Differentiable Genetic Programming for High-dimensional Symbolic Regression

Xiaotian Song, Peng Zeng, Andrew Lensen, *Senior Member, IEEE,* Hengzhe Zhang, *Member, IEEE,* Yuwei Ou, Yanan Sun, *Senior Member, IEEE,* Mengjie Zhang, *Fellow, IEEE,* and Jiancheng Lv, *Senior Member, IEEE*

***Abstract*—Symbolic Regression (SR) aims to uncover hidden relationships within data by generating mathematical expressions, offering a pathway toward interpretable machine learning. Genetic Programming (GP) has traditionally dominated SR due to its flexibility in evolving expression trees. However, as the dimensionality of SR problems increases, the stochastic nature of GP leads to inefficiencies and poor scalability, particularly in high-dimensional real-world tasks. To address this, we propose a novel Differentiable Genetic Programming method, termed DGP, which enables gradient-based optimization for high-dimensional SR for the first time. Specifically, DGP proposes a differentiable symbolic tree that relaxes the discrete structure of GP trees to a continuous representation, thus allowing for efficient gradient-based optimization. To bridge the gap between this continuous space and valid symbolic expressions, DGP designs a sampling strategy that ensures structural validity and introduces a diversification mechanism to escape local optima and enhance global search. Extensive experiments on complex regression and expression recovery high-dimensional benchmarks demonstrate that DGP consistently outperforms state-of-the-art GP and neural network-based SR methods, achieving superior accuracy and expression recovery across a range of noise levels. For example, DGP achieves the best R^2 score of 0.32 on the DLBCL dataset with 7,400 dimensions, whereas most other methods yield nearly zero or even negative R^2 scores. The results demonstrate that DGP offers a reliable and robust solution for tackling complex high-dimensional SR problems, positioning it as a compelling tool for interpretable machine learning. The source code is available at https://github.com/songxt3/DGP.**

***Index Terms*—Symbolic regression, genetic programming, gradient descent, interpretable machine learning, neural networks.**

I. Introduction

In scientific research, a key task is to discover relationships within experimental data and then formalize them into symbolic expressions [1]. This process is widely performed using Symbolic Regression (SR) [2], a fundamental Machine Learning (ML) technique. Over the decades, two mainstream methods have emerged for SR: Neural Networks (NNs) [3] and Genetic Programming (GP) [4].

NN-based SR methods can be categorized into direct and indirect methods, depending on the symbolic expressions are generated directly from NNs or via intermediate expression trees. In direct methods, NNs with mathematical operators as activation functions are trained under sparsity constraints [5], and symbolic expressions are obtained by extracting operators associated with significant weights. For example, Martius *et al.* [6] and Sahoo *et al.* [7] use fully-connected NNs to learn the motion equations for a simple physical system. Kim *et al.* [8] design multiple NNs to address sequential SR tasks. Cranmer *et al.* [9] employ graph NNs [10] to simulate physical particle systems. In contrast, indirect NN-based SR methods employ generative models, e.g., variational autoencoder [11], Transformers [12], and Recurrent NNs (RNNs) [13], to construct expression trees. Then, the trees are converted to symbolic equations [14]–[16]. For instance, Kusner *et al.* [15] integrate variational autoencoders with grammar rules to generate syntactically valid expressions. Vastl *et al.* [17] employ Transformer architectures for SR tasks, while Petersen *et al.* [14] and Mundhenk *et al.* [16] use RNNs to generate expression trees. Furthermore, Kamienny *et al.* [18] leverage Transformer to guide the Monte-Carlo tree search procedure. This design can effectively balance exploration and exploitation to iteratively refine expressions in the discrete search space.

Although the above NN-based SR methods have shown effectiveness in experiments, severe limitations still remain. First, the performance of an NN depends highly on its manually designed architectures, including the number of layers, activation functions, and connectivity patterns. Making these architectural design decisions requires expertise in both NNs and the domain of the problem being solved [19]. Such combined expertise, however, is often not available [20]. Second, due to the non-convexity of NNs, these methods easily get stuck in local optima, resulting in poorly performing, inexact symbolic expressions. This phenomenon has been reported in both the direct and the indirect NN-based SR methods [21].

In contrast to NN-based methods, GP-based methods can automatically construct tree structures [4], which have three major advantages over NNs. First, GP explicitly generates symbolic expressions that can be directly analyzed, thereby providing stronger interpretability [22], [23]. Second, GP does not require to pre-determine the model structures, which can significantly reduce the trial-and-error process in model design [24], [25]. Third, GP can automatically identify relevant features during evolution, which can help reduce overfitting as feature number increases [25], [26]. These inherent structural and conceptual flexibilities make GP particularly well-suited for solving SR problems. In canonical GP-based SR, symbolic

This work was supported in part by the National Natural Science Foundation of China (No. 62276175). *(corresponding author: Yanan Sun.)*

X. Song, P. Zeng, Y. Ou, Y. Sun and J. Lv are with the College of Computer Science, Sichuan University, Chengdu 610064, China (e-mails: {songxt, zengpeng7, ouyuwei}@stu.scu.edu.cn, {ysun, lvjiancheng}@scu.edu.cn).

A. Lensen, H. Zhang, and M. Zhang are with the Evolutionary Computation Research Group, Victoria University of Wellington, Wellington 6140, New Zealand (e-mails:{andrew.lensen, hengzhe.zhang, mengjie.zhang}@ecs.vuw.ac.nz).

expressions are represented as trees, with variables as leaf nodes and operators as internal nodes. GP evolves a population of such trees across multiple generations using genetic operators, and the evolutionary process is guided by a predefined fitness function. Trees with higher fitness are selected as parents to produce the next generation, thereby directing the search towards optimal expressions. In recent years, various GP methods have been designed for SR problems [27]–[31]. Unfortunately, these methods are mainly designed for low-dimensional problems and cannot effectively scale well to high-dimensional problems[1] that are prevalent in our big data era [34]. Specifically, the population-based search process of GPs becomes inefficient as dimensionality grows [31]. Additionally, in high-dimensional SR problems, GP-based methods tend to generate long and nested expressions with redundant components, making them difficult to interpret [35], [36]. This is closely related to the phenomenon of "bloating" in GP, where trees grow unnecessarily large without a corresponding increase in fitness.

To address the problem about the efficiency, multiple works have been proposed. One popular category of methods utilizes Geometric Semantic GP (GSGP) [37] in SR [29], [38], [39]. Specifically, GSGP utilizes semantic-aware operators to enhance the variation process of GP by considering intermediate solution outputs, thereby making the search process more efficient. GSGP enables GP to find more accurate expressions, but often at the cost of complexity [40]. Furthermore, the expression size of GSGP-based SR varies from hundreds to millions of components [29], [38], which makes it almost impossible to understand. Researchers have also explored combining GP and feature engineering to improve the generalization performance of GP on high-dimensional problems [41], [42]. For instance, Chen et al. [41] try to improve the generalization performance of GP for high-dimensional SR by using feature selection. The new improvement experimentally alleviates the generalization performance degradation of GP on high-dimensional SR tasks. However, the inefficiency of GP evolutionary search method to high-dimensional problems, which causes the above problem from the root, remains to be addressed.

To overcome the "bloating" problem that impairs the interpretability of GP, researchers often incorporate a parsimony measure to the fitness function [43], [44] or utilize online program simplification [45]. For instance, Kinzett *et al.* [45] explore the use of two online program simplification methods in the evolution process: algebraic simplification [46] and numerical simplification. These help to reduce bloating in the evolution of GP. However, they need predefined parameters, i.e., simplification rules and a threshold. The final test performance is highly sensitive to these parameters, making it challenging to choose them appropriately. More recently, Franca *et al.* [44] introduce a new representation called interaction-transformation, which constrains the search space in order to exclude larger and more complicated expressions. The method has proven effective in traditional small-scale SR problems, but it does not scale well as problem dimensionality increases [36].

In summary, GP-based SR methods have the potential to be interpretable. However, the stochastic evolutionary-based search makes finding the optimal structure inefficient, resulting in performance degradation on high-dimensional problems. In contrast, NN-based SR methods have high search efficiency due to their use of gradient-based optimization — but the black-box nature of NNs makes it hard to convert an NN into an interpretable expression. In this paper, we propose, for the first time, **Differentiable GP (DGP)**, which is a gradient-based algorithm to effectively construct GP trees on high-dimensional SR problems. The main contributions of this paper are as follows:

- We propose a representation, the differentiable symbolic tree, in DGP that relaxes the discrete tree structure into a continuous space. This design allows the structure to be optimized using a gradient-based optimizer. As a result, DGP can more efficiently search for symbolic expressions with higher performance on high-dimensional problems.
- We introduce a diversification strategy that samples well-behaved symbolic trees near the optimized continuous space and uses them to generate novel trees from the symbolic expression search space. This design enhances structural diversity and provides broader global exploration of the global search space, effectively preventing DGP from becoming trapped in local optima.
- We demonstrate that the proposed DGP method outperforms existing SR methods on various high-dimensional SR benchmarks. Furthermore, we also demonstrate that DGP can recover the exact symbolic expressions on expression recovery datasets across different levels of noise. We conclude that DGP is an effective tool for challenging, real-world SR problems.

The remainder of this article is structured as follows. First, background and related works are presented in Section II. Then, DGP is detailed in Section III. Next, the experiment design and experiment results, as well as the analysis, are given in Sections IV and V. Finally, we conclude the paper and present our vision for future work in Section VI.

II. Background and Related Work

In this section, we introduce background knowledge in SR. We then review the relevant work that uses GP with gradient descent to demonstrate the novelty of DGP.

A. Symbolic Regression (SR)

SR is a sub-field of ML concerned with discovering an analytical model of the given data in the form of a symbolic expression or mathematical formula. In general, the training process of SR can be formalized as Equation (1):

$$f^* = \arg\min_f \sum_{i=1}^{n} \mathcal{L}(f(X_i), y_i) \tag{1}$$

where $X_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ are the input-output pairs with continuous values in a dataset with size n. $f(\cdot) : \mathbb{R}^d \to \mathbb{R}$

[1]We note that there is no formal definition of "high-dimensional" in the literature. Based on most existing SR works [14], [19], [30]–[32] focusing on problems with fewer than 50 features, and that literature specifically addressing high-dimensional SR typically utilizes benchmarks with 53 to 7,400 features [25], [26], [33], we consider, in this paper, that an SR problem is high-dimensional when its input has a dimensionality over 50.

represents a function in the form of a symbolic expression and $\mathcal{L}(\cdot,\cdot)$ is a predefined loss function. The goal of SR is to find a symbolic expression f^* that minimizes $\mathcal{L}(f(X_i), y_i)$ for $\{(X_i, y_i)|i = 1, 2, 3, \cdots, n\}$. After training, the expression f^* can be used to analyze the underlying relationships within the data and can also be used as a model to make predictions on unseen data.

SR has two significant differences from other regression techniques. Firstly, compared to statistical regression techniques, which only perform parameter optimization on a predefined fixed model structure, SR can adaptively discover a suitable model structure with appropriate parameters. For example, linear regression assumes a linear relationship between the input variables and the output by using linear equations, where the linear coefficients need to be optimized. However, SR does not pre-determine the structure of the expression, and the expressions obtained can be linear, polynomial, or of other shapes. This means that SR is a mixed optimization process, where the optimal structure and the parameters are searched at the same time. This advantage allows SR to have broader applications than statistical regression techniques, especially in solving unknown problems, where domain experience is often lacking, as would be needed to pre-design a model structure.

Secondly, other numerical regression methods, such as NNs, are generally treated as black-box models and only judged by their prediction accuracy on unseen data. SR, while producing accurate results, is also particularly well suited for human interpretability and in-depth analysis, as the expressions identify the true underlying functional relationship behind the data. For instance, given a dataset consisting of $x \in \mathbb{R}$ and $y \in \mathbb{R}$, if we use an NN for regression, we only get a black-box model for prediction after training. Instead, if we use SR to process it, we can obtain a symbolic expression, such as $y = -9.8 \times sin(x)$. We can not only make predictions but also reflect that x and y might be the angle and angular velocity taken from the same pendulum system, based on which performing further analysis. Given such advantages, SR is considered to be an effective way to enable interpretable ML [47].

B. Genetic Programming (GP) with Gradient Descent

GP is famous for its ability to automatically generate Lisp-style computer programs and was initially popularised by Koza [4]. GP can easily represent computer programs using hierarchical syntax trees, which means GP is suitable for automatic programming. With such tree structures, the size, shape, and contents of the program can be changed dynamically by genetic operators of GP. In SR, the symbolic solutions are similar to automatic programming, so GP was naturally extended to SR and gradually became the dominant technique among various SR techniques [48].

The idea of combining GP with gradient descent search to enhance GP for SR has been researched before. The initial works in this area were FGP [49] and WLGP [50]. FGP utilized a gradient descent search to optimize the numeric leaf values for GP and found that the gradient search can provide improvements in accuracy as a form of local search. In WLGP, inclusion factors are introduced to different subtrees of GP and the gradient descent search is utilized to optimize them, which collectively increases the overall performance. The gradient descent method in both FGP and WLGP was applied to every program in the population, which negatively impacts the efficiency of the search process [51]–performing gradient descent search for all the programs in the population comes at additional computation cost. Later, Chen *et al.* [52] focused on balancing accuracy and efficiency by applying gradient descent to only the top 20% programs in the population. In recent work, Dick *et al.* [53] found that feature standardization can improve the efficiency of gradient descent: they proposed a hybrid method that combined feature standardization and stochastic gradient descent. Moreover, EQL [6] employs a feed-forward network with specialized activation functions and sparsity regularization, using gradient descent to learn mathematical expressions. Similarly, KAN [54] utilizes gradient descent to optimize learnable activation functions on edges. This design can facilitate the discovery of compositional mathematical structures.

In summary, these existing methods only use gradient descent search as a complement to evolutionary methods for local search. However, in practice, the gradient descent method has already been proved to be effective in both parameter and structure optimization in the field of NN, therefore it is also worth exploring to utilize this efficient method to optimize the structure of GP. Unfortunately, the gradient descent method has never been used to dominate the optimization of the GP structure. This is because the optimization of GP structure is a fundamentally discrete problem. The first contribution of this study is just to address this critical issue.

III. The Proposed Method

In this section, we describe the proposed DGP method in detail. First, we present the overall framework of DGP in Subsection III-A. Then, we detail the optimization and diversification process, which are the key steps in DGP, in Subsections III-B and III-C, respectively.

A. Overall Framework

The overall framework of DGP is described in Algorithm 1. Given the maximum number of iterations G and the training dataset $\mathcal{D}$, DGP first initializes a population of symbolic expressions $\mathcal{P}_0$ (line 1). It then performs a series of alternating optimization and diversification iterations (lines 2-9). During each iteration, every expression in the current population undergoes a continuous gradient-based optimization to refine its structure and parameters (lines 4-6). Subsequently, a diversification step is applied to the optimized candidates to maintain population variance and explore new topological structures, yielding the next generation $\mathcal{P}_t$ (line 8). Finally, the best-performing expression f^* is selected from the final population $\mathcal{P}_G$ (line 10).

In the initialization process, an expression set τ_0, consisting of mathematical expressions, is initialized. Please note that there is no specific prior knowledge introduced to this initialization – τ_0 contains only the most basic mathematical expressions used by other GP tasks, such as x, $x + y$, etc.

Algorithm 1: Overall Framework of DGP

Input: The maximum number of generations G, the training dataset $\mathcal{D}$
Output: The optimal expression f^*

1 $\mathcal{P}_0 \leftarrow$ Randomly initialize a population of symbolic expressions
2 **for** $t = 1, \ldots, G$ **do**
3 $\mathcal{P}' \leftarrow \emptyset$
4 **for** $p \in \mathcal{P}_{t-1}$ **do**
5 $p' \leftarrow$ **GradientOptimization**(p, $\mathcal{D}$)
6 $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{p'\}$
7 **end**
8 $\mathcal{P}_t \leftarrow$ **Diversification**($\mathcal{P}'$, $\mathcal{D}$)
9 **end**
10 $f^* \leftarrow$ Select the best expression from $\mathcal{P}_G$
11 **Return** f^*

During each iteration t, an empty set τ is created to store the expressions found in that iteration (line 3). After that, the process of *Optimization*, i.e., the gradient-based optimization designed in DGP, is performed on D for each mathematical expression e in τ_{t-1}, and then the optimized e is added to τ (lines 4-6). Next, the process of *Diversification* is performed on the optimized expression set τ (line 8).

As shown above, the search process of DGP consists of two important parts: *Optimization* and *Diversification*, which are described in the following subsections.

B. Optimization

The optimization phase is proposed to search for the best structure in a subspace efficiently. The optimization process can be divided into three steps. They are: **Symbolic Tree Construction**, **Continuous Relaxation**, and the **Gradient-Based Optimizer**, which corresponds to Step ① to ③ in Fig. 1. In Step ①, the mathematical expression is represented by the symbolic tree. In Step ②, the symbolic tree is relaxed into a continuous model called a Differentiable Symbolic Tree (DST). In Step ③, DST is optimized with gradients iteratively, and the optimized DST is finally obtained.

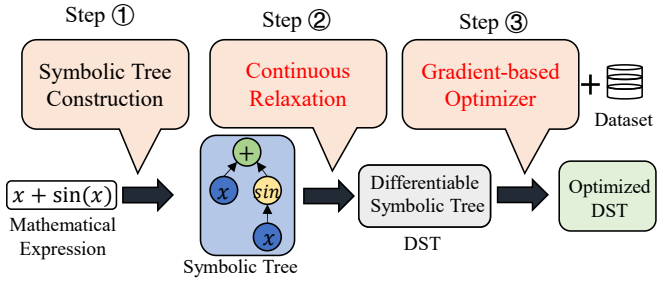


Fig. 1. Illustration of the optimization process, which can be divided into three steps. Step 1: Given an expression, a symbolic tree is constructed to represent it. Step 2: The symbolic tree is relaxed to a continuous DST model. Step 3: DST is then optimized by gradient descent.

Among the three steps, the first step follows the same process as other GP-based SR methods. The function set used is $\{+, -, \times, \div, sin, cos, exp, log\}$ and the terminal set is the input variables in the datasets, which together make up the primitive set. In the following subsections, we will focus on the last two steps.

1) Continuous Relaxation: In traditional GP-based SR, the search space of model structures is discrete. Specifically, a primitive in one node of a symbolic tree can randomly change to another through genetic operators in the evolutionary process. In DGP, we propose optimizing the structure of the symbolic trees by leveraging the gradient information, which requires a continuous search space. Therefore, *continuous relaxation* can construct a continuous subspace from a given tree structure. We achieve this through the use of a node matrix and adjacency matrix, which will be introduced below.

Node Matrix: assuming that the total number of nodes in the tree is K, the node matrix is composed of K vectors. The dimension of each vector is L, which is the length of the primitive set. The values in the vector are floats that represent the proportion of the operation or variable in the node, which can also be viewed as the probability of the node taking a certain operation or variable. As a result, the node matrix can be formulated by Definition 1.

Definition 1: **Node matrix N** is a $K \times L$ matrix that satisfies Equation (2):

$$\begin{cases} N = \begin{pmatrix} N_{nonleaf} \\ N_{leaf} \end{pmatrix} \in \mathbb{R}_+^{K \times L} \\ \quad = \begin{pmatrix} w_{0,0} & \ldots & w_{0,L-1} \\ \ldots & w_{i,j} & \ldots \\ w_{K-1,0} & \ldots & w_{K-1,L-1} \end{pmatrix} \\ s.t. \sum_{j=0}^{L-1} w_{i,j} = 1, i = 0, ..., K-1. \end{cases} \tag{2}$$

where $N_{nonleaf} \in R_+^{n \times L}$ and $N_{leaf} \in R_+^{m \times L}$ are for non-leaf nodes and leaf nodes of the symbolic tree, respectively. n and m are the numbers of non-leaf nodes and leaf nodes in the tree. Let $\hat{w_{ij}}$ be the original value in N. A softmax activation is applied to derive w_{ij}, i.e., $w_{ij} = exp(\hat{w_{ij}}) / \sum_{j'=1}^{L} exp(\hat{w_{i,j'}})$, to implicitly encode the constraint $\sum_{j=1}^{L} w_{ij} = 1$. This constraint guarantee each row of N is a valid primitives' distribution. $\hat{w}$ and the corresponding w are set to be learnable, which guarantees that the proportion of different primitives in each node in the tree can be changed by training. An example of the generation of N is illustrated in Fig. 2. As is shown in the figure, the symbolic tree has 4 nodes, numbered 0 to 3. Each node can be represented by a vector comprising all the primitives. Vectors 0 to 3 are used to represent nodes 0 to 3 correspondingly. In the example, the length of the primitive set is 9, so each vector contains 9 weights representing the proportion of each primitive, giving a 4×9 node matrix.

Adjacency Matrix: given the total number of nodes in the tree is K, the adjacency matrix is of shape $K \times K$. Each row and column in the matrix represent the adjacency information of a node. Specifically, each row in the matrix represents which nodes that the node points to, i.e., the parent node of it, and each column represents the child nodes of the current node. The values in the adjacency matrix indicate the strength of

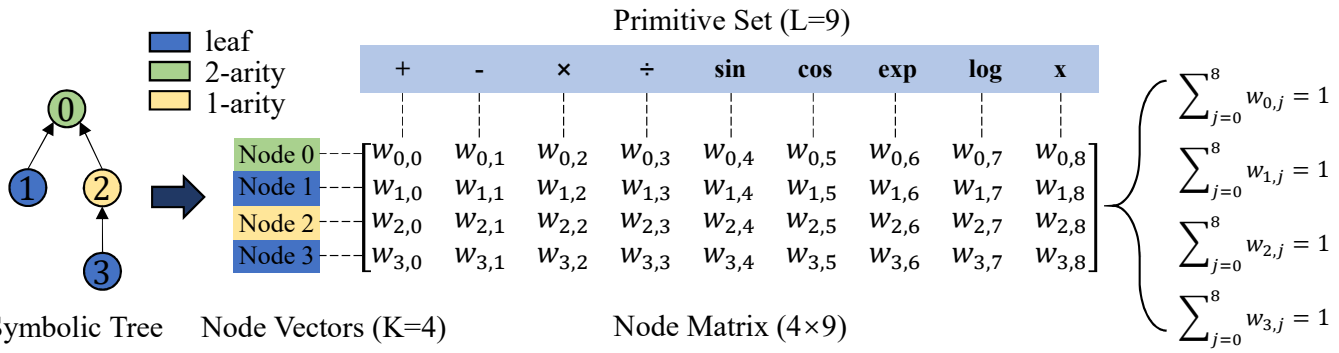


Fig. 2. Illustration of the generation of node matrix. For each node in the symbolic tree, a vector with the same size of the primitive set is initialized with zeros, and then the element is set to one if it is contained by the primitive set. These vectors together form the node matrix. During the GP structure optimization, the values of the elements in the node matrix are changed with the constraint that the sum of each vector is equal to one.

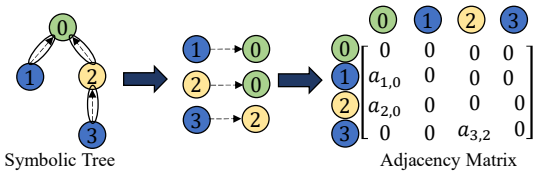


Fig. 3. Illustration of the adjacency matrix. $a_{i,j}$ represents that node i is the child of node j, and 0 means no connection.

the connection between the nodes, which will be used when a node is deleted or replaced in the sampling step. Formally, the adjacency matrix is defined by the Definition 2.

Definition 2: **Adjacency matrix A** is a $K \times K$ matrix that satisfies Equation (3):

$$\begin{cases} A \in \mathbb{R}^{K\times K} = \begin{pmatrix} a_{11} & \dots & a_{1K} \\ \dots & a_{ij} & \dots \\ a_{K1} & \dots & a_{KK} \end{pmatrix}, \\ where\ a_{ij} = \begin{cases} \sigma\left(v_{ij}\right) & , connected \\ 0 & , disconnected \end{cases} \end{cases} \tag{3}$$

where $v \in \mathbb{R}^K$ are the parameters, and the sigmoid function $\sigma(\cdot)$ imposes the constraint $0 \leq a_{ij} \leq 1$. For connected nodes i and j, $a_{ij} = \sigma(v_{ij})$ measures the strength of the connection. The illustration of the generation process of A is given in Fig. 3. As is shown in the figure, the symbolic tree has 4 nodes, numbered 0 to 3, thus its adjacency matrix is 4×4. The values in the matrix are initialized according to the connections in the tree. Specifically, there are 3 directed edges in the tree. They are node 1→node 0, node 2→node 0, and node 3→node 2. So there are 3 values greater than 0 in the matrix, which are $a_{1,0}$, $a_{2,0}$, and $a_{3,2}$, respectively. The rest of the values are 0, which means that there is no connection between these nodes.

Given the node matrix N and the adjacency matrix A, we can show how the fixed node in the original symbolic tree is relaxed to a continuous representation. Overall, there are two kinds of nodes (2-arity and 1-arity) in the symbolic tree, corresponding to two different situations:

1) For a 2-arity node: with the proposed N and A, a 2-arity node is transformed to a mixing node of all the operations. Specifically, the mixing node takes the outputs of two child nodes as inputs. For binary operations, such as $+$, $-$, the node calculates the results of each binary operation with the two inputs and then weights the results using the corresponding weights in N. For unary operations, such as sin, exp, they only need one input. Therefore, the child node with the greater adjacency weight in A would be chosen and the output of it is taken as the input. The results of each unary operation are calculated with the input and then weighted with the corresponding weights in N. Finally, the sum of all the results is returned as the output of this node.

2) For a 1-arity node: a 1-arity node is also transformed to a mixing node of all the operations. Specifically, a 1-arity node takes the only child node as the original input. For unary operations, the node calculates the results of each unary operation with the original input and returns the weighted sum. For binary operations, a leaf node would be added as the second input for the node, and the results of each binary operation are calculated using the two inputs. Through the above method, nodes in the tree are transformed into mixing nodes, allowing continuous representation of the symbolic tree to be obtained.

Note that our current implementation focuses on unary and binary operations for two reasons. First, this design aligns with the conventional settings of most existing SR methods [14], [17], [18], [54], [55]. Second, DGP can be extended to n-ary operations by easily modifying its initialization and selection mechanism. Specifically, we only need to expand the maximum arity of the initialized GP tree to n and accordingly extend the adjacency matrix A to reflect the connection relationships that support the selection mechanism of DST.

As previously discussed, bloating is a common problem in GP-based SR, which also occurs in the proposed method. In order to alleviate this issue, a new unary operation called $pass$ is introduced, to indicate that the input of a node is returned directly as the result. The effect of $pass$ is taken in conjunction with the *Diversification* part designed in the proposed method, and the details are justified in Subsection III-C.

To provide a clearer intuition of the above design, we give an example in Fig. 4 to illustrate how the original discrete nodes are relaxed into continuous mixing nodes. Specifically, DGP supports 9 operations

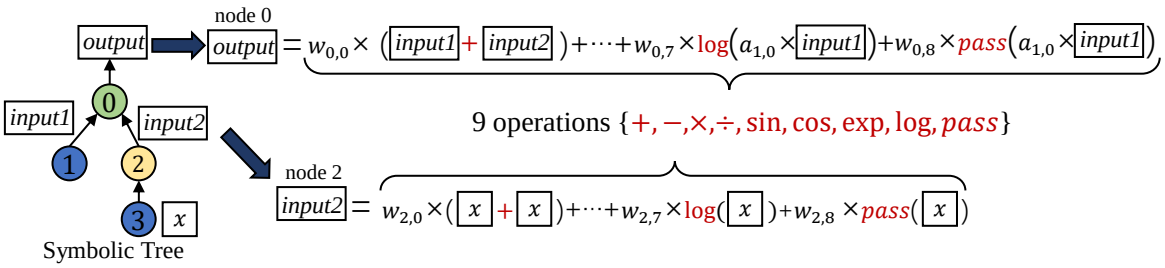


Fig. 4. An example to explain how the fixed node in the symbolic tree is relaxed to a continuous representation.

($\{+, -, \times, \div, \sin, \cos, \exp, \log, pass\}$). Each node now contains all 9 operations, weighted by a continuous probability distribution $w_{i,j}$ from the node matrix N. For example, "node 0" in Fig. 4 is originally a 2-arity node taking *input1* and *input2*. In the DST, its output becomes a weighted sum. For the binary "+" operation, it computes "*input1* + *input2*" and scales it by the probability $w_{0,0}$. For unary operations like $\log$ or *pass*, it selects the input with the higher adjacency weight, e.g., *input1* scaled by $a_{1,0}$ from adjacency matrix A, and weights the results by $w_{0,7}$ and $w_{0,8}$, respectively. Similarly, for 1-arity "node 2" with the x as input, it calculates "$x + x$" for the "+" operation and weights the result by $w_{2,0}$. For unary operations like $\log$ and *pass*, it directly applies them to x and scales the outcomes by $w_{2,7}$ and $w_{2,8}$. Through this relaxation mechanism, the discrete symbolic tree is transformed into a differentiable computation graph. Therefore, the DST operations and the network topology can be directly optimized by gradient descent.

Lastly, in order to take advantage of the previously learned information, N and A are initialized according to the original symbolic tree rather than through a random process. Taking the symbolic tree in Fig. 1 as an example, the primitive at the root node of the symbolic tree is $+$, and so the weight of $+$ is initialized to 1, with others set to 0 in the node vector of the root node when constructing N.

The symbolic tree, the node matrix N, and the adjacency matrix A together form the model called the DST. Through the proposed DST, the structure of a symbolic tree is represented by the corresponding real-valued matrices N and A, which can then be optimized using gradient-based optimization. In addition, the DST allows all features and operations to be selected within a fixed symbolic tree structure, which can maintain the diversity of feature selection and help DGP escape local optima.

2) Gradient-based Optimizer: After mapping the symbolic tree into a continuous space, a gradient-based method can be easily used to optimize the structure. Following the general SR objective defined in Equation (1), we approximate the search of the symbolic tree structure as a differentiable problem concerning the continuous matrices N and A, as shown in Equation (4):

$$N^*, A^* = \arg\min_{N,A} \sum_{i=1}^{n} \mathcal{L}(F_{N,A}(X_i), y_i) \tag{4}$$

where $F_{N,A}$ represents the continuous output of the DST model parameterized by the node matrix N and adjacency matrix A. By adjusting N and A, the DST is optimized in a continuous domain using gradient descent to make its output closely match the target y. While Equation (4) governs the continuous local optimization, the discrete space exploration relies on a genetic beam search. Specifically, in the proposed DGP method, only the best individual in each generation, i.e., with a beam width set to one, is further refined by gradient descent search. This design is inspired by the works of [56] and [57] and follows the core principle of beam search by concentrating on high-quality candidates while progressively pruning suboptimal solutions, thus improving search efficiency and stability. Moreover, this genetic beam search provides a crucial diversification mechanism to jump out of local optima, while its combination with gradient descent establishes a complementary strategy that seamlessly integrates global discrete exploration with local continuous optimization. This optimization method consists of three parts: forward propagation, loss function, and gradient descent optimization.

Forward propagation: As in a symbolic tree, the DST takes the input variables in the dataset as input and returns the computed result of the root node as output. The forward propagation process of DST consists of two parts: the mixing computation within nodes and the propagation of information between nodes.

For the mixing computation within nodes, let O be a set of candidate operations of the node, $O_j(\cdot)$ be the j-th operation to be applied to the input of the node $x^{(i)}$, $w_{i,j}$ be the weight of the i-th row and j-th column in N, and L be the length of O. In forward propagation process, the output of the i-th node $\bar{O}(x^{(i)})$ is a weighted sum over all possible operations, as shown in Equation (5).

$$\bar{O}(x^{(i)}) = \sum_{j=0}^{L-1} w_{i,j} O_j(x^{(i)}) \tag{5}$$

Through this equation, the output of a node in the DST is computed and then provided to the next node as an input.

The proposed DGP method utilizes the adjacency matrix to formally define the propagation of information between nodes, as detailed in Algorithm 2. The forward propagation employs a bottom-up execution strategy, utilizing an array R to store the intermediate computational results of all V nodes in the tree structure $\mathcal{T}$ (lines 1&2). Iterating in reverse topological order from the leaf nodes up to the root (line 3), the algorithm first retrieves the operation weight vector w_i for the current node i from the node matrix N (line 4). If node i is a leaf node, its

Algorithm 2: Forward Propagation Process of DST

Input: Input data X, node matrix N, adjacency matrix A, origin tree structure $\mathcal{T}$, operation set Op
Output: The predicted output of DST $\hat{y}$

1 $V \leftarrow$ Total number of nodes in $\mathcal{T}$
2 $R \leftarrow$ Array of size V to store intermediate node outputs
3 **for** $i = V - 1$ **to** 0 **do**
4 $w_i \leftarrow$ Get the operation weight vector from N for node i
5 **if** *node i is a leaf node* **then**
6 $R[i] \leftarrow$ Compute output using w_i, Op, and input data X via Equation (5)
7 **end**
8 **else**
9 $C(i) \leftarrow$ Identify the child nodes of node i from tree structure $\mathcal{T}$
10 $a_i \leftarrow$ Get the adjacency weights from A for edges connecting $C(i)$ to node i
11 $X^{(i)} \leftarrow$ Compute the weighted inputs from children: $\{a_{c,i} \cdot R[c] \mid c \in C(i)\}$
12 $R[i] \leftarrow$ Compute output of node i using w_i, Op, and $X^{(i)}$ via Equation (5)
13 **end**
14 **end**
15 $\hat{y} \leftarrow R[0]$
16 **Return** $\hat{y}$

inputs are drawn directly from the dataset X, and its output is computed via Equation (5) (lines 5&6). Conversely, for an internal node, the algorithm identifies its child nodes $C(i)$ from the fixed topological structure of $\mathcal{T}$ (line 9) and retrieves the corresponding adjacency weights a_i from A (line 10). The outputs of these child nodes, previously stored in $R[C(i)]$, are multiplied by their respective continuous adjacency weights to form the inputs $X^{(i)}$ for the current node (line 11), and its output is subsequently computed (line 12). Upon completing this traversal, the result at the root node (index 0) is returned as the final prediction $\hat{y}$ of the DST (lines 15&16).

Loss function: In GP-based SR, a standard fitness measure is Normalized Root-Mean-Square (NRMSE) [58] (where the standard RMSE is normalized by the standard deviation of the target values y). Given a dataset consisting of n groups of (X, y) pairs, NRMSE is computed as Equation (6):

$$NRMSE = \frac{1}{\sigma}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6}$$

where $\hat{y} = f(X)$ are the predicted values computed using the candidate expression f, and σ is the standard deviation of y. Normalization by σ makes the measure consistent across different datasets which have different domains. We use this standard measure as our loss function.

Given that we are performing SR, our DGP method needs to reacquire the discrete structure from the continuous encoding. However, there are possible issues when discretizing the continuous encodings. The weight vector of certain nodes may not clearly specify a single operation. For example, if the weight vector of a node is $[0.312, 0.135, ..., 0.315]$, it is hard to justify that an operation weighted by 0.315 is significantly better than the other weighted by 0.312. To address this, an extra loss is explicitly incorporated to push values in the node matrix towards 0 or 1 (as per [59]), formally as Equation (7):

$$Loss_{0-1} = -\frac{1}{L}\sum_{j=1}^{L}(w_j - 0.5)^2 \tag{7}$$

where L is the number of operations mixed in a node, and w is the softmax value of the weight of different operations. In order to control the strength of $Loss_{0-1}$, the loss is weighted by a coefficient λ_{0-1}. Thus, the final loss function is formulated as Equation (8):

$$Loss = NRMSE + \lambda_{0-1} Loss_{0-1} \tag{8}$$

Gradient descent optimization: The optimization process of the learnable parameters, i.e., N and A, follows the standard gradient-based training procedure. The Adam optimizer [60] is used to minimize the loss defined in Equation (8). The overall training process takes three steps:

Step 1: Sample a mini-batch of training data (X, y) from the dataset D;
Step 2: Perform forward propagation through DST to obtain predictions y_{pred}, and compute the loss between y and y_{pred} using Equation (8);
Step 3: Update the parameters in N and A via back-propagation using the Adam optimizer [60]. Repeat above steps until convergence.

The above three parts (**forward propagation**, **loss function**, and **gradient descent optimization**) together form the gradient-based optimizer. In each iteration, the expressions from the previous iteration are first transformed into symbolic trees and then relaxed to generate DSTs. Each DST is iteratively trained on the dataset until convergence. Then, the optimized DSTs go through a *Diversification* process, as introduced in the next subsection.

C. Diversification

This phase is proposed to drive the search from the optimized DST to the broader search space that can help DGP from getting trapped into local optima. Specifically, the goal is achieved by first sampling a set of well-behaved symbolic trees from the optimized DST. Then, a diversifier is applied to generate new symbolic trees, enabling a more thorough exploration of the global search space. In the following, these two steps are introduced in detail.

First, new symbolic trees are sampled from the optimized DST. The sampling step also uses a bottom-up approach. Starting from the leaf node, for each node in the DST, one of the primitives is selected with probability based on the optimized N. We develop three operations: SHIRINK, REPLACE, and EXPAND for the sampling step. Depending on the change of the primitive in the node, one of the three operations is chosen to perform on the node. The SHRINK operation is carried out if the arity of the node decreases; the

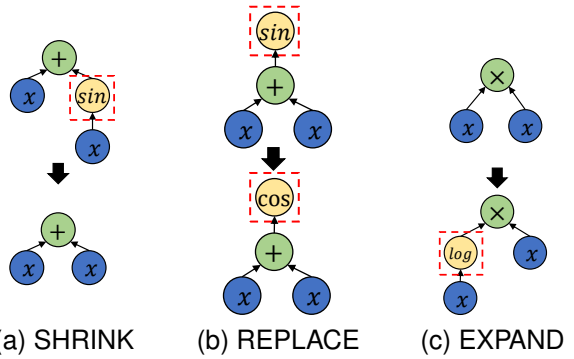


Fig. 5. Examples of the SHRINK, REPLACE, and EXPAND operations. The node(s) targeted by the corresponding operation has been surrounded with red dash rectangle for the purpose of illustration.

REPLACE operation is carried out if the arity is not changed; and the EXPAND operation is carried out if the arity increases. These operations are as follows.

- SHRINK: deleting the node. If the node is unary, the child node will now directly connect to the parent node. Otherwise, the child node with the greater adjacency weight will be chosen to connect to the parent node.
- REPLACE: replacing the node with a new function or terminal node, but not changing the number of arity.
- EXPAND: changing the type of the node, e.g., a leaf node to a 1-arity node or a 1-arity node to a 2-arity node, and connecting new children nodes to it.

Please note that the REPLACE operation is similar to the EXPAND operation, where each node can be changed by another node, no matter whether the node is a terminal or a function. They differ in that the REPLACE operation requires the number of arity unchanged, while the EXPAND operation does not. Fig. 5 illustrates the examples of these three operations. In particular, in Fig. 5 (a), the sin node is shrunk, and its child node is directly connected to its parent node; in Fig. 5 (b), the function sin in the root node is replaced by a new function cos; in Fig. 5 (c), the left child of the root node is expanded. The primitive in it is changed from x to log, and a leaf node is added to connect with it.

By performing the above three operations, we are able to sample well-behaved symbolic trees from the subspace of the symbolic tree. However, the sampled symbolic trees are only the local optimal solutions in subspace, and a mechanism is needed to guide the search towards potentially better subspaces. Inspired by recent work [16], we designed a diversifier to tackle the problem. The diversifier applies common genetic operators including crossover and mutation to the symbolic trees to generate more diverse structures. The evolutionary operations generates individuals that may fall well outside the local region of the search space and can effectively avoid the search process falling into a local optimum.

Finally, the new symbolic trees are evaluated on the training dataset, and their accuracy is computed. If their accuracy reaches a pre-defined threshold, we consider that we have found the correct mathematical expression, and the search is ended. Otherwise, the next iteration of optimization and diversification restarts based on the current symbolic trees. The above process continues until the maximum number of generations is reached, and then the best expression found is returned as the final result.

It is important to note that DGP is fundamentally a hybrid method that incorporates continuous gradient-based optimization and discrete GP optimization. Specifically, the differentiable symbolic tree structure is incorporated not to replace GP, but to provide a continuous, gradient-guided local search enhancement within the GP framework. While the gradient-based component might differentiate its appearance from traditional GP, its core search mechanism remains deeply rooted in GP. The defining rationale for classifying DGP within the GP paradigm lies directly in its search mechanism. Specifically, the search mechanism of DGP is driven by a population-based search strategy and explicitly employs traditional genetic operators (i.e., crossover and mutation operations) during its diversification stage. This design can effectively explore the discrete search space.

IV. Experiment Design

In this section, the experiment settings of DGP and others are given. The benchmark datasets are detailed in Subsection IV-A. The GP-based and NN-based SR recently-proposed baselines are introduced in Subsection IV-B. The evaluation metrics and parameter settings are shown in Subsection IV-C.

A. Benchmark Datasets

To verify if DGP can effectively and efficiently solve high-dimensional SR problems, we chose eight complex regression benchmarks, including four synthetic and four real-world datasets (listed in Table I) commonly used in the SR community [31], [36], [61]. As shown in Table I, these datasets span diverse domains and exhibit varying complexities, with the number of features ranging from 36 to over 7,400. By testing across the wide spectrum of dimensions, we can analyze the effectiveness of DGP against the growth of the task difficulty compared to baseline methods. The first seven datasets are taken from the Penn Machine Learning Benchmarks (PMLB) [62], which is a large collection of curated benchmark datasets for evaluating and comparing supervised ML algorithms. They are the regression benchmarks with the highest number of features in PMLB, ranging from tens to hundreds of features, which is challenging for both GP-based and NN-based SR methods. In order to investigate the effect of DGP on even higher-dimensional problems, we added the last benchmark, i.e., DLBCL [63], to the experiments. The DLBCL dataset is a diffuse large B-cell lymphomas dataset with over $7,000$ features, which is hundreds of times more than commonly-used regression datasets. Each dataset is divided into training and test sets at a ratio of 75% to 25%.

Another common practice in the SR community is to verify if the exact regression equations can be found by SR methods. Thus, we evaluate DGP and its competitors on six expression recovery SR benchmarks following [14], [16], [19]. The detailed configurations of these six benchmarks are

TABLE I
COMPLEX REGRESSION BENCHMARKS

Name	#Features	#Total	#Training	#Test
Satellite_image	36	6,435	4,826	1,609
Fri_c4_50	50	500	750	250
Fri_c0_50	50	250	187	63
Fri_c1_50	50	500	375	125
Fri_c4_100	100	1,000	750	250
GeoOriMusic	117	1,059	794	265
Tecator	124	240	180	60
DLBCL	7,400	240	180	60

TABLE II
EXPRESSION RECOVERY BENCHMARKS

Truth Expression	Denoted	Input Range
$sin(x^2)cos(x) - 1$	S1	$(-1, 1)$
$log(x + 1) + log(x^2 + 1)$	S2	$(0, 2)$
$x^3 + x^2 + x + sin(x) + sin(x^2)$	S3	$(-1, 1)$
$sin(x) + sin(y^2)$	S4	$(0, 1)$
$\frac{x^4}{x+y}$	S5	$(-1, 1)$
$4 \times sin(x)cos(y)$	S6	$(0, 1)$

summarized in Table II. As presented in the table, these expression recovery tasks feature a variety of functional structures, including polynomial, rational, and trigonometric operations, paired with different input domains. This design aims to systematically assess and conclude whether the algorithms possess the capability to recover mathematical relationships rather than merely approximating numerical outputs. The data points are uniformly sampled 20 times, as in previous papers [14], [16].

B. Peer Competitors

We choose a range of high-performing SR methods as the peer competitors in our experiments. Specifically, the classic GP methods, including standard GP [64], NSGP [30], and MGPRC [31], are selected to justify the superiority of DGP on both performance and program size. Moreover, the recent gradient-based GP methods, like GP-gradient [50], GPPDE [51], GP-L [52], and GPZGD [53], are used to further evaluate the performance of DGP. Furthermore, the NN-based methods, including DSR [14], PSSR [16], and Symformer [17], are also incorporated to verify the effectiveness of DGP. Finally, the best-performing methods from SRBench, i.e., GPGOMEA [65], Operon [66], PSTree [67], PySR [55], FEAT [68], SBP-GP [69], and DSR [14], are utilized as the peer competitors on the SRBench benchmark.

C. Evaluation Metrics and Parameter Settings

In order to compare the performance of different methods on complex regression and expression recovery benchmarks, different evaluation metrics are used. For complex regression benchmarks, we follow the commonly evaluation metric [61] and compare the coefficient of determination R^2 between y and $\hat{y}$, excluding NaNs and $\pm\infty$, as follows:

$$R^2 = 1 - \frac{\sum_i^N (y_i - \hat{y}_i)^2}{\sum_i^N (y_i - \bar{y}_i)^2} \tag{9}$$

TABLE III
HYPERPARAMETER SETTINGS FOR THE EXPERIMENTS.

Hyperparameter	**Value**
Common parameters	
Function set	$+$, $-$, $\times$, $\div$, sin, cos, exp, log
Evaluation budget	100k evaluations† (30 repeated trials†)
Parameters of the proposed DGP method	
Optimizer	Adam ($lr = 0.005$)
Loss function	NRMSE + $\lambda_{0-1} \times Loss_{0-1}$ ($\lambda_{0-1} = 0.1$)
Population size	500†
Evolution rates	Crossover: 0.5, Mutation: 0.5
Iteration schedule	1,000 epochs/iter, 20 generations/iter
Parameters of NN-based methods	
Optimizer	Adam ($lr = 0.0025$)
RNN cell (Layers / Size)	LSTM (1 layer / 32 units)
Training method	RSPG (Entropy weight: 0.005)
Expression length	$[4, 30]$
Reward / fitness function	NRMSE
Parameters of GP-based methods	
Population & Initialization	500†, Ramped half-and-half
Crossover	One point ($p = 0.5$)
Mutation	Uniform ($p = 0.5$)
Selection	Tournament (Size: 3)
Mutate tree depth	Range: $[0, 2]$

† The termination criteria, repeated trial numbers, and the population size for expression recovery tasks are 500k, 10, and 1,000, respectively.

The R^2 metric provides an intuitive measure of how well the discovered symbolic expressions capture the variance in the data [70]. According to Equation (9), the R^2 score can be negative. The formula directly subtracts the ratio of two error sums from 1. The numerator $\sum_i^N (yi - \hat{y}_i)^2$ represents the sum of squared residuals of the model's predictions, while the denominator $\sum_i^N (y_i - \bar{y}_i)^2$ represents the error of a naive baseline predicting the mean. The maximum possible score is 1.0, achieved when the model perfectly fits the data, i.e., the numerator is 0. However, if a model's predictions are so inaccurate that its residual error strictly exceeds the baseline error of simply predicting the mean, the fraction becomes strictly greater than 1, yielding a negative R^2. In statistical perspective, this behavior can arise from several conditions, such as omitting the intercept term in linear regression, fitting an inappropriate non-linear function, or evaluating the model on unseen data. In our experiments, the negative score is primarily caused by overfitting. In this paper, the R^2 scores are calculated directly using the official implementation of the *scikit-learn* library.

Additionally, following SRBench [71], [72], we also report Mean Squared Error (MSE), Mean Absolute Error (MAE), model size, and runtime in seconds. For the expression recovery tasks, we further compared the recovery rates [14] between different methods. Recovery rate is defined as the number of runs that successfully recovered the expression from the data as a percentage of the total number of runs. The metric can effectively eliminate the uncertainty of performance evaluation brought by randomness.

In the experiments, every method is trained on each dataset across 30 repeated trials with a different random seed that controls both the train/test split. The parameter settings of the experiments are in Table III. As shown in the table, we clearly list the specific values for structural hyperparameters (e.g., maximum depth and number of variables), evolutionary

TABLE IV
TEST R^2 SCORE OF DGP AND OTHER METHODS ON THE EIGHT COMPLEX REGRESSION BENCHMARKS OVER 30 INDEPENDENT RUNS

Datasets Dimensions	Satellite_image 36	Fri_c4_50 50	Fri_c0_50 50	Fri_c1_50 50	Fri_c4_100 100	GeoOriMusic 117	Tecator 124	DLBCL 7,400
GP [64]	$0.34^{\pm 0.203}$	$0.57^{\pm 0.249}$	$0.58^{\pm 0.164}$	$0.45^{\pm 0.283}$	$0.32^{\pm 0.281}$	$0.54^{\pm 0.044}$	$0.87^{\pm 0.043}$	$0.06^{\pm 0.177}$
NSGP [30]	$0.45^{\pm 0.083}$	$0.55^{\pm 0.434}$	$0.69^{\pm 0.219}$	$0.58^{\pm 0.307}$	$0.24^{\pm 0.586}$	$0.51^{\pm 0.240}$	$0.82^{\pm 0.174}$	$-0.84^{\pm 0.706}$
MGPRC [31]	$0.61^{\pm 0.101}$	$0.47^{\pm 0.166}$	$0.62^{\pm 0.104}$	$0.66^{\pm 0.087}$	$0.43^{\pm 0.156}$	$0.48^{\pm 0.109}$	$0.87^{\pm 0.067}$	$0.30^{\pm 0.131}$
DSR [14]	$-0.39^{\pm 0.831}$	$-2.11^{\pm 1.479}$	$-0.46^{\pm 0.625}$	$-1.13^{\pm 1.644}$	$-3.13^{\pm 0.767}$	$0.12^{\pm 0.245}$	$-0.39^{\pm 1.031}$	$-349.84^{\pm 0.000}$
PSSR [16]	$-0.10^{\pm 0.433}$	$-0.36^{\pm 0.931}$	$0.13^{\pm 0.518}$	$0.00^{\pm 0.178}$	$-1.19^{\pm 1.149}$	$0.27^{\pm 0.243}$	$0.61^{\pm 0.318}$	$-58.68^{\pm 115.657}$
GP-gradient [50]	$\mathbf{0.73}^{\pm 0.001}$	$-1.74^{\pm 88.171}$	$-0.65^{\pm 15.080}$	$-0.59^{\pm 16.157}$	$0.03^{\pm 0.298}$	$0.48^{\pm 0.015}$	$0.91^{\pm 0.009}$	$0.00^{\pm 0.200}$
GPPDE [51]	$0.54^{\pm 0.099}$	$0.15^{\pm 0.096}$	$0.31^{\pm 0.135}$	$0.11^{\pm 0.105}$	$0.15^{\pm 0.058}$	$0.50^{\pm 0.077}$	$0.31^{\pm 0.254}$	$0.07^{\pm 0.100}$
GP-L [52]	$0.22^{\pm 0.015}$	$-0.03^{\pm 0.033}$	$0.11^{\pm 0.032}$	$0.04^{\pm 0.004}$	$0.06^{\pm 0.006}$	$0.30^{\pm 0.018}$	$0.29^{\pm 0.039}$	$0.07^{\pm 0.010}$
GPZGD [53]	$\mathbf{0.79}^{\pm 0.00}$	$0.35^{\pm 0.109}$	$0.23^{\pm 0.184}$	$0.38^{\pm 0.142}$	$0.03^{\pm 0.298}$	$0.46^{\pm 0.113}$	$\mathbf{0.94}^{\pm 0.038}$	$-0.11^{\pm 0.141}$
DGP (Ours)	$\mathbf{0.63}^{\pm 0.118}$	$\mathbf{0.65}^{\pm 0.144}$	$\mathbf{0.72}^{\pm 0.069}$	$\mathbf{0.67}^{\pm 0.087}$	$\mathbf{0.58}^{\pm 0.147}$	$\mathbf{0.62}^{\pm 0.066}$	$0.88^{\pm 0.017}$	$\mathbf{0.32}^{\pm 0.127}$

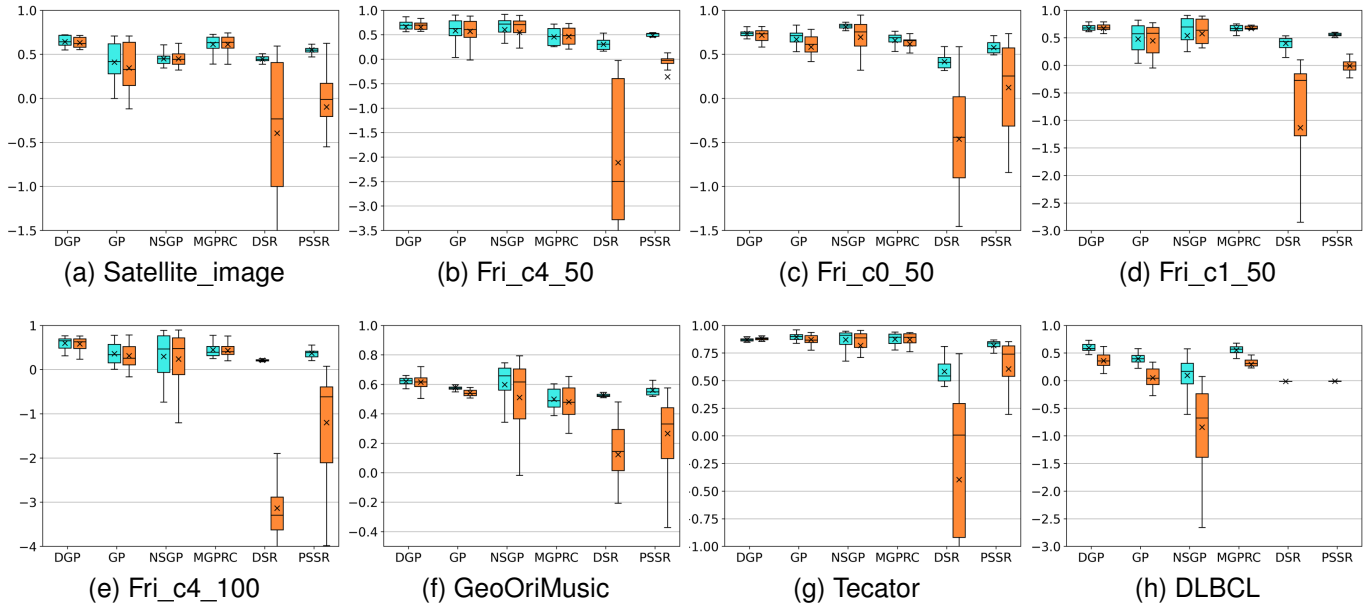


Fig. 6. Box plots on the training and test performance measured by R^2 of the found mathematical expressions by the proposed DGP method against peer competitors on the benchmark datasets. Blue and orange boxes represent the training dataset and the test dataset, respectively.

parameters (e.g., population size and mutation rates), and optimization coefficients (e.g., learning rate and λ_{0-1}). Given that it is difficult to establish an exact correspondence between GP-based and NN-based methods, e.g., how many training epochs in NN-based methods correspond to one evolution generation in GP, we follow the settings in [61] and uniformly set the termination criteria as the number of evaluations performed. We set the maximum number of evaluations to 100k for complex regression benchmarks and 500k for expression recovery benchmarks.

V. RESULTS AND DISCUSSION

The experiments in this section are divided into two groups: complex regression and expression recovery benchmarks. On the complex regression benchmarks, we measure training and generalization performance [31], [36], [39]. On the expression recovery benchmarks, following previous research [14], [16], we perform experiments to determine the recovery rate and the effect of noise. As discussed in Section I, existing GP methods suffer from the problem of bloat on SR problems, resulting in very large and un-interpretable expressions. To compare the ability of the proposed method to address this issue, we also perform program size analysis of the final expressions across different methods in both complex regression and expression recovery experiments.

A. Results on the Complex Regression Benchmarks

Table IV comprehensively reports the averages and standard deviations of the test R^2 scores achieved by DGP and the baseline methods across the eight complex regression benchmarks over 30 independent runs. Higher R^2 values indicate superior predictive accuracy. The distribution of R^2 scores on the training datasets and test datasets are shown in Fig. 6. In the figure, each method has two boxes on each dataset. Blue and orange boxes are for the training dataset and the test dataset, respectively. Results of the statistical significance tests of R^2 scores between DGP and other baseline methods are presented in Table V. In the table, "$-$", "$+$", and "$=$" indicate DGP performs significantly better, worse, or with no significant difference against the compared method, respectively.

TABLE V
RESULTS OF STATISTICAL SIGNIFICANCE TESTS

Datasets	DGP(training, test)vs.				
	GP	NSGP	MGPRC	DSR	PSSR
Satellite_image	$(-,-)$	$(-,-)$	$(=,=)$	$(-,-)$	$(-,-)$
Fri_c4_50	$(=,=)$	$(=,=)$	$(-,-)$	$(-,-)$	$(-,-)$
Fri_c0_50	$(-,-)$	$(+,=)$	$(-,-)$	$(-,-)$	$(-,-)$
Fri_c1_50	$(-,-)$	$(=,=)$	$(=,=)$	$(-,-)$	$(-,-)$
Fri_c4_100	$(-,-)$	$(-,-)$	$(-,-)$	$(-,-)$	$(-,-)$
GeoOriMusic	$(=,=)$	$(=,-)$	$(-,-)$	$(-,-)$	$(-,-)$
Tecator	$(-,=)$	$(=,=)$	$(=,=)$	$(-,-)$	$(-,-)$
DLBCL	$(-,-)$	$(-,-)$	$(=,=)$	$(-,-)$	$(-,-)$

TABLE VI
PROGRAM SIZE ANALYSIS ON THE COMPLEX REGRESSION BENCHMARKS

Benchmark	Size (#Node of the best models)					
	DGP	GP	NSGP	MGPRC	DSR	PSSR
Satellite_image	$61^{\pm 28.9}$	$136^{\pm 63.2}$	$65^{\pm 19.7}$	$64^{\pm 21.5}$	$10^{\pm 2.3}$	$40^{\pm 42.2}$
Fri_c4_50	$37^{\pm 12.0}$	$70^{\pm 24.8}$	$72^{\pm 17.8}$	$42^{\pm 15.6}$	$8^{\pm 2.8}$	$16^{\pm 16.5}$
Fri_c0_50	$26^{\pm 14.1}$	$76^{\pm 39.9}$	$86^{\pm 11.7}$	$32^{\pm 8.9}$	$8^{\pm 2.1}$	$17^{\pm 12.7}$
Fri_c1_50	$34^{\pm 15.5}$	$65^{\pm 40.1}$	$80^{\pm 19.4}$	$34^{\pm 12.0}$	$7^{\pm 2.7}$	$15^{\pm 9.2}$
Fri_c4_100	$29^{\pm 14.8}$	$71^{\pm 43.5}$	$78^{\pm 13.8}$	$44^{\pm 17.1}$	$8^{\pm 1.4}$	$20^{\pm 22.1}$
GeoOriMusic	$39^{\pm 25.0}$	$13^{\pm 27.8}$	$75^{\pm 17.3}$	$10^{\pm 12.9}$	$8^{\pm 1.6}$	$16^{\pm 7.9}$
Tecator	$39^{\pm 21.8}$	$95^{\pm 43.8}$	$53^{\pm 18.1}$	$65^{\pm 21.3}$	$8^{\pm 2.0}$	$28^{\pm 23.9}$
DLBCL	$47^{\pm 21.6}$	$36^{\pm 16.4}$	$76^{\pm 18.6}$	$33^{\pm 13.6}$	$7^{\pm 0.0}$	$5^{\pm 0.4}$

TABLE VII
RESULTS OF STATISTICAL SIGNIFICANCE TESTS ON THE PROGRAM SIZE

Benchmark	DGP vs.				
	GP	NSGP	MGPRC	DSR	PSSR
Satellite_image	-	=	=	+	=
Fri_c4_50	-	-	-	+	+
Fri_c0_50	-	-	=	+	=
Fri_c1_50	-	-	=	+	+
Fri_c4_100	-	-	-	+	=
GeoOriMusic	+	-	+	+	+
Tecator	-	-	-	+	=
DLBCL	=	-	+	+	+

We discuss the training performance and the generalization performance in Subsections V-A1 and V-A2. The program size of the final expressions is compared and analyzed in Subsection V-A3

1) Training Performance: As shown in Fig. 6, GP-based methods achieve higher training performance than NN-based methods. However, the narrower R^2 distributions observed from NN-based methods suggest greater stability across different runs. This highlights a key trade-off, i.e., while GP-based methods may find better-performing expressions, they are also more sensitive to stochasticity and require multiple runs to reach optimal solutions in practice. As shown in Table V, DGP consistently achieves better performance than other methods, especially on high-dimensional datasets, such as DLBCL. The data reveals that DGP achieves a strictly superior $(-)$ or equivalent $(=)$ statistical performance against almost all baselines across both training and test sets. Notably, compared with recent NN-based methods like DSR and PSSR, DGP demonstrates significantly better performance on the most challenging dimensions. This suggests that the gradient-based optimization of DGP both improves performance and consistency on high-dimensional problems.

2) Generalization Performance: As presented in Fig. 6, GP-based SR methods exhibit strong generalization, with test performance closely aligned to training results. In contrast, NN-based methods suffer from significant generalization gaps, especially on high-dimensional datasets such as DLBCL. This suggests that, with comparable evaluation budgets, GP-based methods are more robust and generalize better than NN-based methods in high-dimensional settings. This is supported by by the results shown in Table IV, where DGP consistently achieves a higher R^2 test score than the others. Specifically, on the high-dimensional DLBCL dataset (7,400 features), DGP attains an R^2 of 0.32, whereas baseline methods like DSR collapse to -349.84. To further compare the performance of DGP with recent gradient-based GP methods, we incorporate GP-gradient, GPPDE, GP-L, and GPZGD as the peer competitors and achieve comparable or better R^2 scores. Notably, on the DLBCL dataset with 7,400 features, DGP achieves an R^2 of 0.32, clearly outperforming all other baselines. Additionally, as indicated in Table V, the R^2 score of DGP is significantly better than DSR and PSSR on all eight test datasets. Given the importance of generalization performance on the test dataset of evaluating symbolic regression methods, these results establish DGP as the most effective method among those competitors.

Note that the great performance of DGP on the DLBCL dataset stems from its efficient search strategy and ability to escape from local optima. Specifically, by presenting gradient-based optimization tailored to discrete GP trees, DGP can effectively explore high-quality regions of the search space. Furthermore, the diversification mechanism can prevent premature convergence and promote solution diversity, naturally enhancing generalization in high-dimensional settings. Similarly, MGPRC benefits from a multi-objective evolutionary search strategy that jointly optimizes accuracy and Rademacher complexity, which get a balance between model fit and generalization. The use of a behavior-based complexity measure helps avoid overly complex or overfit models, making MGPRC perform as well as the proposed DGP method on the high-dimensional datasets like DLBCL. More specifically, we have two perspectives as to why our proposed DGP method outperforms the competitors. First, the gradient descent used in DGP allows it to keep continuously searching for better solutions in a relatively small search space, improving search efficiency. Second, from another point of view, the diversifier helps by providing fresh new samples to escape local optima. This is because the gradient-based discrete distributions often concentrate their probability mass on a relatively small portion of the search space, resulting in premature convergence to locally optimal solutions. However, the genetic operations in the diversifier generate new individuals which may fall well outside the concentrated region of the search space. Combining the gradient-based optimizer and the diversifier, DGP is able to regress better solutions more efficiently.

3) Program Size Analysis: Table VI reports the program size, i.e., the structural complexity, of the mathematical expressions found by all evaluated methods on the complex regression benchmarks. As shown in the table, NN-based methods, particularly DSR, tend to produce highly compact

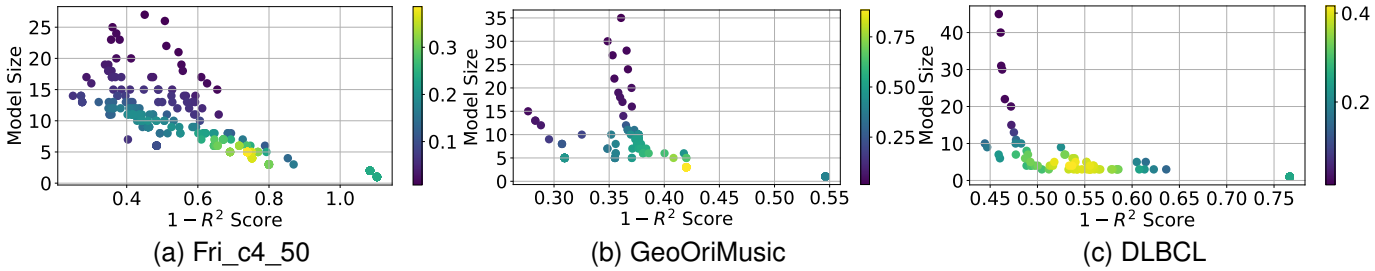


Fig. 7. The multi-objective evaluation results between model size and R^2 score.

TABLE VIII
COMPARISON WITH DGP AND OTHER METHODS ON SRBENCH. THE DIMENSION OF ALL SELECTED DATASETS ARE HIGHER THAN OR EQUAL TO 50.

	Fri Series (13 Datasets)					GeoOriMusic					Tecator				
	R^2	MSE	MAE	Size	Time (s)	R^2	MSE	MAE	Size	Time (s)	R^2	MSE	MAE	Size	Time (s)
GPGOMEA [65]	0.79	0.21	0.37	18	6,689	0.74	0.24	0.37	21	20,789	0.99	2.32	1.04	17.5	22,476
Operon [66]	0.96	0.03	0.15	72.7	852	0.78	0.20	0.34	51	939	0.99	0.56	0.53	51	326
PSTree* [67]	0.97	0.03	0.14	593.8	107	0.76	0.23	0.36	511.3	136	0.99	0.87	0.64	349.1	95
PySR* [55]	0.79	0.20	0.35	17.2	1,578	0.63	0.36	0.44	5.6	2,094	0.99	2.28	1.02	11.7	2,133
FEAT [68]	0.89	0.11	0.25	117.8	10,684	0.77	0.21	0.35	57.5	4,674	0.99	0.95	0.67	63.5	6,195
SBP-GP [69]	0.92	0.08	0.21	922	199,798	0.74	0.26	0.37	999.5	234,138	0.99	0.79	0.58	788.5	130,331
DSR [14]	0.52	0.48	0.55	7.1	32,615	0.63	0.37	0.46	5	34,159	0.98	3.51	1.38	5	36,123
DGP (Ours)	0.83	0.17	0.31	115.0	2,051	0.74	0.20	0.34	98.4	3,026	0.99	2.01	0.99	22.8	1,249

* reproduced with open-source code. Note that the iterations of PySR are reduced to 1,000 to constrain the search time.

models (model size $<$ 10), while standard GP often produces excessively large expressions due to structural bloating. The proposed DGP method yields concise models that rival the compactness of NN-based methods while avoiding the bloat of traditional GP. Furthermore, Table VII presents the statistical significance tests comparing the program size of DGP against the competitors. Similar to the performance analysis, we utilize "$-$", "$+$", and "$=$" to denote whether DGP's program size is significantly smaller (better), larger (worse), or comparable to the baselines. As shown in the table, DGP achieves significantly smaller ($-$) or equivalent ($=$) program sizes compared with traditional GP-based methods (e.g., GP and NSGP) across almost all benchmarks, while yielding moderately larger ($+$) expressions than the heavily compressed NN-based methods like DSR and PSSR. Note that DGP can produce the most compact solutions yet with better performance on the test datasets. Given that expression size directly affects interpretability, these results indicate that DGP achieves a balance between model size and performance.

4) Multi-objective Analysis: Since predictive performance and model size typically exhibit a trade-off, analyzing the conflict between these objectives is crucial for designing models that are both accurate and efficient. Therefore, we perform an additional multi-objective optimization experiment using NSGA-II [73]. This allows us to identify Pareto-optimal solutions and evaluate the conflict between the two objectives. As shown in Fig. 7, we visualize the Pareto front across three different datasets after 30 iterations. More specifically, on the Fri_c4_50 dataset, the Pareto front exhibits a clear negative correlation, i.e., as $1 - R^2$ decreases, the model size tends to shrink significantly. This indicates substantial model compression potential on this dataset, where performance can be maintained or even improved while reducing complexity. For the GeoOriMusic dataset, the Pareto front is sparser and more concentrated, mainly within the range $1 - R^2 \in [0.35, 0.45]$. In the DLBCL dataset, the Pareto front exhibits a typical "L-shaped" curve. Initially, the model size decreases quickly as performance improves, but further enhancement in R^2 requires a steep increase in model complexity. In summary, these results demonstrate that DGP can be effectively integrated with existing multi-objective optimization algorithms, yielding solutions with both high performance and small model size.

5) Comparison on SRBench: To further justify the effectiveness of the proposed DGP method, we also conduct the experiments on high-dimensional SRBench [71], [72] datasets (dimension $\geq$ 50) and compare the proposed DGP method against seven best-performing methods from SRBench (i.e., GPGOMEA [65], Operon [66], PSTree [67], PySR [55], FEAT [68], SBP-GP [69], and DSR [14]), and the results are presented in Table VIII. As can be observed from Table VIII, DGP achieves highly competitive performance in terms of accuracy (R^2 score, MSE, and MAE) while offering relatively small model sizes and moderate search times. Specifically, DGP produces significantly more compact models than PSTree and SBP-GP, with an average model size of 115 compared to 485$\sim$900 on the Fri dataset series. It also requires substantially less computation time than GPGOMEA and SBP-GP, taking 1,296 seconds against 13,033$\sim$22,467 seconds on the Tecator dataset. Moreover, compared with the structurally similar gradient-based method DSR, the proposed DGP method demonstrates significantly higher performance and efficiency across all datasets. For instance, on GeoOriMusic, DGP attains an R^2 score of 0.74, an MSE of 0.20, and an MAE of 0.34, compared to the 0.63, 0.37, and 0.45 obtained by DSR,

TABLE IX
RECOVERY RATES OF DGP AND COMPETITORS ON THE EXPRESSION RECOVERY BENCHMARKS

Bench-mark	Recovery rate (%)												
	DGP	GP	NSGP	MGPRC	DSR	PSSR	Symformer	PSTree	Operon	GPGOMEA	PySR	SBP-GP	FEAT
S1	90	30	60	40	0	80	50	80	100	100	80	90	90
S2	80	0	10	0	0	80	60	0	20	0	10	0	50
S3	50	0	0	40	0	50	0	20	20	10	0	0	70
S4	100	50	100	100	100	100	0	30	0	20	100	20	60
S5	100	0	0	90	70	90	70	0	0	0	0	0	0
S6	20	0	0	0	20	10	10	20	40	60	90	60	30
Avg.	73.3	13.3	28.3	45.0	31.7	68.3	31.7	25	30	31.7	46.7	28.3	50.0

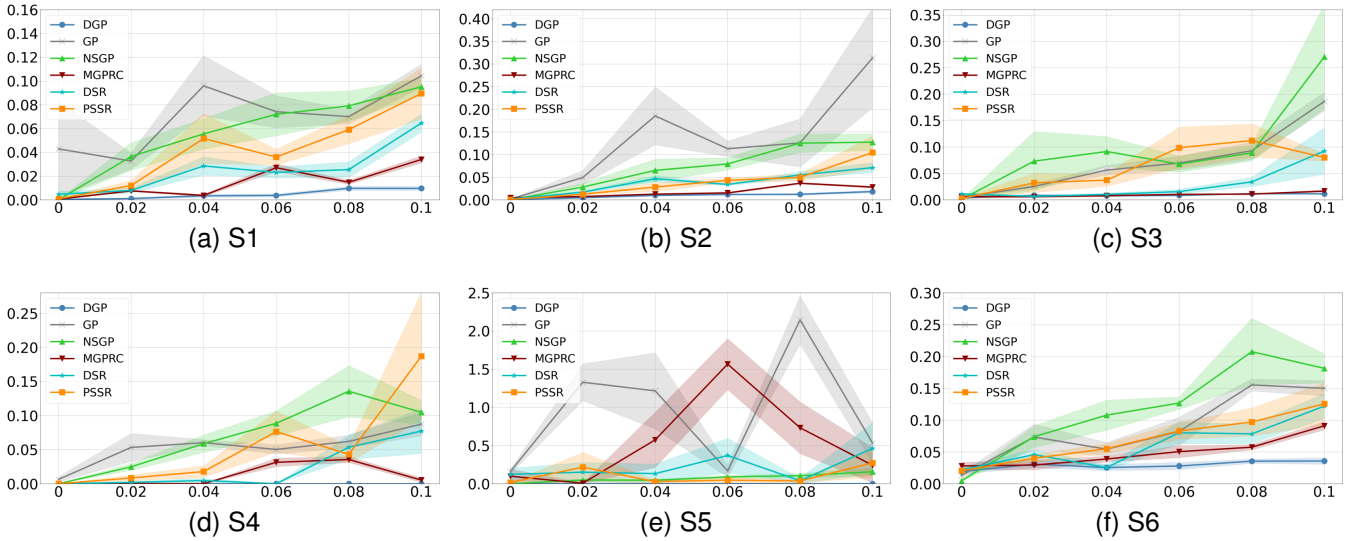


Fig. 8. The average test RMSE of different methods under different levels of dataset noises across all the expression recovery benchmarks.

while drastically reducing the computation time by over an order of magnitude from 34,159 seconds down to about 3,026 seconds. Overall, DGP attains a balance between accuracy, model complexity, and efficiency, showing great advantages in addressing high-dimensional SR problems.

B. Results on the expression recovery Benchmarks

In this section, we first analyse the overall recovery results of the methods on the expression recovery benchmarks in Subsection V-B1. We then analyse the program size of the searched expressions in Subsection V-B3. Finally, we compare the methods' robustness by adding various noise levels to the data and analysing the results in Subsection V-B2.

1) Recovery Results: Table IX details the recovery rates, which measure the frequency with which each algorithm successfully discovers the exact underlying mathematical equation on the expression recovery benchmarks (S1-S6). As shown in the table, DGP achieves the highest mean recovery rate of 73.3% across all benchmarks, significantly outperforming the second-best method PSSR (68.3%) and traditional GP (13.3%). Specifically, on relatively complex mathematical structures such as S2, S3, and S5, many competitors, e.g., GP, NSGP, and several recent SRBench performers like PSTree and SBP-GP, completely fail to identify the exact equations, resulting in a 0% recovery rate. In contrast, DGP maintains robust recovery rates of 80%, 50%, and a perfect 100% on these respective challenging targets. Based on the results, we can conclude that by introducing gradient-based guidance and a shrinking mechanism, DGP effectively mitigates structural bloating and enables a highly directed search.

2) Robustness Analysis: According to the experiments in [14], the search difficulty of true symbolic expressions varies with the noise level of the target variable, so it is important to demonstrate DGP's robustness to noise. We evaluate DGP on noisy data by adding independent Gaussian noise to the target variable, with mean zero and standard deviation proportional to the root-mean-square of the target variable in the training data. We found that most of the methods could not recover the exact symbolic expressions from data at a high noise level (i.e., > 0.06), meaning that the recovery rates are mostly zero and do not differentiate the effectiveness of the methods. Thus, we instead have reported the test RMSE (Fig. 8) of the different methods on various noise levels from 0 (noiseless) to 10^{-1}. As shown in Fig. 8, the performance attenuation of DGP is relatively minor compared to other methods under different noises. These results make DGP a very promising method for SR: it is both highly accurate and robust to noise across a range of datasets.

3) Program Size Analysis: In this section, we compare the program size of the symbolic expressions found by the different methods, where methods were able to successfully recover the expression from the data (i.e., where recovery rate

TABLE X
PROGRAM SIZE ANALYSIS ON THE EXPRESSION RECOVERY BENCHMARKS

Benchmark	Size(#Node of the best models)					
	DGP	GP	NSGP	MGPRC	DSR	PSSR
S1	$19^{\pm 5.4}$	$108^{\pm 7.1}$	$14^{\pm 1.9}$	$22^{\pm 3.8}$	-	$19^{\pm 4.2}$
S2	$34^{\pm 10.4}$	-	$42^{\pm 0.0}$	-	-	$26^{\pm 4.4}$
S3	$22^{\pm 3.1}$	-	-	$41^{\pm 18.5}$	-	$18^{\pm 1.6}$
S4	$9^{\pm 2.0}$	$123^{\pm 51.0}$	$12^{\pm 9.7}$	$23^{\pm 8.1}$	$7^{\pm 0.0}$	$11^{\pm 4.9}$
S5	$14^{\pm 4.7}$	-	-	$32^{\pm 10.5}$	$22^{\pm 9.6}$	$21^{\pm 9.3}$
S6	$19^{\pm 5.5}$	-	-	-	$13^{\pm 1.5}$	$29^{\pm 0.0}$

was greater than 0%). The results are presented in Table X.

As illustrated in Table X, canonical GP exhibits massive structural bloating, producing extremely large programs when it manages to recover a solution, e.g., an average of $108^{\pm 7.1}$ nodes on S1 and $123^{\pm 51.0}$ on S4. Similarly, other evolutionary methods like MGPRC still generate moderately large expressions, such as $41^{\pm 18.5}$ on S3 and $32^{\pm 10.5}$ on S5. In contrast, NN-based methods generally maintain much smaller profiles. The results reveal that DGP effectively bridges this gap, consistently generating highly compact expressions, i.e., $19^{\pm 5.4}$ on S1, $34^{\pm 10.4}$ on S2, $22^{\pm 3.1}$ on S3, $9^{\pm 2.0}$ on S4, $14^{\pm 4.7}$ on S5, and $19^{\pm 5.5}$ on S6. This demonstrates that DGP successfully avoids the bloating of evolutionary methods and rivals the compactness of NN-based SR baselines like DSR and PSSR. We conclude that the symbolic expressions recovered by the proposed DGP method can achieve high accuracy while being concise.

C. Efficacy Study

In this section, we first analyze the ability of DGP to jump out of local optima in Subsection V-C1. We then investigate the sensitivity to key hyperparameters in Subsection V-C2. Next, we analyze the computational efficiency regarding FLOPs in Subsection V-C3. Finally, a qualitative analysis of the searched expressions is provided in Subsection V-C4.

1) The ability of DGP in jumping out of local optima: To identify how often the models encounter local optima or search stagnation within the limited optimization generation, we designed an experiment to analyze the "Trapping Rate" of different methods. Specifically, an algorithm is considered temporarily trapped in a given generation if the R^2 score improvement from the previous generation is less than 10^{-4}. The trapping rate is then defined as the proportion of such trapped generations relative to the total number of generations. To ensure statistical reliability, all experiments were conducted across 30 different random seeds. The overall trapping rate is calculated as the average percentage of such trapped generations across all 30 runs. As presented in Table XI, pure gradient-based methods, i.e., DSR and DGP without diversification (w/o Div.), are highly trapped in local optima. In contrast, the proposed diversification mechanism in DGP can effectively escape these local optima, reaching the trapping frequency to merely 0.74%.

2) Sensitivity to key hyperparameter of DGP: We analyze the sensitivity of DGP to its core hyperparameters, including the penalty coefficient λ_{0-1}, population size, epoch number, and generation number. Table XII summarizes the resulting

TABLE XI
QUANTITATIVE ANALYSIS OF LOCAL OPTIMA TRAPPING FREQUENCY OVER 30 INDEPENDENT RUNS ON FRI_C4_50.

Method	Optimization Paradigm	Trapping Rate
DSR	Pure Gradient	97.49%
DGP (w/o Div.)	Pure Gradient	87.41%
DGP (Full)	Gradient + Diversification	0.74%

TABLE XII
HYPERPARAMETER SENSITIVITY ANALYSIS OF DGP.

Hyperparameter		Fri_c4_50		Fri_c0_50		Fri_c1_50	
Name	Value	R^2	Size	R^2	Size	R^2	Size
Penalty coefficient λ_{0-1}	0.01	0.85	164.8	0.82	104.5	0.86	127.9
	0.1	0.86	125.6	0.82	89.7	0.88	127.1
	1.0	0.66	128.8	0.80	148.1	0.79	97.4
Population size	50	0.60	108.8	0.73	95.4	0.64	117.9
	250	0.76	103.1	0.79	122.0	0.76	105.2
	500	0.79	128.0	0.83	120.1	0.80	100.5
Epoch number	10	0.66	92.8	0.52	55.0	0.52	196.5
	100	0.71	115.3	0.82	148.2	0.81	124.0
	1000	0.81	107.8	0.83	113.2	0.88	149.4
Generation number	5	0.64	75.5	0.74	77.1	0.75	63.2
	10	0.79	85.5	0.81	77.2	0.73	100.7
	20	0.82	108.7	0.84	90.4	0.82	104.3

TABLE XIII
THE FLOPS OF DIFFERENT METHOD UNDER SAME NUMBER OF EVALUATED INDIVIDUALS ON FRI_C4_50.

Method	FLOPs	R^2 score
GP	4.02×10^9	0.27
DSR	9.98×10^{10}	0.19
DGP (Ours)	8.36×10^{10}	0.65

R^2 performance and expression size across different settings. As shown in the table, setting λ_{0-1} to 0.1 provides a balance between an R^2 score of 0.86 and a model size of 125.6 on Fir_c4_50. This can potentially avoid both severe expression bloating ($\lambda_{0-1} = 0.01$) and underfitting ($\lambda_{0-1} = 1.0$). Moreover, increasing the population size, epoch number, and generation number consistently enhances the R^2 performance. This is because these parameters can govern the search breadth, the depth of gradient-based constant optimization, and the range of structural exploration. These results demonstrate that DGP is highly robust across different settings. Overall, the results strongly justify our default parameter settings ($\lambda_{0-1} = 0.1$, population size=500, epochs=1000, generations=20) can yield the most effective trade-off between performance and expression complexity across different datasets.

3) The efficiency analysis on FLOPs: To comprehensively evaluate the computational complexity of different methods, we conducted a quantitative Floating Point of Operations (FLOPs) analysis on traditional GP, DSR, and DGP. To ensure a fair comparison, all three methods were configured to evaluate 82,500 candidate expressions on Fri_c4_50. As shown in Table XIII, DGP yields significant performance gains over traditional GP, despite it achieving the second-best computational cost. Specifically, traditional GP consumes the fewest FLOPs (4.02×10^9) but fails to produce great performance ($R^2 = 0.27$). In contrast, the pure gradient-based DSR method incurs a higher computational cost (9.98×10^{10})

TABLE XIV
EXAMPLES OF SYMBOLIC EXPRESSIONS DISCOVERED BY DGP.

Dataset	Discovered symbolic expression
Fri_c0_50	$1.458 \times (0.434 \times (1.165207 \times 1.098007 \times sin(cos(x0) \times 0.475164 + cos(0.284923 \times x1 \times x1) \times x0 + x1) - (-0.007) \times 0.434 + x3 + 0.434 \times x4 + 0.434 \times log(x2)) - (-0.007) \times x0)$
Fri_c3_50	$0.445 \times (x4 + 2.697 \times sin((-0.05732) \times log((-0.005) \times (-0.029186) \times (x3 + x2 \times x0)) + (-0.979331) \times x0 + (-0.05732) \times x2 \times x0 + (-1.306221) \times x1 + (-0.029186) \times x3))$
Tecator	$(-0.888) \times (x122 - (-0.120084) \times (x123 - (-0.120084) \times x123) - 0.045 \times x111)$

due to frequent neural network updates, yet yields the lowest accuracy, i.e., $R^2 = 0.18$.

4) Qualitative analysis of searched expressions: Table XIV presents qualitative examples of expressions discovered by DGP, demonstrating it can identify interpretable functional forms on different datasets. For example, the results on the Fri_c0_50 and Fri_c3_50 datasets show DGP successfully captures core nonlinear interactions and the important feature, i.e., from x_0 to x_4. Moreover, DGP automatically prunes redundant dimensions, converging on a sparse equation with only three critical variables, i.e., x_{111}, x_{122}, and x_{123}. By maintaining these compact symbolic structures, DGP effectively balances performance with mathematical interpretability.

D. Ablation Studies

To investigate the contribution of each component of the proposed DGP method, we conduct the ablation studies on the Fri_c4_50 dataset. This dataset is relatively more lightweight than DLBCL, enabling faster training and evaluation, which makes it well-suited for systematic ablation studies. As shown in Table XV, DGP achieves the highest performance (an R^2 score of $0.64^{\pm 0.144}$ and a model size of $42^{\pm 20.9}$) when all three components, i.e., differentiable tree structure, sampling strategy, and diversification mechanism, are included. When the sampling strategy is removed, the performance drops significantly to $0.43^{\pm 0.217}$. This indicates the importance of the sampling strategy in guiding the model toward great expressions. Eliminating diversification causes a drastic degradation in performance and an extreme drop in model size (collapsing to an R^2 of $-0.40^{\pm 1.083}$ and a size of $5^{\pm 3.2}$), as the method is more prone to getting trapped in local optima and failing to grow meaningful structures. Similarly, the absence of the differentiable tree structure also causes a substantial performance drop, i.e., yielding an R^2 of $0.17^{\pm 0.077}$, confirming its core effectiveness in guiding symbolic expression formation. Moreover, configurations lacking two or more components generally yield poor or even degenerate performance, e.g., an R^2 of $-1.52^{\pm 2.069}$ when all components are removed. This emphatically highlights the absolute necessity and synergy of each component for achieving high-performance symbolic regression. Note that the sampling strategy mathematically depends on the continuous structure of the differentiable tree and cannot be evaluated independently.

TABLE XV
ABLATION STUDIES OF THE MAIN COMPONENTS ON FRI_C4_50.

Diff. Tree	Sampling	Diversification	R^2 score	Size
√	√	√	$0.64^{\pm 0.144}$	$42^{\pm 20.9}$
√	×	√	$0.43^{\pm 0.217}$	$47^{\pm 37.1}$
√	√	×	$-0.40^{\pm 1.083}$	$5^{\pm 3.2}$
√	×	×	$-0.61^{\pm 0.000}$	$6^{\pm 0.000}$
×	×	√	$0.17^{\pm 0.077}$	$16^{\pm 8.1}$
×	×	×	$-1.52^{\pm 2.069}$	$8.4^{\pm 2.9}$

E. Discussions and Relationship to Prior Works

To highlight the novelty of DGP, we contrast it with two paradigms of gradient-based symbolic regression. The first paradigm uses pre-trained NNs, e.g., DGSR [18], Symformer [17], and DSR [14], as probability generators for sequential symbol sampling. For example, DSR employs a softmax layer to output the symbol probabilities. While DGP also uses softmax, it uniquely applies softmax to create a continuous relaxation of the structural nodes and edges within a fixed tree node rather than generate symbol probabilities. This can transform the discrete search into a differentiable optimization process. The second paradigm embeds mathematical operators directly into the architecture of NNs, including EQL [6] and KAN [54]. Unlike those methods that modify NNs (e.g., activation functions and edges), DGP operates directly on symbolic tree structures. By embedding continuous gradients into the discrete GP framework, DGP leverages the population-based mechanism to escape local optima. In summary, DGP introduces a novel way to solve high-dimensional symbolic regression. By relaxing tree nodes and edges to create a DST structure, DGP enables direct gradient descent optimization. This can improve search efficiency over Symformer and DSR (see Table VIII and Table XIII). Additionally, the GP-based global exploration of DGP preserves hierarchical interpretability and avoids getting trapped in local optima compared with other pure gradient-based methods (see Table XI).

VI. CONCLUSIONS AND FUTURE WORK

The objective of this article was to propose an effective GP method for SR, which can avoid the ineffectiveness and bloating of traditional GP caused by the stochastic nature of its evolutionary-based search method. We have successfully achieved this goal by proposing a new Differentiable Genetic Programming (DGP) method, which uses a gradient descent method to optimize the structure of GP. For the gradient-based optimization, a new representation named the differentiable symbolic tree was proposed that relaxes the discrete structure into a continuous space. In addition, we also designed a loss function for SR, proposed a unique hybrid forward propagation method, and formulated the backward gradient calculation. With this design, the proposed DGP can search for the best structure more efficiently, making it an effective method to deal with high-dimensional symbolic regression problems compared to the state-of-the-art. The proposed method was compared to benchmark methods that included both GP-based and NN-based SR methods by testing on both complex regression and expression recovery benchmark datasets. The

experiment results showed that the training and generalization performance of DGP outperforms almost all the other GP-based and NN-based SR methods, and that DGP produces a relatively small solution size compared to other GP-based methods. This, in conjunction with further robustness testing, demonstrated that DGP is an effective tool for solving SR, especially for high-dimensional real-world problems that are difficult to solve with previous GP-based methods.

We note that the proposed gradient-based optimization method for GP is not only applicable for SR tasks but also for other GP-based ML problems, which should be explored in future work. In addition, our future work will explore simultaneously optimizing constants and tree structures during the training process to improve the performance of the proposed method on SR problems with constants. Finally, our proposed method makes use of the final output of the expression to compute the loss even in the early stage of the search – this potentially may lead to the search falling into a local optimum. Thus, investigating how to combine the search for partial expressions with the search for overall expressions is also a potential direction.

REFERENCES

[1] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[2] J. L. Awange and B. Paláncz, "Symbolic regression," in *Geospatial Algebraic Computations: Theory and Applications*, 2016, pp. 203–216.

[3] A. Prieto, B. Prieto, E. M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas, "Neural networks: An overview of early research, current frameworks and new challenges," *Neurocomputing*, vol. 214, pp. 242–268, 2016.

[4] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[5] E. Hosseini-Asl, J. M. Zurada, and O. Nasraoui, "Deep learning of part-based representation of data using sparse autoencoders with nonnegativity constraints," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 12, pp. 2486–2498, 2015.

[6] G. Martius and C. H. Lampert, "Extrapolation and learning equations," *arxiv.org*, 2016. [Online]. Available: https://arxiv.org/abs/1610.02995v1.

[7] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 4442–4450.

[8] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić, "Integration of neural network-based symbolic regression in deep learning for scientific discovery," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 9, pp. 4166–4177, 2021.

[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," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 17 429–17 442.

[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arxiv.org*, 2013. [Online]. Available: https://arxiv.org/abs/1312.6114v10.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, b. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[13] V. K. Ayyadevara, "Recurrent neural network," in *Pro Machine Learning Algorithms : A Hands-On Approach to Implementing Algorithms in Python and R*, 2018, pp. 217–257.

[14] 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," in *International Conference on Learning Representations*, 2021.

[15] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 1945–1954.

[16] T. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. faissol, and B. K. Petersen, "Symbolic regression via deep reinforcement learning enhanced genetic programming seeding," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 24 912–24 923.

[17] M. Vastl, J. Kulhánek, J. Kubalík, E. Derner, and R. Babuška, "Symformer: End-to-end symbolic regression using Transformer-based architecture," *IEEE Access*, vol. 12, pp. 37 840–37 849, 2024.

[18] P.-A. Kamienny, G. Lample, S. Lamprier, and M. Virgolin, "Deep generative symbolic regression with Monte-Carlo-tree-search," in *International Conference on Machine Learning*. PMLR, 2023, pp. 15 655–15 668.

[19] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo, "Neural symbolic regression that scales," in *Proceedings of the 38th International Conference on Machine Learning*, vol. 139, 2021, pp. 936–945.

[20] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2020.

[21] H. Li, Y. Weng, and H. Tong, "Console: Convex neural symbolic learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5766–5774, 2022.

[22] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza, "Genetic programming: An introduction and tutorial, with a survey of techniques and applications," in *Computational Intelligence: A Compendium*. Springer Berlin Heidelberg, 2008, pp. 927–1028.

[23] Y. A. Radwan, G. Kronberger, and S. Winkler, "A comparison of recent algorithms for symbolic regression to genetic programming," in *International Conference on Computer Aided Systems Theory*. Springer, 2024, pp. 157–171.

[24] J. Dong and J. Zhong, "Recent advances in symbolic regression," *ACM Computing Surveys*, vol. 57, no. 11, pp. 1–37, 2025.

[25] B. Al-Helali, Q. Chen, B. Xue, and M. Zhang, "Genetic programming for feature selection based on feature removal impact in high-dimensional symbolic regression," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 3, pp. 2269–2282, 2024.

[26] Q. Chen, M. Zhang, and B. Xue, "Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 792–806, 2017.

[27] 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*, vol. 12, no. 2, pp. 91–119, 2011.

[28] Q. Lu, J. Ren, and Z. Wang, "Using genetic programming with prior formula knowledge to solve symbolic regression problem," *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–17, 2016.

[29] J. F. Martins, L. O. Oliveira, L. F. Miranda, F. Casadei, and G. L. Pappa, "Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1151–1158.

[30] M. Virgolin, A. de Lorenzo, E. Medvet, and F. Randone, "Learning a formula of interpretability to learn interpretable formulas," in *Parallel Problem Solving from Nature – PPSN XVI*, 2020, pp. 79–93.

[31] Q. Chen, B. Xue, and M. Zhang, "Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression," *IEEE Transactions on Cybernetics*, vol. 52, no. 4, pp. 2382–2395, 2022.

[32] S. M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, p. eaay2631, 2020.

[33] S. Arslan and C. Ozturk, "Multi hive artificial bee colony programming for high dimensional symbolic regression with feature selection," *Applied Soft Computing*, vol. 78, pp. 515–527, 2019.

[34] P. Orzechowski, W. La Cava, and J. H. Moore, "Where are we now? a large benchmark study of recent symbolic regression methods," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1183–1190.

[35] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 877–884.

[36] F. O. de Franca and G. S. I. Aldeia, "Interaction–transformation evolutionary algorithm for symbolic regression," *Evolutionary Computation*, vol. 29, no. 3, pp. 367–390, 2021.

[37] L. Vanneschi, "An introduction to geometric semantic genetic programming," in *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop*, 2017, pp. 3–42.

[38] T. P. Pawlak, B. Wieloch, and K. Krawiec, "Semantic backpropagation for designing search operators in genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 326–340, 2015.

[39] Q. Chen, B. Xue, and M. Zhang, "Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 488–502, 2019.

[40] M. Virgolin, T. Alderliesten, C. Witteveen, and P. Bosman, "A model-based genetic programming approach for symbolic regression of small expressions," *arxiv.org*, 2019. [Online]. Available: http://arxiv.org/abs/1904.02050

[41] Q. Chen, M. Zhang, and B. Xue, "Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 792–806, 2017.

[42] Q. Chen, M. Zhang, and B, Xue, "Genetic programming with embedded feature construction for high-dimensional symbolic regression," in *Intelligent and Evolutionary Systems*, 2017, pp. 87–102.

[43] E. J. Vladislavleva, G. F. Smits, and D. den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 333–349, 2009.

[44] F. O. de França, "A greedy search tree heuristic for symbolic regression," *Information Sciences*, vol. 442-443, pp. 18–32, 2018.

[45] D. Kinzett, M. Johnston, and M. Zhang, "How online simplification affects building blocks in genetic programming," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 2009, pp. 979–986.

[46] P. Wong and M. Zhang, "Algebraic simplification of gp programs during evolution," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 2006, p. 927¨C934.

[47] L. A. Ferreira, F. G. Guimarães, and R. Silva, "Applying genetic programming to improve interpretability in machine learning models," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1–8.

[48] P. Orzechowski, W. La Cava, and J. H. Moore, "Where are we now? a large benchmark study of recent symbolic regression methods," *arxiv.org*, 2018. [Online]. Available: https://arxiv.org/abs/1804.09331.

[49] A. Topchy and W. F. Punch, "Faster genetic programming based on local gradient search of numeric leaf values," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2001, pp. 155–162.

[50] M. Zhang and W. Smart, "Learning weights in genetic programs using gradient descent for object recognition," in *Applications of Evolutionary Computing*, 2005, pp. 417–427.

[51] M. Graff, A. Graff-Guerrero, and J. Cerda-Jacobo, "Semantic crossover based on the partial derivative error," in *European Conference on Genetic Programming*, 2014, pp. 37–47.

[52] Q. Chen, B. Xue, and M. Zhang, "Generalisation and domain adaptation in GP with gradient descent for symbolic regression," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, 2015, pp. 1137–1144.

[53] G. Dick, C. A. Owen, and P. A. Whigham, "Feature standardisation and coefficient optimisation for effective symbolic regression," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 306–314.

[54] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Hou, and M. Tegmark, "KAN: Kolmogorov–arnold networks," in *International Conference on Learning Representations*, vol. 2025, 2025, pp. 70 367–70 413.

[55] M. Cranmer, "Interpretable machine learning for science with PySR and SymbolicRegression. jl," *arXiv preprint arXiv:2305.01582*, 2023.

[56] K. Shahookar, W. Khamisani, P. Mazumder, and S. M. Reddy, "Genetic beam search for gate matrix layout," *IEE Proceedings-Computers and Digital Techniques*, vol. 141, no. 2, pp. 123–128, 1994.

[57] M. Zhang and W. Smart, "Genetic programming with gradient descent search for multiclass object classification," in *European Conference on Genetic Programming*. Springer, 2004, pp. 399–408.

[58] M. O'Neill, R. Poli, W. B. Langdon, and N. F. McPhee, "A field guide to genetic programming," *Genetic Programming and Evolvable Machines*, vol. 10, no. 2, pp. 229–230, 2009.

[59] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *European conference on computer vision*, 2020, pp. 465–480.

[60] D. P. Kingma, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014.

[61] W. La Cava, P. Orzechowski, B. Burlacu, F. de Franca, M. Virgolin, Y. Jin, M. Kommenda, and J. Moore, "Contemporary symbolic regression methods and their relative performance," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, vol. 1, 2021.

[62] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "Pmlb: A large benchmark suite for machine learning evaluation and comparison," *arxiv.org*, 2017. [Online]. Available: https://arxiv.org/abs/1703.00512

[63] A. Rosenwald, G. Wright, W. C. Chan, J. M. Connors, E. Campo, R. I. Fisher, R. D. Gascoyne, H. K. Muller-Hermelink, E. B. Smeland, J. M. Giltnane *et al.*, "The use of molecular profiling to predict survival after chemotherapy for diffuse large-b-cell lymphoma," *New England Journal of Medicine*, vol. 346, no. 25, pp. 1937–1947, 2002.

[64] F. A. Fortin, F. M. Rainville, M. A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research, Machine Learning Open Source Software*, vol. 13, no. 1, pp. 2171–2175, 2012.

[65] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. Bosman, "Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 1041–1048.

[66] B. Burlacu, G. Kronberger, and M. Kommenda, "Operon c++ an efficient genetic programming framework for symbolic regression," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1562–1570.

[67] H. Zhang, A. Zhou, H. Qian, and H. Zhang, "PS-Tree: A piecewise symbolic regression tree," *Swarm and Evolutionary Computation*, vol. 71, p. 101061, 2022.

[68] W. La Cava, T. R. Singh, J. Taggart, S. Suri, and J. H. Moore, "Learning concise representations for regression by evolving networks of trees," in *International Conference on Learning Representations*, 2019.

[69] M. Virgolin, T. Alderliesten, and P. A. Bosman, "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 1084–1092.

[70] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *Peerj computer science*, vol. 7, p. e623, 2021.

[71] W. La Cava, B. Burlacu, M. Virgolin, M. Kommenda, P. Orzechowski, F. O. de França, Y. Jin, and J. H. Moore, "Contemporary symbolic regression methods and their relative performance," *Advances in Neural Information Processing Systems*, vol. 2021, no. DB1, p. 1, 2021.

[72] P. Orzechowski, W. La Cava, and J. H. Moore, "Where are we now? A large benchmark study of recent symbolic regression methods," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1183–1190.

[73] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2026.3734437

Xiaotian Song received the B.S. degree from the School of Computing Science and Engineering, Sun Yat-sen University, Guangzhou, China, in 2019, where he is currently pursuing the Ph.D. degree with the College of Computer Science, Sichuan University, Chengdu.

His main current research interests include neural architecture search, evolutionary algorithms, and evolutionary deep learning.

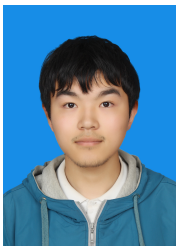
Peng Zeng received the B.E. degree in software engineering from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2021, and the M.S. degree from the College of Computer Science, Sichuan University, Chengdu, China, in 2024.

His research interests include genetic programming and machine learning.

Andrew Lensen (Member, IEEE) received the B.Sc., B.Sc. (First-Class Hons.), and Ph.D. degrees in computer science from Te Herenga Waka—Victoria University of Wellington (VUW), Wellington, New Zealand, in 2015, 2016, and 2019, respectively.

He is currently a Senior Lecturer in Artificial Intelligence with the Centre of Data Science and Artificial Intelligence and School of Engineering and Computer Science, VUW. His current research interests are mainly in the use of evolutionary computation for feature manipulation in unsupervised learning, with a particular focus on the use of genetic programming for manifold learning and clustering, as well as explainable artificial intelligence and AI ethics.

Dr. Lensen serves as a Regular Reviewer of several international conferences, including IEEE Congress on Evolutionary Computation, and international journals, such as the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Cybernetics, and the IEEE Transactions on Emerging Topics In Computational Intelligence. He is also a Committee Member of the IEEE NZ Central Section and a member of the IEEE Computational Intelligence Society.



Hengzhe Zhang (Member, IEEE) is a postdoctoral researcher in computer science at Victoria University of Wellington, New Zealand. He received the B.Sc. degree in software engineering from Xiangtan University, Hunan, China, in 2019, and the M.Sc. degree in computer science from East China Normal University, Shanghai, China, in 2022.

His current research interests include symbolic regression, genetic programming, evolutionary computation, and statistical machine learning.

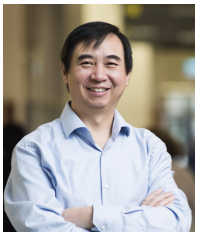
Yuwei Ou received the M.S. degree in computer science from Sichuan University, Chengdu, China, in 2024.

He is currently working with EZVIZ Inc., Hangzhou, China. His research interests during his studies include neural architecture search and adversarial attacks. His current research interests include embodied intelligence.

Yanan Sun (Senior Member, IEEE) received the Ph.D. degree in computer science from Sichuan University, Chengdu, China, in 2017.

He is currently a Professor with the College of Computer Science, Sichuan University. His research interests include evolutionary computation, neural networks, and their applications on neural architecture search.

Dr. Sun led a team winning the First Place on the third unseen-data neural architecture search challenge in CVPR2023. He serves as an Associate Editor for IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.



Mengjie Zhang (Fellow, IEEE) is Professor of Computer Science (Artificial Intelligence) at Victoria University of Wellington, where he heads the interdisciplinary Evolutionary Computation and Machine Learning Research Group. He is also the Director of the Centre for Data Science and Artificial Intelligence at the University. His research is mainly focused on AI, machine learning and big data, particularly in evolutionary learning and optimisation, feature selection/construction, computer vision and image analysis, scheduling and combinatorial optimisation, classification with unbalanced data and missing data, and evolutionary deep learning and transfer learning. He received the "Evo* Award for Outstanding Contribution to Evolutionary Computation in Europe 2023", the "2024 Australasian Artificial Intelligence Distinguished Research Contribution Award", and the ACM SIGEVO Outstanding Contribution Award in 2025.

He is also a Clarivate Highly Cited Researcher in 2023, 2024 and 2025. Since 2007, he has been listed as a top five (currently No. 2) world genetic programming researchers by the GP bibliography. Prof Zhang is a member of the IEEE CIS Nomination Committee. He is also a part chair for CIS Awards Committee, a past chair of the IEEE CIS Intelligent Systems Applications Technical Committee, the Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, a past Chair for IEEE CIS PubsCom Strategic Planning Committee, and the founding chair of the IEEE Computational Intelligence Chapter in New Zealand. He is a Fellow of Royal Society of New Zealand, a Fellow of Engineering New Zealand, and a Fellow of IEEE.

Jiancheng Lv (Senior Member, IEEE) received the PhD degree in computer science and engineering from University of Electronic Science and Technology of China in 2006. He was a research fellow at Department of Electrical and Computer Engineering, National University of Singapore, Singapore. He is currently a professor at College of Computer Science, Sichuan University, China. His research interests include neural networks, machine learning, and big data.