NEURAL SYMBOLIC REGRESSION USING CONTROL VARIABLES

Xieting Chu
School of Computer Science
Zhejiang University
creatix@zju.edu.cn

Hongjue Zhao
School of Control Science and Engineering
Zhejiang University
hongjue0830@zju.edu.cn

Enze Xu
Department of Computer Science
College of William and Mary
exu03@wm.edu

Hairong Qi
Department of EECS
University of Tennessee, Knoxville
hqi@utk.edu

Minghan Chen
Department of Computer Science
Wake Forest University
chenm@wfu.edu

Huajie Shao
Department of Computer Science
College of William and Mary
hshao@wm.edu

ABSTRACT

Symbolic regression (SR) is a powerful technique for discovering the analytical mathematical expression from data, finding various applications in natural sciences due to its good interpretability of results. However, existing methods face scalability issues when dealing with complex equations involving multiple variables. To address this challenge, we propose SRCV, a novel neural symbolic regression method that leverages control variables to enhance both accuracy and scalability. The core idea is to decompose multi-variable symbolic regression into a set of single-variable SR problems, which are then combined in a bottom-up manner. The proposed method involves a four-step process. First, we learn a data generator from observed data using deep neural networks (DNNs). Second, the data generator is used to generate samples for a certain variable by controlling the input variables. Thirdly, single-variable symbolic regression is applied to estimate the corresponding mathematical expression. Lastly, we repeat steps 2 and 3 by gradually adding variables one by one until completion. We evaluate the performance of our method on multiple benchmark datasets. Experimental results demonstrate that the proposed SRCV significantly outperforms state-of-the-art baselines in discovering mathematical expressions with multiple variables. Moreover, it can substantially reduce the search space for symbolic regression. The source code will be made publicly available upon publication.

1 Introduction

Symbolic regression (SR) aims to uncover the underlying mathematical expressions from observed data [16, 9]. It has been widely used for scientific discovery across various disciplines [1, 29] owing to its ability to learn analytical expressions between the input and output. The implementation of SR involves two steps [15]. The first step is to predict the skeleton of mathematical expressions based on a pre-defined list of basic operations $(+, -, \times, \div)$ and functions $(\sin, \cos, \exp, \log)$. For instance, we can identify the skeleton of a symbolic equation as $f(x) = \log ax + \sin(bx) + c$. Next, we adopt optimization methods, such as Broyden–Fletcher–Goldfarb–Shanno (BFGS), to estimate the parameters a, b, c in the skeleton. The key challenges of SR lie in: 1) how to improve the accuracy and scalability for multiple input variables, and 2) how to speed up the discovery process.

In the past few decades, a plethora of SR methods [22] have been developed to discover underlying mathematical equations from data in science and engineering domains. One popular approach among them is genetic programming (GP) [4, 7, 27, 10, 2], which uses evolutionary operations, such as mutation, crossover, and selection, to estimate the symbolic expressions in a tree structure. However, GP would suffer from instability and its inference time is expensive in the context of multiple input variables [15]. Another method, SINDy [5], adopts sparse linear regression to discover the governing equations of dynamical systems. However, SINDy's performance relies heavily on prior knowledge of a known set of candidate functions, and it is difficult to uncover complex equations from data solely through linear regression. To overcome these limitations, some studies explore deep neural networks-based techniques, such as Deep Symbolic Regression (DSR) [25] and Transformer-based pre-training, for symbolic learning. Although these
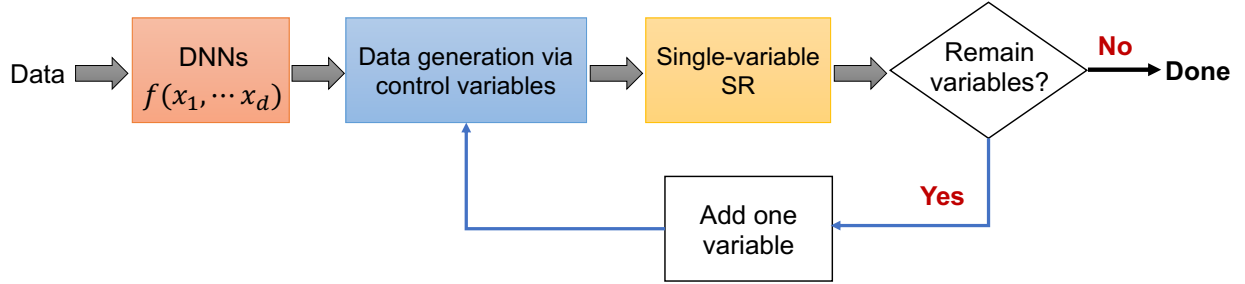
Figure 1: The overall framework of SRCV, consisting of three main components: i) learn a data generator using DNNs; ii) generate data for each independent variable via control variables; iii) apply single-variable SR to estimate the mathematical equation for the current independent variable.

approaches obtain good prediction accuracy, they do not scale well to mathematical equations with multiple variables. Recently, researchers develop Symbolic Physics Learner (SPL), a physics-informed Monte Carlo Tree Search (MCTS) algorithm for symbolic regression. While SPL outperforms most GP-based methods, it still struggles with multiple variables in mathematical expressions. In summary, existing methods suffer from scalability issues when dealing with complex multi-variable equations as they require a much larger search space to identify the combination of different variables. Thus, the question is, how can we reduce the search space of symbolic regression for complex equations involving multiple variables?

In this paper, we propose a novel neural symbolic regression with control variables (SRCV) that combines neural networks and symbolic regression to discover analytical expressions from data, as illustrated in Fig. 1. Inspired by divide and conquer [26], SRCV addresses the multi-variable symbolic regression by decomposing it into a set of single-variable SR problems and then combines the estimated symbolic equation for each variable in a bottom-up manner. The proposed method is performed in four steps as follows. 1) We learn a data generator from observed data using DNNs, allowing for generating data for a specific variable. 2) Generate data via control variables. Specifically, we generate data samples for the current independent variable by manipulating the previously learned variables and other control variables. For example, for estimating the symbolic expression of variable x_i, we can generate data samples by varying x_i while fixing the other variables. 3) Single-variable symbolic regression is employed to estimate the mathematical expression of the current variable based on the generated data in step 2. Here any symbolic regression models can be inserted into the framework. 4) We gradually add the remaining variables one by one to step 2 and proceed with step 3 until all the variables are covered. Extensive experimental results on multiple SR benchmarks demonstrate the superiority of our SRCV over the state-of-the-art methods in discovering complex multi-variable equations. Moreover, the proposed approach is able to discover complex expressions in a reduced search space.

Our main contributions are three-fold: 1) we propose SRCV, a simple and effective neural symbolic regression method using control variables; 2) we illustrate that the proposed method exhibits a significant reduction in search space for complex symbolic equations; 3) the evaluation results demonstrate that our method can significantly outperform the baselines in terms of accuracy and inference time.

2   Related Work

GP-based Symbolic Regression. Genetic Programming (GP) is one of the most popular algorithms for symbolic regression. The basic idea is to adopt the evolutionary operations, including mutation, crossover, and selection, to iteratively estimate the mathematical expressions until the desired accuracy is achieved. As a typical representative, the commercial software Eureqa [11] has been widely used in real-world applications. A recent study [23] combined genetic programming with reinforcement learning to enhance performance. While GP yields satisfactory results in many scenarios, it does not scale well to multiple input variables and is highly sensitive to hyperparameters [24].

DNNs-based Symbolic Regression. Some studies have employed DNN techniques [19, 18, 20] to discover symbolic equations from data. Early approaches proposed to replace the activation functions in DNNs with some basic functions like "sin(.)", "cos(.)", and "exp(.)". This substitution may lead to training instability and exploding gradient issues. Recently, AI-Feynman [31, 30] was developed to decompose the process of finding an equation into a flow based on the assumption of known physical properties. However, this method relies heavily on prior physics knowledge, such as symmetries or invariances. A more recent approach, Deep Symbolic Regression (DSR) [24], combined recurrent neural networks (RNN) with reinforcement learning for symbolic regression. Despite outperforming many GP-based approaches, DSR struggles with equations that contain multiple variables and constants.

Tree-based Symbolic Regression. Furthermore, a few recent studies proposed Monte Carlo tree search (MCTS) [6, 8, 21, 29] for symbolic regression. The MCTS is performed in the following four steps: 1) selection, 2) expansion, 3) simulation, and 4) backpropagation. It takes advantage of the trade-off between exploration and exploitation to better discover mathematical expressions. For instance, a most recent work developed Symbolic Physics Learner (SPL) [29] to accelerate discovery based on prior physics knowledge. However, SPL does not scale well to mathematical equations with many variables.

Pretraining-based Symbolic Regression. Inspired by large language models, researchers also adopted a pre-training technique based on Transformer [33, 17, 3] for the discovery of symbolic equations. For example, Biggio et al. [3] developed a large scale pre-training model for symbolic regression. To overcome the ill-posed problem in skeleton prediction, recent work developed an end-to-end (E2E) symbolic regression by training Transformer on a large amount of synthetic data. However, Transformer-based symbolic regression requires a ton of training data, which is not practical in real world applications. Moreover, it does not scale well to high-dimensional functions with many variables.

3 Proposed Method

In this section, we first state the problem of symbolic regression, and then elaborate on the proposed SRCV. A walk-through example is provided to enhance the understanding of our approach. Furthermore, we study how the proposed method effectively reduces the search space in symbolic regression.

3.1 Problem Statement

Given a set of N data samples $\mathcal{D} = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^{N}$, where $\mathbf{x}^{(n)} \in \mathbb{R}^d$ and $y^{(n)} \in \mathbb{R}$. Here d denotes the dimension of input data. The goal of symbolic regression is to learn an analytical mathematical expression, $y = f(\mathbf{x}) = f(x_1, x_2, \ldots, x_d)$, based on observed data $\mathcal{D}$.

3.2 Proposed SRCV

To improve the accuracy and scalability for multi-variable SR, we propose a novel neural symbolic regression with control variables (SRCV) to decompose it into a set of single-variable SR problems. The key idea is to learn a data generator from observed data using DNNs, and then use it to generate data samples by manipulating an independent variable each time. After that, we estimate the symbolic equation of the current variable based on its generated samples and then combine the discovered equations by adding variables one by one. Fig. 1 shows the overall framework of the proposed SRCV, which consists of three main parts: i) data generator with DNNs; ii) data generation via control variables; iii) single-variable symbolic regression (SR). Below, we will describe these three components in detail.

Data Generator with DNNs. In many real-world applications, we only obtain the data samples from multiple input variables, rather than from a single control variable. In order to control data generation for a single variable, we first need to learn a data generator using deep neural networks (DNNs). After learning the mapping function between the input and output, $f(x_1, x_2, \ldots, x_d)$, we can manipulate the input variables to generate different data samples as needed. For instance, we can vary variable x_1 while keeping the other variables fixed to generate data for x_1, i.e., $f_{x_2, \ldots, x_d}(x_1)$.

Data Generation via Control Variables. For this part, we aim to generate different data samples by controlling the input variables. As mentioned earlier, our goal is to decompose multi-variable SR into a set of single-variable SR problems. Suppose that we have learned a symbolic equation of the prior i variables, denoted by $x_{\leq i}$ $(i = 1, 2, \ldots)$. Next, we will estimate the mathematical equation of a newly added variable x_{i+1}. To achieve this, we use the above data generator to generate K groups of data samples for the current variable x_{i+1}. For each group, we will generate M data samples via varying the previously learned variables, i.e., $x_{\leq i}$, given a specific value of x_{i+1}, as shown in Fig. 2. Specifically, we randomly assign M different values to $x_{\leq i}$ while keeping other control variables $x_{\geq i+2}$ fixed and assigning a value to x_{i+1}. Then they will be fed into the data generator, denoted by $f_{x_{\geq i+2}}(x_{\leq i}, x_{i+1})$, to produce M samples for a given x_{i+1}. Here we use $\mathbf{F}^k$ to represent the k-th group of samples for x_{i+1}, and $\mathbf{X}^k$ to represent different values of previously learned variables $x_{\leq i}$. By randomly choosing K different values for x_{i+1}, we can generate K groups of data samples $\mathbf{F} = \{\mathbf{F}^k\}_{k=1}^{K}$. Our next step is to perform single-variable symbolic regression to estimate the expression of x_{i+1} based on the generated samples.

Single-Variable Symbolic Regression. We propose single-variable SR to predict the mathematical expression for the current independent variable x_{i+1}. The key idea is to estimate the coefficients in the skeleton of previously learned variables, e.g., $f_{x_{\geq i+1}}(x_{\leq i}) = C_1 x_i + C_2 x_{i-1} x_1 + \cdots + C_j$, using the generated samples of x_{i+1}. As illustrated in Fig. 3, our approach is performed in two steps. (1) We adopt optimization techniques, such as BFGS, to estimate K groups of coefficients $\mathbf{C}^K = \{C_1^k, \ldots, C_j^k\}_{k=1}^{K}$ in the skeleton using K groups of data samples $\{\mathbf{F}^k\}_{k=1}^{K}$ and
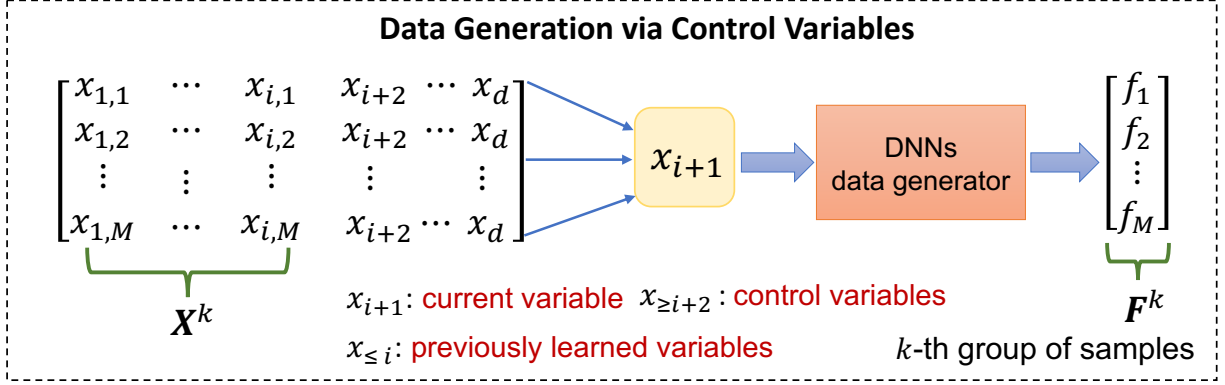
Figure 2: The framework of data generation with control variables. Specifically, we generate a group of data points for a newly added variable x_{i+1} assigned with a random value and then vary the previously learned variables while fixing other control variables. By choosing K different values for the current variable x_{i+1}, we can generate K groups of data samples.
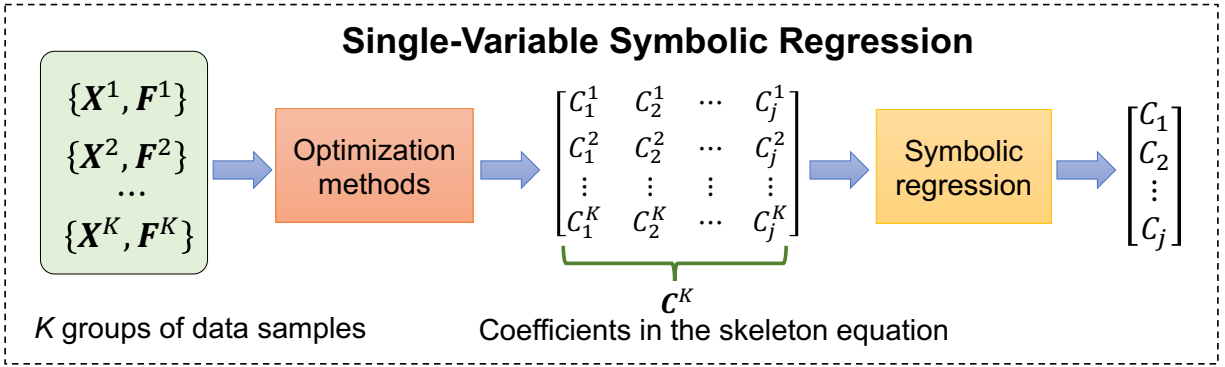

Figure 3: The framework of single-variable SR. It is performed in two steps: 1) we first use an optimization method, such as BFGS, to estimate K groups of coefficients for the current independent variable; 2) we then use a single-variable SR model to estimate coefficients in the mathematical equation related to the current variable.

the corresponding values of previously learned variables $\{\mathbf{X}^k\}_{k=1}^K$. Here, the coefficient C_j in the skeleton can be viewed as a function of variable x_{i+1}. This step enables us to obtain K groups of data samples $\mathbf{C}^K$ related to variable x_{i+1} by manipulating it with K different values. (2) We then apply symbolic regression to estimate the mathematical expression about x_{i+1} given K groups of $\mathbf{C}^K$ in the first step. Specifically, we feed the coefficient matrix $\mathbf{C}^K$ and the corresponding K different values of x_{i+1} into a symbolic model to estimate its skeleton and the corresponding coefficients, $\{C_1, \ldots, C_j\}$. Finally, we repeat the above two steps by adding variables one by one until all the variables are covered.

3.3 A Walk-through Example

To better understand the proposed method, we use a walk-through example to explain its core idea. Take $y = x_1 x_2 + 2x_2 + 2$ as an example. Given a set of data points $\{x_1^{(n)}, x_2^{(n)}, y^{(n)}\}_{n=1}^N$, we first adopt DNNs to learn a mapping function $f(x_1, x_2)$ between the input variables x_1, x_2 and the output y, which will serve as a data generator. Then we use the data generator to generate different data samples for the independent variable x_1 by varying x_1 while keeping variable x_2 unchanged (e.g., $x_2 = 2$), i.e., $f_{x_2}(x_1)$. Next, we leverage a symbolic regression model, such as GP and MCTS, to estimate the mathematical equation about x_1, e.g., we get $f_{x_2}(x_1) = 2x_1 + 6$. Since it is hard to directly derive x_2 from the discovered equation $f_{x_2}(x_1)$, we need to convert it into the following skeleton, $f_{x_2}(x_1) = C_1 x_1 + C_2$, where C_1 and C_2 can be viewed as a function of x_2 that need to be estimated later. After that, we add another independent variable x_2 to the data generator $f(x_1, x_2)$, and then generate M data samples given a random value of x_2, as shown in Fig. 2. By choosing K different values of x_2, we can generate K groups of data samples, denoted by $\{\mathbf{F}^k(x_1, x_2)\}_{k=1}^K$. The next step is to use an optimization method, such as BFGS, to estimate the k-th group of coefficients C_1^k and C_2^k in the skeleton $f_{x_2}(x_1)$, given $\mathbf{F}^k(x_1, x_2)$ and $\mathbf{X}^k = [x_{1,1}, x_{1,2}, \ldots, x_{1,M}]^\top$.

Finally, we apply single-variable symbolic regression to estimate the symbolic regression about x_2 given K groups of coefficients $\{C_1^k\}_{k=1}^K$ and $\{C_2^k\}_{k=1}^K$, as shown in Fig. 3. For instance, we can get $C_1 = x_2$ and $C_2 = 2x_2 + 2$. Since there are no remaining variables, we complete the process of discovering symbolic equation. If there are additional variables, we repeat this process to estimate their symbolic expressions until all the variables are covered.

3.4 Reduction of Search Space

We also analyze the relationship between the complexity of a mathematical expression and search space, and then illustrate that the proposed method can significantly reduce the search space. In this work, the complexity of an expression is defined below.

Definition 1. Following prior work [25], complexity is defined as twice the number of binary operators $\{+, -, \times, \div\}$, denoted by N_b, plus the number of unary operators $\{\sin, \cos, \exp, \log\}$, denoted by N_u, in the equation. Mathematically, the complexity can be formulated as $2N_b + N_u$.

The main reason why we define the above complexity is that it is identical to the number of nodes in an expression tree minus one. Plus, most existing symbolic regression and brute force methods often adopt the expression tree for heuristic searching. Hence, we can use this metric to measure the difficulty of symbolic regression.

Fig. 4 shows the relationship between complexity and search space for our method and the state-of-the-art MCTS based on 1000 equations with different complexity. Regarding how to sample these equations, please refer to the detailed description in Appendix A. We can see from the two black curves search space will rise as the complexity is increased. Our method can significantly reduce the search space for discovering the same equation compared to the original MCTS in [29]. The blue dashed line and solid line respectively represent the brute force and MCTS. We can see that our method can discover more complex equations under the same search space for both brute force and MCTS. For example, the original MCTS can discover an equation with a complexity of 16, while our method can estimate an equation with a complexity of 31, as shown in Fig. 4.
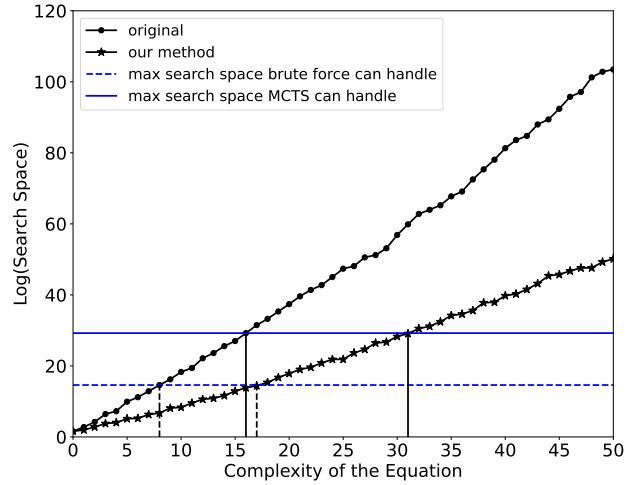

Figure 4: The relationship between complexity and search space for different methods based on 1000 equations with different complexity.

3.5 Algorithm Summary

We summarize the proposed method in Algorithm 1. We first learn a data generator from observed data using DNNs in Line 3. Lines 5-15 aim to generate K groups of data samples $\mathbf{F} = \{\mathbf{F}^k\}_{k=1}^K$ and $\mathbf{X} = \{\mathbf{X}^k\}_{k=1}^K$ by manipulating the current variable x_{i+1} with K different values. Then, we use optimization methods to estimate the coefficients $\mathbf{C}^K$ in the skeleton based on $\mathbf{F}$ and $\mathbf{X}$ in Line 16. In Line 17, we apply single-variable SR to estimate the symbolic equation of x_{i+1} based on K groups of $\mathbf{C}^K$ and the current variable. We will repeat this process until all the variables are completed.

4 Experiment

In this section, we carry out extensive experiments to evaluate the performance of SRCV. We first compare the discovery rate of our method with state-of-the-art baselines on two SR benchmarks. Next, we apply SRCV to identify the governing equations of two gene regulatory networks. Finally, we perform ablation studies to explore the impact of certain hyper-parameters on symbolic regression.

4.1 Datasets

We use two SR benchmark datasets, Nguyen [32] and Jin [14], for the first set of experiments. To illustrate the effectiveness of our method on complex regression, we specifically select equations containing at least two variables.

Algorithm 1 Proposed SRCV

1: **Input:** N data samples $\mathcal{D} = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^N$.
2: **Initial:** Previously learned variables $X_v = \{\}$ and controlled variables $X_c = \{x_1, \ldots, x_d\}$.
3: Learn a data generator using DNNs, $f(x_1, \ldots, x_d)$, based on data $\mathcal{D}$;
4: **for** $i = 0, \ldots, d - 1$ **do**
5:     $X_c \leftarrow X_c - \{x_{i+1}\}$ // remove one controlled variable;
6:     /*Data generation via manipulating the newly added variable x_{i+1}*/
7:     Assign random values to controlled variables in X_c;
8:     $\mathbf{F} = \{\}, \mathbf{X} = \{\}, \mathbf{X}_{i+1} = \{\}$;
9:     **for** $k = 1, \ldots, K$ **do**
10:         Assign a random value to x_{i+1} for each k;
11:         Generate a group of samples $\mathbf{F}^k$ for the current variable x_{i+1}; Each group generates M samples about previously learned variables in X_v, denoted by $\mathbf{X}^k$;
12:         $\mathbf{F} \leftarrow \mathbf{F} + \{\mathbf{F}^k\}$;
13:         $\mathbf{X} \leftarrow \mathbf{X} + \{\mathbf{X}^k\}$;
14:         $\mathbf{X}_{i+1} \leftarrow \mathbf{X}_{i+1} + \{x_{i+1}\}$;
15:     **end for**
16:     Use optimization methods to estimate the coefficient matrix $\mathbf{C}^K$ based on K groups of samples $\mathbf{F}$ and $\mathbf{X}$;
17:     Apply single-variable SR to estimate the symbolic equation of x_{i+1} based on $\mathbf{C}^K$ and $\mathbf{X}_{i+1}$;
18:     $X_v \leftarrow X_v + \{x_{i+1}\}$ // add one variable;
19: **end for**
20: **Output:** Discover analytical mathematical expression $y = f(\mathbf{x})$.

We also evaluate our method on two gene regulatory networks, including the genetic toggle switch and the repressilator, using synthetic data. Detailed descriptions of these datasets are presented in Appendix B.

4.2 Baselines

Four baseline approaches are used for comparison with the proposed SRCV.

- Symbolic Physics Learner (SPL) [29]. This method incorporates prior knowledge into Monte Carlo tree search for scientific discovery.
- Deep Symbolic Regression (DSR) [25]. It combines RL-based search method and recurrent neural networks (RNN) for symbolic regression.
- Gplearn (GP) [28]. It is a classic genetic programming method implemented in Python.
- Neural-Guided Genetic Programming (NGGP) [23]. It is a hybrid method that combines RNN with GP for symbolic regression.

4.3 Experimental Setup

In the experiments, we have a pre-defined list of basic operations $(+, -, \times, \div, \text{const})$ and basic functions $(\sin, \cos, \exp, \log)$. For SR benchmarks, we generate $N = 8000$ data samples and then split them into 6400 and 1600 for training and validation, respectively. The proposed SRCV aims to discover the underlying mathematical expressions from data based on the above two lists of candidate operations. The discovered equations will be compared with the ground-truth expressions. For the data generator, we use three fully connected layers (MLP) with hidden sizes of 128, 256, and 128, respectively. Then we train the MLP using Adam optimizer with an initial learning rate of 0.1 and cosine annealing schedule. In addition, we use a single batch containing all input data due to the small number of training samples. For single-variable symbolic regression, we choose $M = 200$ data samples for the current independent variable with $K = 200$ different values. Also, we adopt MCTS in the prior work [29] to estimate the symbolic equation with a single variable. This paper will use these hyperparameters in the following experiments, unless specified otherwise. Note that we will conduct ablation studies to investigate the impact of some important hyperparameters on the prediction performance of our method.

Evaluation Metrics. We run 10 independent tests for each case and calculate the recovery rate for each model. A successful discovery is evaluated using the following two criteria: i) prediction precision and ii) equation equivalence to ground truth. First, the mean square relative error (MSRE) between the prediction and ground truth should be less than 10^{-3}. Second, the discovered equation should be in an identical or equivalent symbolic form to the target equation. We manually check the discovered symbolic equations to ensure their correctness.

4.4 Evaluation on SR Benchmarks

First, we evaluate the proposed SRCV on two widely used SR benchmarks: Nguyen and Jin. Table 1 illustrates the comparison of discovery rate for different methods using 10 random seeds. It can be observed that our method achieves higher recovery rates than the baselines. This is because the proposed SRCV adopts the similar idea of "divide and conquer" that decomposes multi-variable SR into a subset of single-variable SR problems. Besides, our method can significantly reduce the search space of symbolic regression, thus speeding up the discovery. Please refer to the comparison of computational cost in Appendix D.

Table 1: Recovery rate comparison of the proposed SRCV and other baselines on SR benchmarks, Nguyen and Jin. Our method significantly outperforms the baselines in terms of the average recovery rate. Note that GP does not work well since it only learns an approximated equation.

Benchmark	Expression	SRCV (ours)	SPL	NGGP	DSR	GP
Nguyen-09	$\sin(x) + \sin(y^2)$	90%	100%	40%	30%	20%
Nguyen-10	$2\sin(x)\cos(y)$	90%	70%	100%	100%	90%
Nguyen-11	x^y	70%	70%	100%	90%	0%
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	70%	30%	10%	0%	10%
Jin-1	$2.5x^4 - 1.3x^3 + 0.5y^2 - 1.7y$	70%	0%	0%	0%	0%
Jin-2	$8.0x^2 + 8.0y^3 - 15$	100%	90%	70%	50%	10%
Jin-3	$0.2x^3 + 1.5y^3 - 1.2y - 0.5x$	90%	90%	0%	0%	0%
Jin-4	$1.5\exp(x) + 0.5\cos(y)$	100%	100%	40%	10%	0%
Jin-5	$6.0\sin(x)\cos(y)$	100%	80%	100%	100%	0%
Jin-6	$1.35xy + 5.5\sin((x - 1.0)(y - 1.0))$	0%	0%	0%	0%	0%
Average		**78%**	63%	46%	38%	13%

4.5 Evaluation on Gene Regulatory Networks

Next, we apply the proposed method to discover the underlying governing equations of two classic gene regulatory networks, the genetic toggle switch and the repressilator.

Genetic toggle switch. The genetic toggle switch [13] is a synthetic gene regulatory network that has been extensively studied as a fundamental concept in the field of synthetic biology. It has numerous prospective applications in biotechnology, such as the development of biosensors, gene therapies, and synthetic memory devices. The genetic toggle switch consists of two mutually repressive genes controlled by their respective promoters, creating a bistable system that can be toggled between two stable states as follows.

$$\begin{cases} \dfrac{dU}{dt} = \dfrac{\alpha_1}{1 + V^\beta} - U \\ \dfrac{dV}{dt} = \dfrac{\alpha_2}{1 + U^\gamma} - V, \end{cases} \tag{1}$$

where α_1 and α_2 are the synthesis rates of repressors U and V, respectively. β and γ are the cooperativities of repression on two promoters.

In this experiment, following the bistable region in prior work [13], we choose $\alpha_1 = 4$, $\alpha_2 = 4$, $\beta = 3$, $\gamma = 3$, and the initial conditions $U(0), V(0) \in [0, 4]$. To train our model, we generate 1000 trajectories by randomly choosing 1000 initial conditions. We use 800 of them as training data and the remaining 200 as validation data. The time span of each trajectory is $t \in [0, 1]$ with a sampling time interval of 0.01. Namely, we sample 100 data points for each trajectory.

Fig. 5 (a) illustrates the predicted trajectories of the genetic toggle switch using SRCV with a random initial condition. It can be observed that SRCV precisely predicts the trajectory, closely matching the ground truth obtained from the ODE solver (odeint). The mean square relative error (MSRE) of our method is about 9.03×10^{-4}. Importantly, our method successfully discovers the underlying governing equation from observed data as follows. We can see that it is quite close to the target model in Eqs. 1. We also present the experimental results of the *baselines* in Appendix F.

$$\begin{cases} \dfrac{dU}{dt} = \dfrac{3.919}{0.972 + V^3} - U \\ \dfrac{dV}{dt} = \dfrac{3.921}{0.972 + U^3} - V, \end{cases} \tag{2}$$
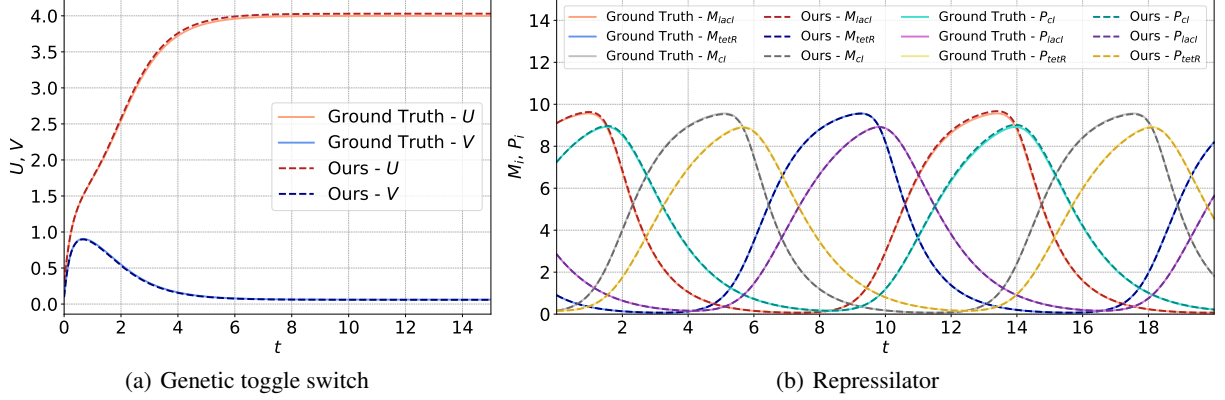
7

(a) Genetic toggle switch

(b) Repressilator

Figure 5: Trajectory prediction of the genetic toggle switch and the repressilator using SRCV. The SRCV can precisely predict their trajectories, closely matching the ground truth.

Repressilator. The Repressilator [12] is another type of gene regulatory network that exhibits oscillatory behavior. This model is critical in studying the dynamics of genetic circuits and provides insights into the principles of oscillatory systems in biology. Comprising three genes, it operates through a feedback loop in which each gene produces a repressor protein that suppresses the expression of the subsequent gene. This process results in a cyclic pattern of gene expression, described as follows.

$$\begin{cases} \dfrac{dM_i}{dt} = -M_i + \dfrac{\alpha}{1 + P_j^n} + \alpha_0 \\ \dfrac{dP_i}{dt} = -\beta \left(P_i - M_i\right) \end{cases} \quad \begin{pmatrix} i = lacI, tetR, cI \\ j = cI, lacI, tetR \end{pmatrix}, \tag{3}$$

In Eqs. 3, P_i denotes the repressor protein concentrations, and M_i represents the corresponding mRNA concentrations, where i is $lacI$, $tetR$, or cI. If there are saturating amounts of repressor, the number of protein copies produced from a given promoter is α_0. Otherwise, this number is $\alpha + \alpha_0$. β represents the ratio of the protein decay rate to the mRNA decay rate, and n is a Hill coefficient.

In this experiment, we set $\beta = 1$, $\alpha_0 = 10^{-5}$, $\alpha = 10$, $n = 3$, and the initial conditions $M_i, P_j \in [0, 5]$. To train our model, we generate 5000 trajectories using 5000 random initial conditions. They are split into 4000 training data and 1000 validation data respectively. The time span of each trajectory is $t \in [0, 4]$ with a sampling time interval of 0.01.

Fig. 5 (b) illustrates the trajectory predictions of repressilator using our method. We can see that the trajectory predicted by SRCV closely aligns with the ground truth, with MSRE of about 7.54×10^{-5}. Additionally, our method successfully identifies the underlying governing equations, which are provided in Appendix E.

4.6 Ablation Studies

Effect of M in data generation. First, we examine the impact of M different values of previously learned variables in data generation on the recovery rate. As shown in Table 2, when M varies from 50 to 200, the recovery rates of our approach remain fairly consistent. This suggests that our method is not sensitive to the number of generated samples M as it is sufficiently large.

Table 2: Effect of M samples on recovery rate of the proposed SRCV using Nguyen.

Benchmark	Expression	M=50	M=100	M=150	M=200
Nguyen-09	$\sin(x) + \sin(y^2)$	90%	90%	90%	90%
Nguyen-10	$2\sin(x)\cos(y)$	90%	90%	90%	90%
Nguyen-11	x^y	70%	70%	70%	70%
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	60%	70%	70%	70%

Effect of K groups in data generation. Next, we study the effect of K groups of generated data samples for the current variable on the recovery rate of SRCV. We can observe from Table 3 that the recovery rates almost keep the same as K changes from 50 to 200.

Effect of N training samples. Lastly, we investigate the influence of the training data size (N) on the proposed SRCV. It can be seen that our method achieves good performance when the number of training samples is sufficiently large. If

Table 3: Effect of K groups on the recovery rate of our SRCV using Nguyen.

Benchmark	Expression	K=50	K=100	K=150	K=200
Nguyen-09	$\sin(x) + \sin(y^2)$	90%	90%	90%	90%
Nguyen-10	$2\sin(x)\cos(y)$	90%	90%	90%	90%
Nguyen-11	x^y	70%	70%	60%	70%
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	60%	70%	70%	70%

the equations are more complex, it might be necessary to increase the amount of training data provided to the DNNs for optimal results.

Table 4: Effect of N training samples on the recovery rate of our SRCV using Nguyen.

Benchmark	Expression	N=800	N=1600	N=3200	N=4800	N=6400
Nguyen-09	$\sin(x) + \sin(y^2)$	90%	90%	90%	90%	90%
Nguyen-10	$2\sin(x)\cos(y)$	90%	90%	90%	90%	90%
Nguyen-11	x^y	70%	70%	70%	70%	70%
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	60%	60%	60%	70%	70%

5   Conclusion

In this work, we developed two non-exemplar-based methods, YONO and YONO+, for class-incremental learning. Specifically, YONO only needs to store and replay one prototype for each class without generating synthetic data from stored prototypes. As an extension of YONO, YONO+ proposed to create synthetic replay data from stored prototypes via a high-dimensional rotation matrix and Gaussian noise. The evaluation results on multiple benchmarks demonstrated that both YONO and YONO+ can significantly outperform the baselines in terms of accuracy and average forgetting. In particular, the proposed YONO achieved comparable performance to YONO+ with data synthesis. Importantly, this work offered a new perspective of optimizing class prototypes for exemplar-free incremental learning.

References

[1] A. M. Alaa and M. van der Schaar. Demystifying black-box models with symbolic metamodels. *Advances in Neural Information Processing Systems*, 32, 2019.

[2] I. Arnaldo, K. Krawiec, and U.-M. O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 879–886, 2014.

[3] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. PMLR, 2021.

[4] L. Billard and E. Diday. From the statistics of data to the statistics of knowledge: symbolic data analysis. *Journal of the American Statistical Association*, 98(462):470–487, 2003.

[5] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[6] T. Cazenave. Monte-carlo expression discovery. *International Journal on Artificial Intelligence Tools*, 22(01):1250035, 2013.

[7] T. Cornforth and H. Lipson. Symbolic regression of multiple-time-scale dynamical systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 735–742, 2012.

[8] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[9] M. Cranmer, A. Sanchez Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33:17429–17442, 2020.

[10] F. O. de Franca and G. S. I. Aldeia. Interaction–transformation evolutionary algorithm for symbolic regression. *Evolutionary computation*, 29(3):367–390, 2021.

[11] R. Dubčáková. Eureqa: software review, 2011.

[12] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[13] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[14] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

[15] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton. End-to-end symbolic regression with transformers. In *NeurIPS*, 2022.

[16] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

[17] G. Lample and F. Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[18] M. Landajuela, C. S. Lee, J. Yang, R. Glatt, C. P. Santiago, I. Aravena, T. Mundhenk, G. Mulcahy, and B. K. Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.

[19] M. Landajuela, B. K. Petersen, S. Kim, C. P. Santiago, R. Glatt, N. Mundhenk, J. F. Pettit, and D. Faissol. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, pages 5979–5989. PMLR, 2021.

[20] H. Li, Y. Weng, and H. Tong. Console: Convex neural symbolic learning. *arXiv preprint arXiv:2206.00257*, 2022.

[21] Q. Lu, F. Tao, S. Zhou, and Z. Wang. Incorporating actor-critic in monte carlo tree search for symbolic regression. *Neural Computing and Applications*, 33:8495–8511, 2021.

[22] N. Makke and S. Chawla. Interpretable scientific discovery with symbolic regression: A review. *arXiv preprint arXiv:2211.10873*, 2022.

[23] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

[24] B. K. Petersen, M. Landajuela, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[25] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

[26] E. A. Posner, K. E. Spier, and A. Vermeule. Divide and conquer. *Journal of Legal Analysis*, 2(2):417–471, 2010.

[27] M. Quade, M. Abel, K. Shafi, R. K. Niven, and B. R. Noack. Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1):012214, 2016.

[28] M. Sipper, T. Halperin, I. Tzruia, and A. Elyasaf. Ec-kity: Evolutionary computation tool kit in python with seamless machine learning integration. *SoftwareX*, 22:101381, 2023.

[29] F. Sun, Y. Liu, J.-X. Wang, and H. Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *ICLR*, 2023.

[30] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Advances in Neural Information Processing Systems*, 33:4860–4871, 2020.

[31] S.-M. Udrescu and M. Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

[32] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12:91–119, 2011.

[33] M. Valipour, B. You, M. Panju, and A. Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

A   Generate Expression Tree in Fig. 4

We introduce how to generate a mathematical expression with a specific complexity to estimate the search space in symbolic regression. In this work, we attempt to sample equations uniformly, i.e., all *valid* equations should have the same probability to be selected. A valid equation defined as follows.

Definition 2. An equation is *valid* if it is a proper mathematical equation consisting of M_t ($M_t = 5$) terminal symbols $\{x_1, x_2, x_3, x_4, const\}$, M_u ($M_u = 4$) unary operators $\{\sin, \cos, \log, \exp\}$, and M_b ($M_b = 4$) binary operators $\{+, -, \times, \div\}$. Moreover, it should not contain nested unary operators, such as $sin(cos(x_1) + 1)$ or $log(sin(x_1))$, since they are not very meaningful in most real cases.

Before describing our sampling method, we first define F_i and G_i that will be used for sampling.

Definition 3. Let F_i and G_i be the number of valid equations with complexity i containing no unary operator and at least one unary operator, respectively. The criterion for distinguishing two equations is based on the structure of their expression trees rather than their algebraic equivalence. According to this, some equations that are algebraically equivalent, such as $x_1 + x_2 + x_3$, will be counted multiple times due to different tree structures. Nevertheless, it almost has no influence on the overall results.

Next, we introduce the idea of calculating F_i and G_i using dynamic programming. First, we consider F_i. When the complexity i is 0, the valid expression set contains only M_t terminals without unary operators, so $F_0 = M_t$. When the complexity is greater than or equal to 1, we need to consider the root of the expression tree. Since F_i counts an expression tree with no unary operator, its root has to be a binary operator with M_b possibilities. Suppose the left subtree has complexity of j ($j = 0, \ldots, i - 2$), then the complexity of the right subtree is $i - j - 2$. As a result, it has $F_j F_{i-j-2}$ feasible options given a certain j. Finally, we can sum up all the cases and multiply the result by M_b to get F_i.

$$F_i = \begin{cases} M_t & i = 0 \\ M_b \sum_{j=0}^{i-2} F_j F_{i-j-2} & i > 0, \end{cases} \tag{4}$$

For G_i, when the complexity is 0, there is no unary operator, so $G_0 = 0$. When the complexity is greater than or equal to 1, the root can be either a unary operator or a binary operator. (1) If the root is a unary operator, we have $M_u F_{i-1}$ options for tree structures. (2) If the root is a binary operator, we assume that the left subtree has complexity of j ($j = 0, \ldots, i - 2$), then the right subtree has the complexity of $i - j - 2$. According to the Definition 3, one or both of the subtrees should contain at least one unary operator, so the number of options for tree structures should be $G(i,j) = G_j G_{i-j-2} + G_j F_{i-j-2} + F_j G_{i-j-2}$. Since the root has M_b choices in this scenario, we can multiply $G(i,j)$ by M_b to get the total options for a binary operator. Combining (1) and (2), we can get G_i as follows.

$$G_i = \begin{cases} 0 & i = 0 \\ M_u F_{i-1} + M_b \sum_{j=0}^{i-2} G_j G_{i-j-2} + G_j F_{i-j-2} + F_j G_{i-j-2} & i > 0, \end{cases} \tag{5}$$

Lastly, we summarize the sample subroutine in Algorithm 2. First, we need to figure out whether a mathematical expression contains a unary operator or not. The probability of containing at least one unary operator is $G_i/(F_i + G_i)$, and that of not containing one is $F_i/(F_i + G_i)$. If there is no unary operator, we use SAMPLEBYNU in Lines 5-17 to generate the expression. This function operates as follows: if the complexity is 0, we select a terminal symbol M_u ($M_u = 4$) as mentioned above. Otherwise, we sample the operator at the root and the complexity of the left subtree according to the number of their options. After this, we recursively invoke subroutines to generate the subtrees. If there exists at least one unary operator, we use SAMPLEBYU instead, which uses a similar way to that in SAMPLEBYNU. By doing this, we can generate an expression tree with a specified complexity.

B   Dataset Description

First, we introduce how to generate training data and validation data using two SR benchmarks. As shown in Table 5, we choose a certain range and then generate 8000 data samples. Then we split them into 6400 and 1600 for training and validation, respectively.

Next, we describe how to generate the synthetic data for two gene regulatory networks: the genetic toggle switch and the repressilator. For the genetic toggle switch, we generate 1000 trajectories by randomly choosing 1000 initial conditions. We use 800 of them as training data and the remaining 200 as validation data. The time span of each

---

Algorithm 2 Expression Tree Sampling Algorithm

1: **Input:** Target complexity N, and pre-calculated F_i, G_i
2: **function** SAMPLEBYWEIGHT($weights$)
3:     **return** i with possibility $\frac{weights_i}{\sum weights}$. $weights$ indexed from 0.
4: **end function**
5: **function** SAMPLENU(N)
6:     // Sample equation with no unary operator according to Equation 4
7:     **if** $N = 0$ **then**
8:         **return** one terminal symbol sampled from {'$x1$', '$x2$', '$x3$', '$x4$', 'const'}
9:     **end if**
10:     $weights \leftarrow \{\}$
11:     **for** $i = 0, \ldots, N - 2$ **do**
12:         $weights \leftarrow weights + \{F_i \cdot F_{N-i-2}\}$
13:     **end for**
14:     Sample i using SAMPLEBYWEIGHT($weights$)
15:     Sample $operator$ from {'+', '−', '×', '÷'}
16:     **return** SAMPLENU(i) + $operator$ + SAMPLENU($N - i - 2$)
17: **end function**
18: **function** SAMPLEU(N)
19:     // Sample equation with at least one unary operator according to Equation 5
20:     **if** SAMPLEBYWEIGHT($\{M_u F_{N-1}, G_N - M_u F_{N-1}\}$) $= 0$ **then**
21:         Sample $operator$ from {'sin', 'cos', 'log', 'exp'}
22:         **return** $operator$ + SAMPLENU($N - 1$)
23:     **else**
24:         Use the same method as SAMPLENU to sample a tree with binary operator as the root
25:     **end if**
26: **end function**
27: **if** SAMPLEBYWEIGHT($\{F_N, G_N\}$) $= 0$ **then**
28:     Sample equation Equ using SAMPLENU(N)
29: **else**
30:     Sample equation Equ using SAMPLEU(N)
31: **end if**
32: **Output:** The sampled equation Equ

Table 5: Detailed description of the SR benchmark datasets.

Benchmark	Expression	Range	Num. of samples
Nguyen-09	$\sin(x) + \sin(y^2)$	[-3, 3]	8000
Nguyen-10	$2\sin(x)\cos(y)$	[-3, 3]	8000
Nguyen-11	x^y	[1, 2]	8000
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	[-3, 3]	8000
Jin-1	$2.5x^4 - 1.3x^3 + 0.5y^2 - 1.7y$	[-3, 3]	8000
Jin-2	$8.0x^2 + 8.0y^3 - 15$	[-3, 3]	8000
Jin-3	$0.2x^3 + 1.5y^3 - 1.2y - 0.5x$	[-3, 3]	8000
Jin-4	$1.5\exp(x) + 0.5\cos(y)$	[-3, 3]	8000
Jin-5	$6.0\sin(x)\cos(y)$	[-3, 3]	8000
Jin-6	$1.35xy + 5.5\sin((x - 1.0)(y - 1.0))$	[-3, 3]	8000

trajectory is $t \in [0, 1]$ with a sampling time interval of 0.01. Namely, we sample 100 data points for each trajectory. For the repressilator, we generate 5000 trajectories using 5000 random initial conditions. They are split into 4000 training data and 1000 validation data respectively. The time span of each trajectory is $t \in [0, 4]$ with a sampling time interval of 0.01.

C   Experimental Setup for Different Methods

In this subsection, we present the experimental settings for the baselines. Following the prior work [29], for gplearn, we set the population to be 10000 and the number of generations to 50. For other baselines, SPL, NGGP, and DSR, we directly use their source code with the default parameters to implement experiments.

D   Computational Cost of Different Methods

We also compare the computational cost of the proposed SRCV and baseline approaches on Nguyen benchmark, as shown in Table 6. Our experimental results illustrate that our method, including DNNs and single-variable SR, have less running time than the baseline on Nguyen-12. However, it needs more running time than GP on Nguyen-09 and Nguyen-10, than DSR and NGGP on Nguyen-10 and Nguyen-11. However, our method has much higher recovery rate than DSR and NGGP, as illustrated in Table 1.

Table 6: Comparison of running time (seconds)

Benchmark	Expression	SRCV (ours)	SPL	NGGP	DSR	GP
Nguyen-09	$\sin(x) + \sin(y^2)$	697.00	2773.32	2551.89	1080.58	36.01
Nguyen-10	$2\sin(x)\cos(y)$	675.46	2447.58	383.27	89.40	10.46
Nguyen-11	x^y	655.60	4674.99	131.48	45.32	999.91
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	820.44	4932.86	3428.79	3156.63	1754.50

E   Discovered Governing Equations of Repressilator

Below, we present the discovered equations of the repressilator using our method. We can see that the equations are very close to the target model, except for $\alpha_0 = 10^{-5}$. The main reason is that α_0 is too tiny to be estimated. However, it does not impact the trajectory prediction too much, according to our results in Fig. 5.

$$\begin{cases} \dfrac{dM_{lacI}}{dt} = -M_{lacI} + \dfrac{9.939}{0.982 + P_{tetR}^3} \\[2mm] \dfrac{dM_{tetR}}{dt} = -M_{tetR} + \dfrac{10.338}{1.035 + P_{cI}^3} \\[2mm] \dfrac{dM_{cI}}{dt} = -M_{cI} + \dfrac{9.845}{0.987 + P_{lacI}^3} \\[2mm] \dfrac{dP_{cI}}{dt} = M_{lacI} - P_{cI} \\[2mm] \dfrac{dP_{lacI}}{dt} = M_{tetR} - P_{lacI} \\[2mm] \dfrac{dP_{tetR}}{dt} = M_{cI} - P_{tetR}, \end{cases} \tag{6}$$

F   Baselines on Gene Regulatory Networks

We also adopt the baseline methods to identify the governing equations of two gene regulatory networks. As illustrated in Table 7, we can observe that the governing equations uncovered by our SRCV method are close to the ground truth, while the baselines fail to discover governing equations from data. Note that we only list one representative equation in the following Table, but you can refer to the discovered equations of our method in Eqs.6 and 2 above. Thus, the proposed SRCV demonstrates the superior performance over the baselines in discovering symbolic equations.

G   Limitations and Future Work

The accuracy of our evaluation results is impacted by the accuracy of single-variable symbolic regression. In this work, we adopt the state-of-the-art MCTS method for symbolic regression. For future work, we need to develop new single-variable symbolic regression models to improve the accuracy. The accuracy of prediction results is also impacted by the number of training samples for DNNs. If the number of training data is limited, we need to explore a physics-enhanced neural symbolic regression model.

Table 7: The discovered results of different methods (one representative equation)

Task	Model	Discovered Equation
Toggle	Truth	$dU/dt = 4/(1 + V^3) - U$
	SRCV	$dU/dt = 3.919/(0.972 + V^3) - U$
	SPL	$dU/dt = -U + 2.539 \exp(\cos(V)) - \cos(V) - 1.749$
	NGGP	$dU/dt = U - V + \cos(V - \log(-1/(0.084U^2) + 0.084U \log(U) - 3.194)) + 0.067) + 3.544$
	DSR	$dU/dt = -U - V - sin(1.357V + 5.138) + 3.357$
	GP	$dU/dt = \sin(\exp(\exp(\exp(U))^2) - \exp(\exp(\sin(\exp(V))^2 \cdot \sin(\cos(U) - U - 7.252))\ldots$
Protein	Truth	$dM_{lacI}/dt = -M_{lacI} + 10/(1 + P_{tetR}^3) + 10^{-5}$
	SRCV	$dM_{lacI}/dt = -M_{lacI} + 9.939/(0.982 + P_{tetR}^3)$
	SPL	$dM_{lacI}/dt = 3.234 - 0.455M_{tetR}$
	NGGP	$dM_{lacI}/dt = M_{lacI}(-2P_{cI}P_{tetR} + P_{lacI} + 6.564)/(P_{cI} + 10.913)$
	DSR	$dM_{lacI}/dt = -1.582 + 3.482M_{tetR}^2/(P_{tetR}M_{tetR}^2 + 0.842)$
	GP	$dM_{lacI}/dt = 2.162/(M_{cI}^7 + M_{cI}P_{lacI})$

H Broader Impact

The goal of this work is to improve the accuracy and scalability of symbolic regression for scientific discovery. The proposed SRCV method has demonstrated superior performance over state-of-the-art methods in discovering analytical expressions from data, which can promote AI for scientific discovery. Note that this fundamental research will not cause any potential negative societal impacts.

I Computing Resources

We implement our experiments on the server with 1 A5000 GPUs with 24 GB graphics memory. The server has 32-Core 3.4 GHz AMD EPYC 7532 processors with 250GB RAM with 4TB SSD of storage capacity.