483

- Agarwal, R. P. Difference equations and inequalities: theory, methods, and applications. CRC Press, 2000.
- Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. LAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).
- 458 Balay, S., Abhyankar, S., Adams, M. F., Benson, S., Brown, 459 J., Brune, P., Buschelman, K., Constantinescu, E. M., 460 Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., 461 Gropp, W. D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, 462 D., Kaushik, D., Knepley, M. G., Kong, F., Kruger, S., 463 May, D. A., McInnes, L. C., Mills, R. T., Mitchell, L., 464 Munson, T., Roman, J. E., Rupp, K., Sanan, P., Sarich, 465 J., Smith, B. F., Zampini, S., Zhang, H., Zhang, H., and 466 Zhang, J. PETSc Web page. https://petsc.org/, 467 2024. URL https://petsc.org/. 468
- 469 Barrata, I. A., Dean, J. P., Dokken, J. S., Habera, M., HALE, 470 J., Richardson, C., Rognes, M. E., Scroggs, M. W., Sime, 471 N., and Wells, G. N. Dolfinx: The next generation fenics 472 problem solving environment. 2023. 473
 - Bers, L., John, F., and Schechter, M. Partial differential equations. American Mathematical Soc., 1964.
 - Chen, K. Matrix preconditioning techniques and applications. Number 19. Cambridge University Press, 2005.
 - Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., Lu, Y., et al. Symbolic discovery of optimization algorithms. Advances in neural information processing systems, 36, 2024.
- 484 Ciarlet, P. G. and Raviart, P.-A. A mixed finite element 485 method for the biharmonic equation. In Mathematical 486 aspects of finite elements in partial differential equations, 487 pp. 125-145. Elsevier, 1974. 488
- 489 Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., 490 Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. 492 Advances in neural information processing systems, 33: 493 17429-17442, 2020. 494

- Demmel, J. W. Applied numerical linear algebra. SIAM, 1997.
- Driscoll, T. A., Hale, N., and Trefethen, L. N. Chebfun guide, 2014.
- Evans, L. C. Partial differential equations, volume 19. American Mathematical Society, 2022.
- Glowinski, R. and Pironneau, O. Numerical methods for the first biharmonic equation and for the two-dimensional stokes problem. SIAM review, 21(2):167-212, 1979.
- Golub, G. H. and Van Loan, C. F. Matrix computations. JHU press, 2013.
- Götz, M. and Anzt, H. Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation. In 2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA), pp. 49–56. IEEE, 2018.
- Greenbaum, A. Iterative methods for solving linear systems. SIAM, 1997.
- Greenfeld, D., Galun, M., Basri, R., Yavneh, I., and Kimmel, R. Learning to optimize multigrid pde solvers. In International Conference on Machine Learning, pp. 2415–2423. PMLR, 2019.
- Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning neural pde solvers with convergence guarantees. arXiv preprint arXiv:1906.01200, 2019.
- Johnson, C. Numerical solution of partial differential equations by the finite element method. Courier Corporation, 2009.
- Koric, S. and Abueidda, D. W. Data-driven and physicsinformed deep learning operators for solution of heat conduction equation with parametric heat source. International Journal of Heat and Mass Transfer, 203:123809, 2023.
- Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. arXiv preprint arXiv:2108.08481, 2021.
- Lample, G. and Charton, F. Deep learning for symbolic mathematics. arXiv preprint arXiv:1912.01412, 2019.
- Landajuela, M., Petersen, B. K., Kim, S., Santiago, C. P., Glatt, R., Mundhenk, N., Pettit, J. F., and Faissol, D. Discovering symbolic policies with deep reinforcement learning. In International Conference on Machine Learning, pp. 5979-5989. PMLR, 2021.

- Leon, S. J., De Pillis, L. G., and De Pillis, L. G. *Linear algebra with applications*. Pearson Prentice Hall Upper
 Saddle River, NJ, 2006.
- LeVeque, R. J. Finite difference methods for ordinary and partial differential equations: steady-state and timedependent problems. SIAM, 2007.
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang,
 Z., and Karniadakis, G. E. A comprehensive and fair
 comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.
- Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I.
 Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pp. 6489–6499. PMLR, 2020.
- Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi,
 M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B.,
 Ahern, A., et al. Faster sorting algorithms discovered
 using deep reinforcement learning. *Nature*, 618(7964):
 257–263, 2023.
- Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.
 - Poli, R., Langdon, W., and McPhee, N. A field guide to genetic programming (with contributions by jr koza)(2008). *Published via http://lulu. com*, 2008.

529

530

531

540

541

- Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K.
 U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.
- Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206– 215, 2019.
 - Ruge, J. W. and Stüben, K. Algebraic multigrid. In *Multi*grid methods, pp. 73–130. SIAM, 1987.
- 543 Saad, Y. Iterative methods for sparse linear systems. SIAM, 2003.
- Saad, Y. and Schultz, M. H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

- Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.
- Sharma, R., Farimani, A. B., Gomes, J., Eastman, P., and Pande, V. Weakly-supervised deep learning of heat transport via physics informed loss. *arXiv preprint arXiv:1807.11374*, 2018.
- Stanaityte, R. ILU and Machine Learning Based Preconditioning for the Discretized Incompressible Navier-Stokes Equations. PhD thesis, University of Houston, 2020.
- Taghibakhshi, A., MacLachlan, S., Olson, L., and West, M. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in neural information* processing systems, 34:12129–12140, 2021.
- Tamar, A., Glassner, Y., and Mannor, S. Policy gradients beyond expectations: Conditional value-at-risk. Citeseer, 2015.
- Trefethen, L. N. and Bau, D. *Numerical linear algebra*. SIAM, 2022.
- Trottenberg, U., Oosterlee, C. W., and Schuller, A. *Multi-grid*. Elsevier, 2000.
- Udrescu, S.-M. and Tegmark, M. Ai feynman: A physicsinspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.
- Wang, H., Hao, Z., Wang, J., Geng, Z., Wang, Z., Li, B., and Wu, F. Accelerating data generation for neural operators via krylov subspace recycling. *arXiv preprint arXiv:2401.09516*, 2024.
- Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
- Young, D. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.
- Zaremba, W. and Sutskever, I. Learning to execute. *arXiv* preprint arXiv:1410.4615, 2014.
- Zhang, E., Kahana, A., Turkel, E., Ranade, R., Pathak, J., and Karniadakis, G. E. A hybrid iterative numerical transferable solver (hints) for pdes based on deep operator network and relaxation methods. *arXiv preprint arXiv:2208.13273*, 2022.

A. Related work

A.1. Machine Learning for Algorithm Discovery

Machine learning has the potential to uncover implicit rules beyond human intuition from training data, enabling the construction of algorithms that outperform handcrafted programs. Approaches to algorithm discovery in machine learning encompass symbolic discovery, program search, and more. Specifically, program search focuses on optimizing the computational processes of algorithms. For example, Mankowitz et al. (2023) explores the discovery of faster sorting algorithms, while Chen et al. (2024) investigates efficient optimization algorithms.

In contrast, symbolic discovery aims to search within the space of small mathematical expressions rather than computational streams (Petersen et al., 2019; Landajuela et al., 2021). This approach is analogous to an extreme form of model distillation, where knowledge extracted from black-box neural networks is distilled into explicit mathematical expressions. Traditional methods for symbolic discovery have relied on evolutionary algorithms, including genetic programming (Poli et al., 2008). Recently, deep learning has emerged as a powerful tool in this domain, offering enhanced representational capacity and new avenues for solving symbolic discovery problems (Schmidt & Lipson, 2009; Cranmer et al., 2020).

A.2. Neural Networks for Matrix Preconditioning

Recent studies have explored the use of neural networks to improve matrix preconditioning techniques. (Greenfeld et al., 2019; Luz et al., 2020; Taghibakhshi et al., 2021) demonstrate the effectiveness of neural networks in refining multigrid preconditioning algorithms, thus streamlining the computational process. (Götz & Anzt, 2018) utilized Convolutional Neural Networks (CNNs) for the optimization of block Jacobi preconditioning algorithms, while (Stanaityte, 2020) developed corresponding Incomplete Lower-Upper Decomposition (ILU) preconditioning algorithms leveraging machine learning insights. Although these algorithms achieved impressive results, they still face challenges such as limited interpretability and reduced computational efficiency when deployed in pure CPU environments. This paper attempts to address these issues by incorporating symbolic discovery into the framework.

B. Detailed introduction of matrix preconditioning

B.1. Overview of Matrix Preconditioning Methods

- **Jacobi Method**: The Jacobi preconditioner utilizes only the diagonal elements of a matrix to precondition a linear system. By approximating the inverse of the diagonal matrix, this method is computationally simple and effective for systems with strong diagonal dominance. However, its convergence rate can be slow, and its performance diminishes for poorly conditioned or weakly diagonally dominant matrices. The Jacobi method is typically used as a baseline for comparison with more sophisticated preconditioners (Saad, 2003).
- **Gauss-Seidel (GS) Method**: The Gauss-Seidel preconditioner improves upon the Jacobi method by considering both the lower triangular and diagonal parts of the matrix in a sequential manner. Unlike the Jacobi method, which updates all variables simultaneously, the GS method updates each variable in sequence using the most recent values. This leads to faster convergence, especially for diagonally dominant matrices. However, the GS method can still struggle with poorly conditioned systems, and its forward-only approach can limit performance in some applications (Saad, 2003).
- Successive Over-Relaxation (SOR): The SOR method builds on the Gauss-Seidel method by introducing a relaxation factor ω to accelerate convergence. This factor allows for over-relaxation ($\omega > 1$) or under-relaxation ($\omega < 1$), tuning the method for faster performance on certain types of problems. SOR can significantly reduce the number of iterations needed for convergence compared to both the Jacobi and GS methods, but choosing the optimal relaxation factor is problem-dependent (Young, 1954).
- Symmetric Successive Over-Relaxation (SSOR): SSOR is a symmetric version of the SOR method, where relaxation is applied in both forward and backward sweeps of the matrix. This bidirectional process improves stability and is well-suited for use with iterative solvers like the conjugate gradient method, which requires symmetric preconditioners. SSOR's symmetry ensures that the preconditioner maintains the properties needed for efficient and stable convergence, making it a popular choice for symmetric positive-definite systems (Golub & Van Loan, 2013).
- Algebraic Multigrid (AMG): AMG is an advanced preconditioning technique designed to handle large, sparse systems of linear equations, especially those arising from the discretization of partial differential equations. Unlike traditional

methods, AMG operates on multiple levels of the matrix structure, coarsening the matrix to form a hierarchy of smaller systems that are easier to solve. Solutions on the coarser grids are then interpolated back to the finer grids. This multilevel approach makes AMG highly efficient for large-scale problems, as it can dramatically reduce the number of iterations needed to achieve convergence. AMG is often used in combination with methods like SSOR or Gauss-Seidel as a smoother on each grid level, and it is particularly effective in cases where the problem exhibits a multiscale nature (Ruge & Stüben, 1987).

Relationship Among Jacobi, GS, and SOR Methods: The Jacobi method is the simplest of the three, using only diagonal information. The GS method improves upon the Jacobi method by using both diagonal and lower triangular matrix elements to achieve faster convergence. SOR further refines the GS method by introducing a relaxation factor to optimize the update process. Both the GS and SOR methods can be seen as iterative improvements on the Jacobi method, with SOR offering a more flexible and potentially faster alternative by adjusting the relaxation factor. SSOR extends SOR symmetrically, making it suitable for use in more advanced iterative solvers like the conjugate gradient method (Saad, 2003; Golub & Van Loan, 2013).

B.2. Parameters in Matrix Preconditioning

The choice of preconditioning parameters significantly influences the effectiveness of the preconditioning process, especially in the iterative solving of linear systems (Chen, 2005). Below, we discuss three specific preconditioning techniques—SOR, SSOR, and AMG—focusing particularly on how their key parameters affect the preconditioning results.

B.2.1. Relaxation Factor ω in SOR and SSOR Methods

In the SOR preconditioning method, the relaxation factor ω is a critical parameter that determines the acceleration of iteration. SOR evolves from the Gauss-Seidel method by introducing ω to speed up convergence. The SOR iteration formula is given by:

$$\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} + \omega \boldsymbol{L})^{-1} \left[(1-\omega) \boldsymbol{D} \boldsymbol{x}^{(k)} + \omega \boldsymbol{b} - \omega \boldsymbol{U} \boldsymbol{x}^{(k)} \right],$$
(12)

where D, L, and U are the diagonal, strictly lower triangular, and strictly upper triangular parts of the matrix A, respectively (Golub & Van Loan, 2013).

The SSOR preconditioning method can be represented by the following formula:

$$\boldsymbol{M}_{\text{SSOR}} = \frac{1}{\omega(2-\omega)} (\boldsymbol{D} - \omega \boldsymbol{U}) \boldsymbol{D}^{-1} (\boldsymbol{D} - \omega \boldsymbol{L}), \tag{13}$$

where M_{SSOR} constitutes the preconditioner, and D, L, U, and ω are defined similarly to their roles in the SOR method. This symmetrical formulation enhances the stability and effectiveness of the preconditioning, particularly benefiting symmetric positive-definite matrices by optimizing the convergence properties of the iterative solver (Golub & Van Loan, 2013).

The choice of ω directly impacts the speed of convergence and the condition number of the matrix. Different problems and scenarios often require different choices of ω , which typically need to be determined based on the specific properties of the problem and through numerical experimentation (Golub & Van Loan, 2013). In the PETSc library, the default relaxation factor ω for both SOR and SSOR is set to 1, at which point SOR degenerates to GS preconditioning.

B.2.2. THRESHOLD PARAMETERS θ_T in AMG

In the AMG method, the threshold parameter θ_T determines whether the non-zero elements of the matrix are "strong" enough to be considered in the construction of a coarse grid during the multigrid process. This parameter is crucial for establishing the connectivity between coarse and fine grids in the hierarchical multilevel structure (Ruge & Stüben, 1987).

The AMG method solves the equation system through multiple levels of grids, each corresponding to a coarser version of the original problem. During this process, the threshold parameter is used to determine whether a given non-zero matrix element is strong enough to keep the corresponding grid points connected during coarsening. A lower threshold often leads to more elements being considered as strong connections, which might increase the complexity of the coarse grid but can help preserve the essential features of the original problem, thus improving the efficiency and convergence of the multigrid method.

• A higher threshold might result in fewer strong connections, thereby reducing the complexity of the coarse grid. However, this can weaken the effectiveness of the AMG method, especially in maintaining the features of the original problem.

668 Different values of θ_T directly influence the condition number of the preconditioned matrix. Selecting the appropriate 669 threshold parameter typically involves considering the specific structure and features of the problem, and adjustments are 670 made through experimental fine-tuning to achieve the optimal balance (Trottenberg et al., 2000). In the PETSc library, the 671 default threshold parameter θ_T is set to 0.

C. Algorithm Pseudocode

- 766
- 767
- 769

Algorithm 1 RNN-based Symbolic Discovery Process **Input:** RNN with parameter θ , the library of tokens \mathcal{L} $\tau \leftarrow []$ parent(0), sibling(0) \leftarrow empty node $x_0 \leftarrow parent(0) || sibling(0) / x$ is the concatenation of parent and sibling nodes $h_0 \leftarrow 0$ // Initialize hidden state of RNN for $t = 1, 2, \cdots$ do $(\psi_t, h_t) \leftarrow \text{RNN}(x_{t-1}, h_{t-1}; \theta)$ // ψ_t is the categorical distribution of the next token $\psi_t \leftarrow \text{ApplyConstraint}(\psi_t, \mathcal{L}, \boldsymbol{\tau})$ // Regularize the distribution Sample token $\tau_t \sim \psi_t$ if $Arity(\tau_t) > 0$ then // Arity($oldsymbol{ au}_i$) denotes the number of operands of $oldsymbol{ au}_i$ $parent(t) \leftarrow \boldsymbol{\tau}_t$ sibling(t) \leftarrow empty node else // When $\operatorname{Arity}(m{ au}_t)=0$, go back to the last incomplete operator node count $\leftarrow 0$ for $i = t, t - 1, \dots, 1$ do // Backward iteration count \leftarrow count + Arity $(\tau_i) - 1$ if count = 0 then $parent(t) \leftarrow \boldsymbol{\tau}_i$ sibling $(t) \leftarrow \boldsymbol{\tau}_{i+1}$ break if count = -1 then // The expression sequence is complete _ break $x_t \leftarrow \text{parent}(t) || \text{sibling}(t)$ **Output:** Prefix expression sequence τ Algorithm 2 Deep Symbolic Optimization for Matrix Preconditioning Parameter **Input** :RNN with initial parameter θ_0 , the library of tokens \mathcal{L} , batch size N, iteration number J, risk factor ε , and learning rate α $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$ $j \leftarrow 0$ repeat for i = 1, 2, ..., N do $\boldsymbol{\tau}^{(i)} \leftarrow \text{SymbolicDiscover}(\boldsymbol{\theta}, \mathcal{L})$ $\xi^* \leftarrow \{\xi \text{ in } \boldsymbol{\tau} \text{ as constant placeholder} : R(\boldsymbol{\tau}; \xi)\}$ // Constant optimization $\boldsymbol{\tau}^{(i)} \leftarrow \text{ReplaceConstant}(\boldsymbol{\tau}^{(i)}, \boldsymbol{\xi}^*)$ Compute \hat{g}_1 using $\boldsymbol{\tau}^{(i)}$ and $\boldsymbol{\theta}$ // See Eq. (11)Compute \hat{g}_2 as entropy gradient $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\hat{g}_1 + \hat{g}_2)$ // Update the parameter Train model: update p_{θ} via PPO by optimizing $J(\theta; \epsilon)$ **until** j = J or convergence **Output :** The best symbolic expression au^*

D. Experiment Settings

D.1. Datasets

1. Darcy Flow Problem

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

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

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

In our experiment, K(x, y) is generated using truncated Chebyshev polynomials. We convert the Darcy flow problem into a system of linear equations using the central difference scheme of Finite Difference Methods (FDM) (LeVeque, 2007). The coefficients of the Chebyshev polynomials serve as input features for our symbolic discovery framework.

2. Second-order Elliptic Partial Differential Equation

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

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

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 variable and its 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. Similar to the approach with the Darcy flow problem, we convert the PDE into a system of linear equations using the central difference scheme of FDM. The coefficients $a_0, a_1, a_2, a_{11}, a_{12}, a_{22}$ serve as input features for our symbolic discovery framework. When discussing SSOR preconditioning, we set a_1 and a_2 to zero to ensure the resulting matrix remains symmetric.

3. Biharmonic Equation

We consider the biharmonic equation, a fourth-order elliptic equation, defined on a domain $\Omega \subset \mathbb{R}^2$. The equation is expressed as follows (Ciarlet & Raviart, 1974; Glowinski & Pironneau, 1979; Barrata et al., 2023):

$$\nabla^4 u = f \quad \text{in } \Omega = [0, a] \times [0, b],$$

where $\nabla^4 \equiv \nabla^2 \nabla^2$ represents the biharmonic operator and $f = 4.0\pi^4 \sin(\pi x) \sin(\pi y)$ is the prescribed source term.

In our experiments, we construct the dataset by varying the solution domain $\Omega = [0, a] \times [0, b]$. We utilize the discontinuous Galerkin finite element method from the FEniCS library to transform this problem into a system of linear equations (Barrata et al., 2023). The parameters a, b of the domain serve as input features for our symbolic discovery framework.

4. Poisson Equation

We consider a two-dimensional Poisson equation, which can be described by the following equation (Wang et al., 2024; Hsieh et al., 2019; Zhang et al., 2022):

$$\nabla^2 u = f \quad \text{in } \Omega = [0, 1]^2.$$

Physical Contexts in which the Poisson Equation Appears: 1. Electrostatics; 2. Gravitation; 3. Fluid Dynamics.

In our experiments, we address the Poisson equation within a square domain , where both the boundary conditions on all four sides and the source term f on the equation's left-hand side are generated using third-order truncated Chebyshev polynomials. The finite difference method with a central difference scheme is employed to discretize the equation into a linear system. The Chebyshev coefficients serve as parameters for our symbolic discovery framework (Driscoll et al., 2014).

8238245. Thermal Problem

We consider a two-dimensional thermal steady state equation, which can be described by the following equation (Wang et al., 2024; Sharma et al., 2018; Koric & Abueidda, 2023):

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0,$$

where T is the temperature. We examine the steady-state thermal equation in thermodynamics. As with the previous equation, we still solve this equation in the square domain. The boundary temperatures on the left and right boundaries are determined by random values ranging from -100 to 0 and 0 to 100, respectively. The top and bottom boundary temperature functions are generated by third-order truncated Chebyshev polynomials. The boundary temperature and the coefficients of the Chebyshev polynomials serve as parameters for our symbolic discovery framework.

836 837 **D.2. Environment**

831

832

833

834

835

838

839

840 841

842

843 844

845

846

847 848

849

850

851

852 853 854

855

856

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

1. Environment (Env1):

- Platform: Windows11 version 22631.4169, WSL
- Operating System: Ubuntu 22.04.3
- CPU Processor: AMD Ryzen 9 5900HX with Radeon Graphics CPU, clocked at 3.30GHz
- 2. Environment (Env2):
 - Platform & Operating System: Ubuntu 18.04.4 LTS
 - CPU Processor: Intel(R) Xeon(R) Gold 6246R CPU at 3.40GHz
 - GPU Processor: GeForce RTX 3090 24GB
 - Library: CUDA Version 11.3

Speed tests for solving linear systems were performed in Env 1, while all training related to symbolic discovery was conducted in Env 2.

857 D.3. Training Data Generation

We employed an adaptive grid search to generate the training dataset. Initially, we traversed a coarse grid, sampling every 0.05, and from this dataset, we selected the three points with the smallest values. Subsequently, we conducted a finer grid search around these points, sampling every 0.001, to identify the point with the minimum value, which we designated as our optimal parameter. Particularly, after experimental validation confirmed the dataset's convexity, we utilized a binary search sampling method for a dataset derived from the second-order elliptic equation's SOR preconditioning. Starting with points at 0.0, 1.0, and 2.0, we compared these values. If the value at 0.0 was lowest, we computed at 0.5; if at 2.0, then at 1.5; and if at 1.0, then at both 0.5 and 1.5. This process was repeated until achieving a minimum point with a precision of 0.001.

For SOR preconditioning, we evaluated second-order elliptic equations, Darcy flow equations, and biharmonic equations, with solution time as the metric for optimal preprocessing parameters, achieved by minimizing solution time using the previously described grid method. In SSOR preconditioning, applied to second-order elliptic and Darcy flow equations, we utilized a hybrid metric that combined normalized computation time and iteration counts, aiming to simultaneously optimize both iteration counts and solution times. For AMG preconditioning, also examined with second-order elliptic and Darcy flow equations, we used the condition number of the preconditioned matrix as the metric, where a lower value indicates better performance.

- 873
- 874 875
- 876
- 877
- 878
- 879

880 D.4. Parameters of SymMAP

Experimental Setup. SymMAP is implemented using the LSTM architecture with one layer and 32 units. More details about the hyperparameters are provided in Table 7.

	Hyperparameter	Value
—	Number of LSTM layers	1
	Number of LSTM units	32
	Number of training samples	2,000,000
	Batch size	1,000
	Risk factor ε	0.05
	Minimal expression length	4
	Maximal expression length	64
	Learning rate	0.0005
	Weight of entropy regularization	0.03

Table 7: Hyperparameters of SymMAP (Default Model)

Restricting searching space. We employ specific constraints within our framework to streamline the exploration of
 expression spaces effectively and ensure they remain within practical and manageable bounds:

- 902
 903
 904
 905
 1. Bounds on expression length. To strike a balance between complexity and manageability, we set boundaries for expression lengths: a minimum of 4 and a maximum of 64 characters. This ensures that expressions are neither overly trivial nor excessively complicated.
- 2. Constant combination. We restrict expressions such that the operands of any binary operator are not both constants.
 This is out of the simple intuition that, if both operands are constants, the combination of the two can be precomputed and replaced with a single constant.
- 3. Inverse operator exclusion. We preclude unary operators from having their inverses as children to avoid redundant computations and meaningless expressions, such as in $\log(\exp(x))$.
 - 4. Trigonometric Constraints. Expressions involving trigonometric operators should not include descendants within their formulation. For instance, sin(x + cos(x)) is restricted because it combines trigonometric operators in a way that is uncommon in scientific contexts.

D.5. Computational Time for Related Algorithms

Dataset Generation Time:

- Darcy Flow Problem: 40 hours
- Second-order Elliptic Partial Differential Equation: 40 hours
- Biharmonic Equation: 100 hours
- Poisson Equation: 6 hours
- Thermal Problem: 6 hours
- SymMAP Execution Time: For each run, 1000 iterations are performed.
 - Without polynomials in the Token Library: approximately 800 seconds.
 - With polynomials in the Token Library: approximately 2600 seconds.

E. Experimental Data and Supplementary Experiments

7 E.1. Symbolic Expressions from Main Experiments

This section documents some of the learned expressions from the main experiments, corresponding to "SymMaP1" in Tables 1, 2, and 3.

• Second-order elliptic PDE problem, AMG preconditioning:

$$\frac{x_1x_3 + 1.0}{x_1 + 7.0}$$

Parameter meanings: x_1 - x_6 represent the coefficients a_{11} , a_{12} , a_{22} , a_1 , a_2 , and a_0 in the second-order elliptic equation.

• Second-order elliptic PDE problem, SSOR preconditioning:

$$\begin{array}{l} (2x_2 - \frac{1}{4x_1 + 2x_3})(-0.21785x_1^3 - 63.6118x_1^2x_2 + 0.206541x_1^2x_3 - 0.235667x_1^2x_4 \\ &\quad + 0.269472x_1^2 - 967.517x_1x_2^2 - 61.2291x_1x_2x_3 + 1.68205x_1x_2x_4 + 5.07925x_1x_2 \\ &\quad - 0.0221322x_1x_3^2 - 0.454257x_1x_3x_4 - 0.0693756x_1x_3 + 0.411528x_1x_4^2 + 0.0311608x_1x_4 \\ &\quad - 7.53439x_1 + 9506.4x_2^3 - 468.735x_2^2x_3 - 154.885x_2^2x_4 + 410.223x_2^2 - 25.5913x_2x_3^2 \\ &\quad + 7.92627x_2x_3x_4 + 5.30828x_2x_3 + 3.82512x_2x_4^2 - 7.0487x_2x_4 - 0.612507x_2 - 0.180432x_3^3 \\ &\quad - 0.0462734x_3^2x_4 + 0.310649x_3^2 + 0.257121x_3x_4^2 - 0.0962336x_3x_4 - 3.86012x_3 + 0.13906x_4^3 \\ &\quad - 0.389893x_4^2 + 0.3144x_4 - 0.0111835) \end{array}$$

Parameter meanings: x_1 - x_4 represent the coefficients a_{11} , a_{12} , a_{22} , and a_0 in the second-order elliptic equation.

• Darcy flow problem, SSOR preconditioning:

$$\frac{1.0}{x_1(x_{14}+x_4)+1.0}$$

Parameter meanings: x_1 - x_{16} represent the 16 coefficients of a second-order truncated Chebyshev polynomial in two dimensions, ordered as follows: 1, x, x^2 , x^3 , y, xy, x^2y , x^3y , y^2 , xy^2 , x^2y^2 , x^3y^2 , y^3 , xy^3 , x^2y^3 , and x^3y^2 .

• Darcy flow problem, AMG preconditioning:

$$1.0 + \frac{1.0}{1.0 + \frac{1.0}{x_{16} + x_3 + x_9^2 + 2.0}}$$

Parameter meanings: x_1 - x_{16} represent the 16 coefficients as described above.

• Biharmonic Equation, SOR preconditioning:

$$\left(\frac{1.0x_2}{-4x_2 - 1.0 + \frac{1.0}{x_2}} + 1.0\right)^4$$

Parameter meanings: x_1 and x_2 represent the length and width of the equation's boundary, respectively.

Biharmonic Equation, AMG preconditioning:

$$1.0 + \frac{1.0}{3.0 + \frac{1.0x_1 + 1.0}{x_2}} + \frac{1.0}{x_2}$$

Parameter meanings: x_1 and x_2 represent the length and width of the equation's boundary, respectively.

• Poisson Equation, SOR preconditioning:

$$\sqrt{\exp\left(\frac{1.0}{x_3 + \exp(2\exp(x_1^2))}\right)}$$

Parameter meanings: x_1 - x_8 represent the coefficients of two second-order truncated Chebyshev polynomials for the boundary functions.

• Poisson Equation, SSOR preconditioning:

$$1.15024107160485 (0.106506978919201x_2^2 + 1)^{1/16}$$

Parameter meanings: x_1 - x_8 represent the coefficients of two second-order truncated Chebyshev polynomials for the boundary functions.

• Poisson Equation, AMG preconditioning:

$$\frac{1.0}{x_8^2 + 7.0}$$

Parameter meanings: x_1 - x_8 represent the coefficients of two second-order truncated Chebyshev polynomials for the boundary functions.

• Thermal problem, SOR preconditioning:

$$\exp\left(\frac{0.778800783071405}{(1-x_6)^{1/4}}\right)^{1/4}$$

Parameter meanings: x_1 - x_4 represent the coefficients of the Chebyshev polynomial for the boundary temperature function on the upper and lower boundaries, while x_5 and x_6 represent the coefficients for the boundary temperature function on the left and right boundaries.

• Thermal problem, SSOR preconditioning:

$$1.0 - \frac{1.0}{\log\left(4.0(1 - 0.5x_6)^2\right)}$$

Parameter meanings: x_1 - x_4 represent the coefficients of the Chebyshev polynomial for the boundary temperature function on the upper and lower boundaries, while x_5 and x_6 represent the coefficients for the boundary temperature function on the left and right boundaries.

• Thermal problem, AMG preconditioning:

 $\frac{1.0}{2.71828182845905\exp(0.135335283236613x_6)+8.15484548537714}$

Parameter meanings: x_1 - x_4 represent the coefficients of the Chebyshev polynomial for the boundary temperature function on the upper and lower boundaries, while x_5 and x_6 represent the coefficients for the boundary temperature function on the left and right boundaries.

10371038 E.2. Interpretable Analysis Details

As shown in Table 8, the variables are defined as follows: in the first row, x_1 and x_2 represent the size of the boundary for PDE solutions; in the second row, x_2 represents the coefficient of a second-order coupling term; in the third row, x_4 is the coefficient of the fourth x-term multiplied by the first y-term in a two-dimensional Chebyshev polynomial; in the fourth row, x_2 again denotes the coefficient of a second-order coupling term; in the fifth row, x_1x_3 signifies the coefficient of a second-order non-coupling term.

	Table 8: Sym	bolic expression	ons learned from the fi	iam experiments	
	Precondition	Dataset	Symbolic e	xpression	
	SOR	Biharmonic	$\frac{10+10/(40+10/r_{0})+10/r_{1}}{10+10/r_{1}}$		-
	SOR	Elliptic PDE	$1.0 + 1.0/(x_2 + 1.0)$	$+1.0/(x_2 + 4.0))$	
	SOR	Darcy Flow	10 + 10/($r_4 + 1.0$	
	SSOR	Elliptic PDF	1.0 + 1.0/($x_2 + 1.0$	
	AMG	Elliptic PDE	$(x_1 x_2 + x_3)$	(-1)/7	
			(@1@3	-)/ •	-
E.3. Analysis of Hy	yperparameters				
The performance of	f SumMaD is primar	ily influenced l	by the learning rate of	the DNN betch size	a and detect size
conducted experime	ents to study the imp	ity infinitenceu i	ernarameters	i ule Kinin, batch siz	le, and dataset size.
		let of these hyp	erparameters.		
Symbolic Learning					
Symbolic Learning	g RNN Parameters:				
Table 9: Performan	g RNN Parameters:	nMaP under vai	rious symbolic learnin	g RNN parameters (1	ower condition num
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses	nMaP under van on optimizing A	rious symbolic learnin AMG preconditioning	g RNN parameters (l coefficients in the D	ower condition num arcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses	MaP under van on optimizing A	rious symbolic learnin AMG preconditioning	g RNN parameters (l coefficients in the D	ower condition num barcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate	hMaP under van on optimizing A Batch Size	rious symbolic learnin AMG preconditioning Condition number	g RNN parameters (I coefficients in the D Training time(s)	ower condition num arcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate	hMaP under van on optimizing A Batch Size 500	rious symbolic learnin, AMG preconditioning Condition number 6780	g RNN parameters (I coefficients in the D Training time(s) 1173.09	ower condition num Darcy Flow dataset.
Fable 9: Performand re preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01	MaP under van on optimizing A Batch Size 500 1000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51	ower condition num Darcy Flow dataset.
able 9: Performand re preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01	hMaP under var on optimizing A Batch Size 500 1000 2000	rious symbolic learnin AMG preconditioning Condition number 6780 5168 6898	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80	ower condition num arcy Flow dataset.
able 9: Performand re preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01	MaP under var on optimizing A Batch Size 500 1000 2000 500	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16	ower condition num arcy Flow dataset.
able 9: Performand e preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774	g RNN parameters (1 coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40	ower condition num barcy Flow dataset.
ble 9: Performand e preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001	hMaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85	ower condition num arcy Flow dataset.
able 9: Performand re preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718	g RNN parameters (1 coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45	ower condition num arcy Flow dataset.
able 9: Performand e preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17	lower condition num Darcy Flow dataset.
ble 9: Performand preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001 0.001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935 4718 5935 5935	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17 549.36	lower condition num Darcy Flow dataset.
Fable 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001 0.0005	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935 4718 5935 5935	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17 549.36 1324.00	lower condition num arcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001 0.0005 0.0001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935 4718 5935 5935 12228 7508	g RNN parameters (1 coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17 549.36 1324.00 837.18	lower condition num Darcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001 0.0005 0.0001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935 4718 5935 5935 12228 7508 6884	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17 549.36 1324.00 837.18 603.62	lower condition num Darcy Flow dataset.
Table 9: Performand are preferable). The	g RNN Parameters: ce comparison of Syn e experiment focuses Learning Rate 0.01 0.001 0.0005 0.0001	MaP under var on optimizing A Batch Size 500 1000 2000 500 1000 2000 500 1000 2000 500 1000 2000	rious symbolic learnin, AMG preconditioning Condition number 6780 5168 6898 5935 11774 5935 4718 5935 4718 5935 5935 12228 7508 6884	g RNN parameters (I coefficients in the D Training time(s) 1173.09 863.51 522.80 1104.16 676.40 505.85 1026.45 703.17 549.36 1324.00 837.18 603.62	lower condition num Darcy Flow dataset.

1083 Results in Table 9 indicate that an appropriate combination of RNN learning rate and batch size can enhance performance.

Dataset size:

Table 10: Performance comparison of SymMaP across varying dataset sizes (lower condition numbers indicate better
 performance). The experiment evaluates the optimization of AMG preconditioning coefficients for the Darcy Flow dataset.

Dataset size	Condition number	Training time (s)
10	7032	669.68
50	6980	737.80
100	4892	812.02
500	3811	699.54
1000	5345	703.17

1097
1098
1099Table 10 demonstrates that increasing the dataset size enhances the performance of symbolic expressions learned by SymMaP,
as expected.