Interpretable scientific discovery with symbolic regression: a review

Nour Makke[1] · **Sanjay Chawla**[1]

Abstract

Symbolic regression is emerging as a promising machine learning method for learning succinct underlying interpretable mathematical expressions directly from data. Whereas it has been traditionally tackled with genetic programming, it has recently gained a growing interest in deep learning as a data-driven model discovery tool, achieving significant advances in various application domains ranging from fundamental to applied sciences. In this survey, we present a structured and comprehensive overview of symbolic regression methods, review the adoption of these methods for model discovery in various areas, and assess their effectiveness. We have also grouped state-of-the-art symbolic regression applications in a categorized manner in a living review.

Keywords Symbolic Regression · Automated Scientific Discovery · Interpretable AI

1 Introduction

Symbolic Regression (SR) is a rapidly growing subfield within machine learning (ML) to infer symbolic mathematical expressions from data (Koza 1994; Schmidt and Lipson 2009). Interest in SR is being driven by the observation that it is not sufficient to only have accurate predictive models; however, it is often necessary that the learned models be interpretable (Rudin 2019). A model is interpretable if the relationship between the input and output of the model can be logically or mathematically traced in a succinct manner. In other words, learnable models are interpretable if expressed as mathematical equations. As "disciplines" become increasingly data-rich and adopt ML techniques, the demand for interpretable models is likely to grow. For example, in the natural sciences (e.g., physics), mathematical models derived from first principles make it possible to reason about the underlying phenomenon in a way that is not possible with predictive models like deep neural networks. In critical disciplines like healthcare, non-interpretable models may never be allowed to be deployed - however accurate they maybe (Mozaffari-Kermani et al. 2015).

✉ Nour Makke
nmakke@hbku.edu.qa

Sanjay Chawla
schawla@hbku.edu.qa

1 Qatar Computing Research Institute, HBKU, Doha, Qatar

Example: Consider a data set consisting of samples (q_1, q_2, r, F), where q_1 and q_2 are the charges of two particles, r is the distance between them and F is the measured force between the particles. Assume q_1, q_2, and r are the input variables, and F is the output variable. Suppose we model the input–output relationship as $F = \theta_0 + \theta_1 q_1 + \theta_2 q_2 + \theta_3 r$. Then, using the data set, we can infer the model's parameters (θ_i). The model will be interpretable because we will know the impact of each variable on the output. For example, if θ_3 is negative, then that implies that as r increases, the force F will decrease. From physics, we know that this form of the model is unlikely to be accurate. On the other hand, we could model the input-output relationship using a neural network (NN), i.e., $F = NN(q_1, q_2, r, \theta)$. We expect the model to be highly accurate and predictive because neural networks are universal function approximators. However, the model is uninterpretable because the input–output relationship is not easily apparent. The input feature vector subsequently undergoes several layers of nonlinear transformations, i.e., $y = \sigma(\sum_i W_i \, \sigma(\sum_j W_j \, \sigma(\sum_k W_k \, \sigma(\cdots \sum_\ell W_\ell x))))$, where σ is a nonlinear activation function, and W_{idx} are the learnable parameters of the NN layer of index *idx*. Such models, called *"black-box"*, do not have an internal logic to let users understand how inputs are mathematically mapped to outputs. Explainability is the application of other methods to explain model predictions and to understand how it is learned. It refers to why the model makes the decision that way. What distinguishes explainability from interpretability is that interpretable models are transparent (Rudin 2019). For example, the linear regression model predictions can be interpreted by evaluating the relative contribution of individual features to the predictions using their weights. An ideal SR model will return the relationship as $k\frac{q_1 q_2}{r^2}$, which is the definition of the Coulomb force between two charged particles with a constant[1] $k = 8.98 \times 10^9$. However, learning the SR model is highly non-trivial as it involves searching over a large space of mathematical operations and identifying the right constant (k) that will fit the data. SR models can be directly inferred from data or can be used to *"whitebox"* a *"blackbox"* model such as a neural network.

The ultimate goal of SR is to bridge data and observations following the Keplerian trial and error approach (Kepler 1953). Kepler developed a data-driven model for planetary motion using the most accurate astronomical measurements of the era, which resulted in elliptic orbits described by a power law. In contrast, Newton developed a dynamic relationship between physical variables that described the underlying process at the origin of these elliptic orbits. Newton's approach (Newton et al. 1729) led to three laws of motion later verified by experimental observations. Whereas both methods fit the data well, Newton's approach could be generalized to predict behavior in regimes where no data were available. Although SR is regarded as a data-driven model discovery tool, it aims to find a symbolic model that simultaneously fits data well and could be generalized to uncovered regimes.

SR is deployed as an interpretable and predictive ML model or a data-driven scientific discovery method. SR was investigated as early as 1970 in research works (Gerwin 1974; Langley 1981; Falkenhainer and Michalski 1986) aiming to rediscover empirical laws. Such works iteratively apply a set of data-driven heuristics to formulate mathematical expressions. The first AI system meant to automate scientific discovery is called BACON (PW 1979; Langley et al. 1987). It was developed by Patrick Langley in the late 1970s and was successful in rediscovering versions of various physical laws, such as Coulomb's law and Galileo's laws for the pendulum and constant acceleration, among many others. SR was later studied by Koza (1989, 1990, 1994) who proposed that genetic programming (GP) can be used to discover symbolic models by encoding mathematical expressions as computational trees, where GP is an evolutionary algorithm that iteratively evolves an initial population of individuals via

---

[1] k is the electric force (or Coulomb) constant, $k = 8.9875517923 \times 10^9$ kg m^{3}s^{-4}A^{-2} in SI base units.

biology-inspired operations. SR was since then tackled with GP-based methods (Koza 1994; Keijzer 2003; Vladislavleva et al. 2009; Korns 2011; Uy et al. 2010; Jin et al. 2019; Petersen 2019; McConaghy 2011; Virgolin et al. 2019; de França and Aldeia 2019; Arnaldo et al. 2014; Cava et al. 2018). Moreover, it was popularized as a data-driven scientific discovery tool with the commercial software Eureqa (Dubcakova 2011) based on a research work (Schmidt and Lipson 2009). Whereas GP-based methods achieve high prediction accuracy, they do not scale to high dimensional data sets and are sensitive to hyperparameters (Petersen 2019). More recently, SR has been addressed with deep learning-based methods (Udrescu and Tegmark 2019; Martius and Lampert 2016; Petersen 2019; Mundhenk et al. 2021; Alaa and Schaar 2019; Kamienny et al. 2022; Biggio et al. 2021; Champion et al. 2019) which leverage neural networks (NNs) to learn accurate symbolic models. SR has been applied in fundamental and applied sciences such as astrophysics (Lemos et al. 2022), chemistry (Batra et al. 2020; Hernandez et al. 2019), materials science (Wang et al. 2019; Weng et al. 2020), semantic similarity measurement (Martinez-Gil and Chaves-Gonzalez 2020), climatology (Abdellaoui and Mehrkanoon 2021), medicine (Virgolin et al. 2020), among many others. Many of these applications are promising, showing the potential of SR. A recent SR benchmarking platform *SRBench* is introduced by Cava et al. (2021). It comprises 14 SR methods (among which ten are GP-based), applied on 252 data sets. The goal of *SRBench* was to provide a benchmark for rigorous evaluation and comparison of SR methods.

This survey aims to help researchers effectively and comprehensively understand the SR problem and how it could be solved, as well as to present the current status of the advances made in this growing subfield. The survey is structured as follows. First, we define the SR problem, present a structured and comprehensive review of methods, and discuss their strengths and limitations. Furthermore, we discuss the adoption of these SR methods across various application domains and assess their effectiveness. Along with this survey, a living review (Makke and Chawla 2022) aims to group state-of-the-art SR methods and applications and track advances made in the SR field. The objective is to update this list often to incorporate new research works.

This paper is organized as follows. The SR problem definition is presented in Sect. 2. We present an overview of methods deployed to solve the SR problem in Sect. 3, and the methods are discussed in detail in Sects. 4, 5 and 6. Selected applications are described and discussed in Sect. 7. Section 8 presents an overview of existing benchmark data sets. Finally, we summarize our conclusions and discuss perspectives in Sects. 9, 10.

2 Problem definition

The problem of symbolic regression can be defined in terms of classical Empirical Risk Minimization (ERM) (Vapnik 1991).

Data: Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the input vector and $y_i \in \mathbb{R}$ is a scalar output.
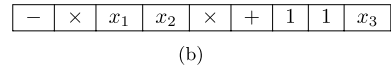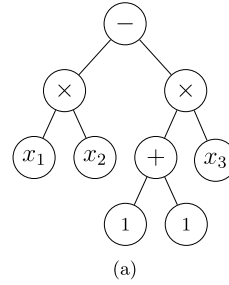
Function Class: Let $\mathcal{F}$ be a function class consisting of mappings $f : \mathbb{R}^d \to \mathbb{R}$.

Loss Function: Define the loss function for every candidate $f \in \mathcal{F}$:

$$l(f) := \sum_{i=1}^{n} l(f(\mathbf{x}_i), y_i) \tag{1}$$

A common choice is the squared difference between the output and prediction, i.e. $l(f) = \sum_i (y_i - f(\mathbf{x}_i))^2$.

Fig. 1 **a** Example of a unary-binary tree that encodes $f(x) = x_1x_2 - 2x_3$. **b** Sequence representation of the tree-like structure of $f(x)$

(a)

(b)

Optimization: The optimization task is to find the function (f) over the set of functions $\mathcal{F}$ that minimizes the loss function:

$$f^* = \arg\min_{f \in \mathcal{F}} l(f) \tag{2}$$

As stated below, what distinguishes SR from conventional regression problems is the discrete nature of the function class $\mathcal{F}$. Different methods for solving the SR problem reduce to characterizing the function class.

2.1 Class of function

In SR, to define $\mathcal{F}$, we specify a library of elementary arithmetic operations and mathematical functions and variables, and an element $f \in \mathcal{F}$ is the set of all functions that can be obtained by function composition in the library (Virgolin and Pissis 2022). For example, consider a library:

$$L = \{\mathrm{id}(\cdot),\ \mathrm{add}(\cdot, \cdot),\ \mathrm{sub}(\cdot, \cdot),\ \mathrm{mul}(\cdot, \cdot), +1, -1\} \tag{3}$$

Then the set of all of the polynomials (in one variable x) with integer coefficients can be derived from L using function composition.

2.2 Expression representation

It is convenient to express symbolic expressions in a sequential form using either a unary-binary expression tree or the polish notation (Robinson 1958). For example, the expression $f(x) = x_1x_2 - 2x_3$ can be derived using function composition from L (Eq. 3) and represented as a tree-like structure illustrated in Fig. 1a. By traversing the (binary) tree top to bottom and left to right in a depth-first manner, we can represent the same expression as a unique sequence called the polish form, as illustrated in Fig. 1b.

In practice, the library L includes many other common elementary mathematical functions, including the basic trigonometric functions like sine, cosine, logarithm, exponential, square root, power low, etc. A prior domain knowledge is advantageous for library definition because it reduces the search space to only include the most relevant mathematical operations to the studied problem. Furthermore, a large range of possible numeric constants should be possible
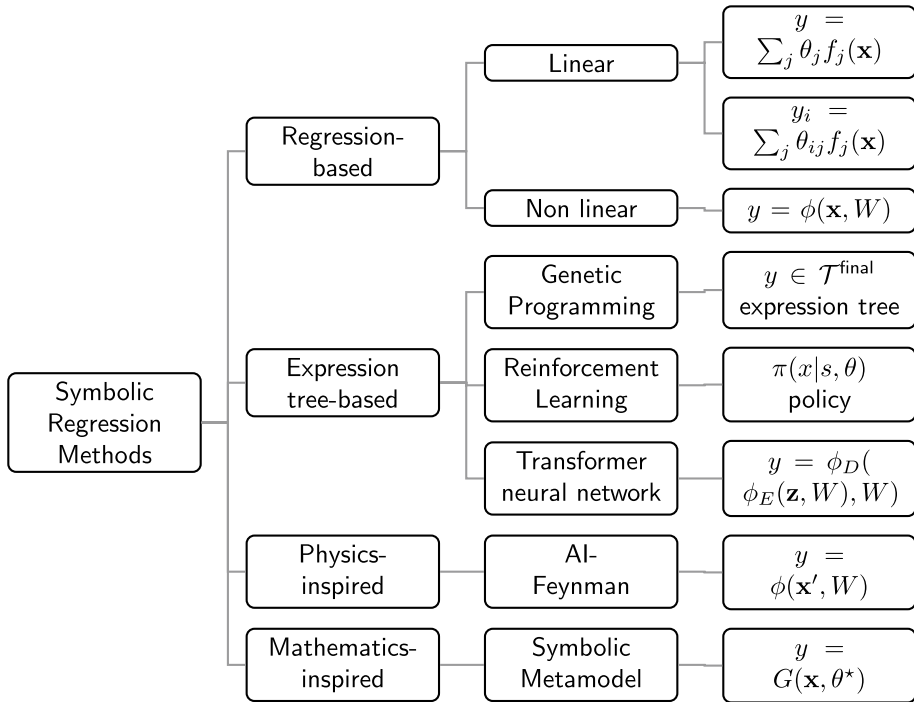
Fig. 2 Taxonomy based on the type of symbolic regression methods. ϕ denotes a neural network function, W denotes the set of learnable parameters in a NN. $\mathbf{x}$ denotes the input data, $\mathbf{z}$ denotes a reduced representation of $\mathbf{x}$, and $\mathbf{x}'$ denotes a new representation of $\mathbf{x}$, e.g., by defining new features based on the original ones. $\mathcal{T}$ represents the final population of selected expression trees in genetic programming. The lower indices (E) and (D) refer to the encoder and the decoder components of a transformer neural network. G denotes the Meijer function which will be discussed in the following

to express. For example, numbers in base-10 floating point notation rounded up to four significant digits can be represented as triple of (*sign*, *mantissa*, *exponent*) (Kamienny et al. 2022). The function $\sin(3.456x)$, for example, can be represented as $[\sin, \ mul, \ 3456, \ E - 3, \ x]$.

3 Symbolic regression methods overview

In this survey, we categorize SR methods in the following manner: regression-based methods, expression tree-based methods, physics-inspired and mathematics-inspired methods, as presented in Fig. 2. For each category, a summary of the mathematical tool, the expression form, the set of unknowns, and the search space, is presented in Table 1.

The linear method defines the functional form as a linear combination of nonlinear functions of x that are comprised in the predefined library L. Linear models are expressed as:

$$f(\mathbf{x}, \theta) = \sum_j \theta_j h_j(\mathbf{x})$$

(4)

where j spans the base functions of L. The optimization problem reduces to find the set of parameters $\{\theta\}$ that minimizes the loss function defined over a continuous parameter space $\Theta = \mathbb{R}^M$ as follows:

Table 1 Table summarizing symbolic regression methods

Method	Tool	Expression form	Unkown	Search space
Linear SR	Uni-D linear system	$y = \sum_i \theta_i f_i(\mathbf{x})$	$\{\theta\}_i$	$\mathbb{R}$
	Multi-D linear system	$y_i = \sum_j \theta_j f_j(\mathbf{x})$	$(\{\theta\}_i)_j$	$\mathbb{R}$
Nonlinear SR	Neural Network	$y = f(W \cdot \mathbf{x} + b)$	$\{W, b\}$	$\mathbb{R}$
Expression-tree search	Genetic Programming	Expression tree		trees
	Transformers	set2seq mapping	$\{W_q, W_k, W_v\}$	$\mathbb{R}$
	Reinforcement learning	set2seq mapping	$\pi(\theta)$	$\mathbb{R}$
Physics-inspired	AI-Feynman	$y = f(\mathbf{x}, \theta)$	$-$	$-$
Mathematics-inspired	Symbolic metamodels	$G(\mathbf{x}, \theta)$	θ	$\mathbb{R}$

The mathematical tool, the expression form, the set of unknowns, and the search space are specified for each method

Set2seq set-to-sequence

$$\theta^* = \underset{\theta \in \Theta}{\arg\min} \sum_i l(f(\mathbf{x}_i, \theta), y_i) \tag{5}$$

This method is advantageous for being deterministic and disadvantageous because it imposes a single model structure which is fixed during training when the model's parameters are learned.

The nonlinear method defines the model structure by a neural network. Nonlinear models can thus be expressed as:

$$f(\mathbf{x}, W) = \sigma \left(\sum_i W_i \, \sigma \left(\sum_j W_j \, \sigma \left(\cdots \sum_\ell W_\ell \mathbf{x} \right) \right) \right) \tag{6}$$

where σ is a nonlinear activation function, and W_{idx} are the learnable parameters of the NN layer of index *idx*. Similarly to the linear method, the optimization problem reduces to find the set of parameters $\{W, b\}$ of neural network layers, which minimizes the loss function over the space of real values.

Expression tree-based methods treat mathematical expressions as unary-binary trees whose internal nodes are operators and terminals are operands (variables or constants). This category comprises GP-based, deep neural transformers, and reinforcement learning-based methods. In GP-based methods, a set of transition rules (e.g., mutation, crossover, etc.) is defined over the tree space and applied to an initial population of trees throughout many iterations until the loss function is minimized. Transformers (Vaswani et al. 2017) represent a novel architecture of neural network (encoder and decoder) that uses attention mechanism. The latter was primarily used to capture long-range dependencies in a sentence. Transformers were designed to operate on sequential data and to perform sequence-to-sequence (seq2seq) tasks. For their use in SR, input data points $(\mathbf{x}, y)$ and symbolic expressions (f) are encoded as sequences and transformers perform set-to-sequence tasks. The unknowns are the weight parameters of the encoder and the decoder. Reinforcement learning (RL) is a machine learning method that seeks to learn a policy $\pi(x|\theta)$ by training an agent to perform a task by interacting with its environment in discrete time steps. An RL

setting requires four components: state space, action space, state transition probabilities, and reward. The agent selects an action that is sent to the environment. A reward and a new state are sent back to the agent from its environment and used by the agent to improve its policy at the next time step. In the context of SR, symbolic expression (sequence) represents a state, predicting an element in a sequence represents an action, the parent and sibling represent the environment, and the reward is commonly chosen as the mean square error (MSE). RL-based SR methods are commonly hybrid and use various ML tools (e.g., NN, RNN, etc.) in a joint manner with RL.

4 Linear symbolic regression

The linear approach assumes, by definition, that the target symbolic expression ($f(x)$) is a linear combination of nonlinear functions of feature attributes:

$$f(\mathrm{x}) = \sum_j \theta_j h_j(\mathrm{x}) \tag{7}$$

Here x denotes the input features vector, θ_j denotes a weight coefficient, and $h_j(\cdot)$ denotes a unary operator of the library L. This approach predefines the model's structure and reduces the SR problem to learn only the model's parameters by solving a system of linear equations. The particular case where $f(x)$ is a linear combination of degree-one monomial reduces to a conventional linear regression problem, i.e., $f(x) = \sum_j \theta_j x^j = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots$. There exist two cases for this problem: (1) a unidimensional case defined by $f : \mathbb{R}^d \to \mathbb{R}$; and (2) a multidimensional case defined by $f : \mathbb{R}^d \to \mathbb{R}^m$, with d the number of input features and m the number of variables required for a complete description of a system; for example, the Lorenz system for fluid flow is defined in terms of three physical variables which depend on time.

4.1 Unidimensional case

Given a data set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, the mathematical expression could be either univariate ($x_i \in \mathbb{R}$, $y_i = f(x_i)$) or multivariate ($\mathbf{x}_i \in \mathbb{R}^d$, $y_i = f(\mathbf{x}_i)$). The methodology of linear SR is presented in detail for the univariate case in Secion 4.1.1 for simplicity and is extended for the multivariate case in Sect. 4.1.2.

4.1.1 Univariate function

Data set: $\mathcal{D} = \{x_i \in \mathbb{R}; y_i = f(x_i)\}$.

Library: L can include any number of mathematical operators such that the dimension of the data set is always greater than the dimension of the library matrix (see discussion below).

In this approach, a coefficient θ_j is assigned to each candidate function ($f_j(\cdot) \in L$) as an activeness criterion such that:

$$y = \sum_j \theta_j f_j(x) \tag{8}$$

Applying Eq. 8 to input–output pairs (x_i, y_i) yields a system of linear equations as follows:

$$\begin{aligned}
y_1 &= \theta_0 + \theta_1 f_1(x_1) + \theta_2 f_2(x_1) + \cdots + \theta_k f_k(x_1) \\
y_2 &= \theta_0 + \theta_1 f_1(x_2) + \theta_2 f_2(x_2) + \cdots + \theta_k f_k(x_2) \\
&\vdots \\
y_n &= \theta_0 + \theta_1 f_1(x_n) + \theta_2 f_2(x_n) + \cdots + \theta_k f_k(x_n)
\end{aligned} \tag{9}$$

which can be represented in a matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & f_1(x_1) & f_2(x_1) & \cdots & f_k(x_1) \\ 1 & f_1(x_2) & f_2(x_2) & \cdots & f_k(x_2) \\ \vdots & & & & \\ 1 & f_1(x_n) & f_2(x_n) & \cdots & f_k(x_n) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{bmatrix} \tag{10}$$

Equation 10 can then be presented in a compact form:

$$Y = U(X) \cdot \Theta \tag{11}$$

where $\Theta \in \mathbb{R}^{(k+1)}$ is the sparse vector of coefficients, and $U \in \mathbb{R}^{n \times (k+1)}$ is the library matrix which can be represented as a function of the input vector X as follows:

$$U(X) = \begin{bmatrix} | & | & | & & | \\ 1 & f_1(X) & f_2(X) & \cdots & f_k(X) \\ | & | & | & & | \end{bmatrix} \tag{12}$$

Example: For a library defined as:

$$L = \{1, \ x, \ (\cdot)^2, \ \sin(\cdot), \ \cos(\cdot), \ \exp(\cdot)\} \tag{13}$$

The matrix U becomes:

$$U(X) = \begin{bmatrix} | & | & | & | & | & | \\ 1 & X & X^2 & \sin(X) & \cos(X) & \exp(X) \\ | & | & | & | & | & | \end{bmatrix}$$

Each row (of index i) in Eq. 12 is a vector of $(k + 1)$ functions of x_i. The vector of coefficients, i.e., the model's parameters, is obtained by solving Eq. 11 as follows[2]:

$$\Theta = (U^T U)^{-1} U^T Y \tag{14}$$

The magnitude of a coefficient θ_k effectively measures the size of the contribution of the associated function $f_k(\cdot)$ to the final prediction. Finally, the prediction vector $\hat{Y}$ can be evaluated using Eq. 11.

An exemplary schematic is illustrated in Fig. 3 for the univariate function $f(x) = 1 + \alpha x^3$. Only coefficients associated with functions $\{1, x^3\}$ of the library are non-zero, with values equal to 1 and α, respectively.

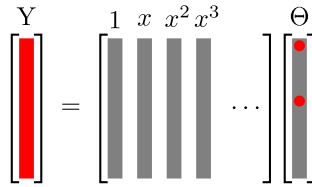---

[2] Technically the pseudo-inverse, U^+

Fig. 3 Schematic of the system of linear equations of Eq. 11 for $f(x) = 1 + \alpha x^3$. A library matrix U(X) of nonlinear functions of the input is constructed, where $L = \{1, x, x^2, x^3, \cdots\}$. The marked entries in the Θ vector denote the non-zero coefficients determining which functions of the library are active

Table 2 Results of linear SR in the case of univariate functions

Benchmark	Expression	L_1		L_2	
		Exp	R^2	Exp	R^2
Nguyen-2	$x^4 + x^3 + x^2 + x$	T	1.0	F	0.886
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	T	1.0	F	0.867
Livermore-21	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	T	1.0	F	0.869
Livermore-9	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	T	1.0	F	0.882
Livermore-6	$x + 2x^2 + 3x^3 + 4x^4$	T	1.0	F	0.417
Livermore-19	$x + x^2 + x^4 + x^5$	T	1.0	F	−0.079
Livermore-14*	$x + x^2 + x^3 + \sin(x)$	F	1.0	F	−0.857
Nguyen-5	$\sin(x^2)\cos(x) - 1$	F	0.999	F	−3.97
Nguyen-6	$\sin(x) + \sin(x + x^2)$	F	0.999	F	0.564

$\mathcal{D} = \{(x_i; y_i)\}$; $\quad x_i \in U(-1, 1, 20) \quad$ and $\quad y_i = f(x_i)$. $\quad L_1 = \{x, (\cdot)^2, \cdots, (\cdot)^9\} \quad$ and $L_2 = L_1 \cup \{\sin(\cdot), \cos(\cdot), \tan(\cdot), \exp(\cdot), \text{sigmoid}(\cdot)\}$
T true, F false

Fig. 4 Result of linear SR for the Nguyen-1 benchmark, i.e., $f(x) = x + x^2 + x^3$. Red points represent (test) data set. The red curve represents the true function. The blue and black dashed curves represent the learned functions using L_1 and L_2, respectively
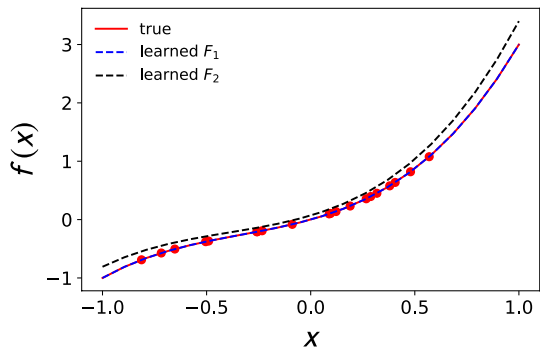
In the following, linear SR is tested on synthetic data. In each experiment, training and test data sets are generated. Each set consists of twenty data points randomly sampled from a uniform distribution U(−1, 1), and y is evaluated using a univariate

function, i.e., $\mathcal{D} = \{(x_i, f(x_i))\}_{i=1}^n$. Two libraries are considered in these experiments: $L_1 = \{x, (\cdot)^2, (\cdot)^3, \cdots, (\cdot)^9\}$ and $L_2 = L_1 \cup \{\sin(\cdot), \cos(\cdot), \tan(\cdot), \exp(\cdot), \text{sigmoid}(\cdot)\}$. The results are reported in terms of the output expression (Eq. 7) and the coefficient of determination R^2. SR problems are grouped into (i) pure polynomial functions and (ii) mixed polynomial and trigonometric functions. In each experiment, parameters are learned using the training data set, and results are reported for the test data set in Table 2.

For polynomial functions, an exact output is obtained using L_1 with an $R^2 = 1.0$, whereas only approximate output is obtained using L_2. In the latter case, the quality of the fit depends on the size of the training data set. An exemplary result is shown in Fig. 4 for $f(x) = x + x^2 + x^3$. Points represent the (test) data of the input file, i.e., X; the red curve represents $f(x)$ as a function of x, and the blue and black dashed curves represent the predicted function $\hat{f}(x)$ obtained using L_1 and L_2 respectively. An exact match between the ground-truth function and the predicted one is found using L_1, whereas a significant discrepancy is obtained using L_2. This discrepancy could be explained by the fact that various functions in L_2 exhibit the same x-dependence over the covered x-range.

For mixed polynomial and trigonometric expressions, both library choices do not produce the exact expression. However, a better R^2-coefficient is obtained using L_1. In the case of Nguyen-5 benchmark for example, i.e., $f(x) = \sin(x^2)\cos(x) - 1$, the resulting function is the Taylor expansion of f:

$$\hat{y}(x) \approx -1 + 0.9x^2 - 0.5x^4 - 0.13x^6 + \mathcal{O}(x^8)$$

In conclusion, this approach can not learn the ground-truth function when the latter is a multiplication of two functions (i.e., $f(x) = f_1(x) * f_2(x)$) or when it has a multiplicative or an additive factor to the variable (e.g., $\sin(\alpha + x)$, $\exp(\lambda * x)$, etc.). In the best case, it outputs an approximation of the ground-truth function. Furthermore, this approach fails to predict the correct mathematical expression when the library is extended to include a mixture of polynomial, trigonometric, exponential, and logarithmic functions.

4.1.2 Multivariate function

For a given data set $\mathcal{D} = \{x_i \in \mathbb{R}^d; y_i = f(x_1, \cdots, x_d)\}$, where d is the number of features, the same equations presented in Sect. 4.1.1 are applicable. However, the dimension of the library matrix U changes to consider the features vector dimension. For example, for the same library shown in Eq. 13 and a two dimensional features vector, i.e., $X \in \mathbb{R}^2$, U(X) becomes:

$$U(X) = \begin{bmatrix} | & | & | & | & | & | \\ 1 & X & X^{P2} & \sin(X) & \cos(X) & \exp(X) \\ | & | & | & | & | & | \end{bmatrix}$$

$$= \begin{bmatrix} | & | & | & | & | & | & | & | \\ 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 & \sin(x_1) & \sin(x_2) & \cdots \\ | & | & | & | & | & | & | & | \end{bmatrix} \tag{15}$$

Here, X^{P_q} denotes polynomials in X of the order q.

Table 3 presents the results of the experiments performed on two-variables dependent functions, i.e., $f(x_1, x_2)$. Similarly to Sect. 4.1.1, training and test data sets are generated

Table 3 Results for multivariate functions using linear SR

Benchmark	Expression	L_1		L_2	
		Result	R^2	Result	R^2
Nguyen-12	$x_1^4 - x_1^3 + \frac{1}{2}x_2^2 - x_2$	T	1.0	F	≈ 1
Livermore-5	$x_1^4 - x_1^3 + x_1^2 - x_2$	T	1.0	F	≈ 1
Nguyen-9	$\sin(x_1) + \sin(x_2^2)$	F	≈ 1	F	≈ 1
Nguyen-10	$2\sin(x_1)\cos(x_2)$	F	≈ 1	F	≈ 1

$\mathcal{D} = \{(x_1, x_2) \in U(-1, 1, 20); y = f(x_1, x_2)\}.$ $L_1 = \{x, (\cdot)^2, \cdots, (\cdot)^9\}$ and $L_2 = L_1 \cup \{\sin(\cdot), \cos(\cdot), \tan(\cdot), \exp(\cdot), \text{sigmoid}(\cdot)\}$

T and F true and false

Fig. 5 Two-dimensional multivariate normal distribution used in test applications
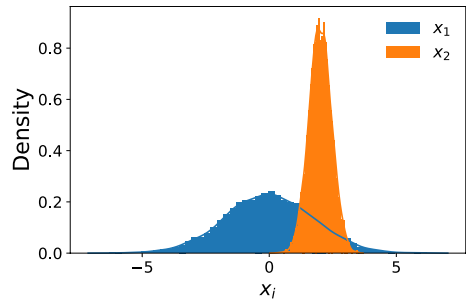

Table 4 Library bases used in test problems of Sect. 4.1.2

	Name	List of functions
Library	U1	$1, X, X^{P2}, X^{P3}, X^{P4}$
	U2	$1, X, X^{P2}, \cos(X), \sin(X)$
	U3	$1, X, X^{P2}, \cos(X), \sin(X), \tan(X), \exp(X), \text{sigmoid}(X)$

by randomly sampling twenty pairs of points (x_1, x_2) from a uniform distribution $U(-1, 1)$ such that $\mathcal{D} = \{(x_{1i}, x_{2i}, f(x_{1i}, x_{2i}))\}_{i=1}^n$. The same choices for the library are considered: $L_1 = \{x, (\cdot)^2, \cdots, (\cdot)^9\}$ and $L_2 = L_1 \cup \{\sin(\cdot), \cos(\cdot), \tan(\cdot), \exp(\cdot), \text{sigmoid}(\cdot)\}$. An exact match between the ground-truth and predicted function is obtained using L_1 for any polynomial function, whereas only approximate solutions are obtained for trigonometric functions. The results are approximate of the ground-truth function using L_2.

Furthermore, linear SR is tested on a dataset generated using a two-dimensional multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, as shown in Fig. 5. Different analytic expressions for $f(x_1, x_2)$ were tested with different library bases that are summarized in Table 4, including pure polynomial basis functions, polynomial and trigonometric basis functions, and a mixed library.
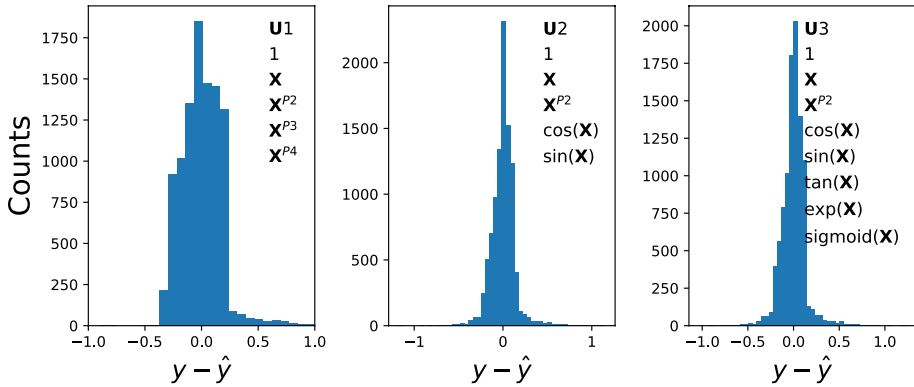
Fig. 6 Difference between true (y) and predicted $(\hat{y})$ values of the function $y = x_1(1 + x_2) + \cos(x_1) * \sin(x_2)$, for the three libraries defined in Table 4: U1 (left), U2 (center), U3 (right)

The function $y_1 = \cos(x_1) + \sin(x_2)$ is explored with all three bases. In the case of a pure polynomial basis, the correct terms of the Taylor expansion of both $\cos(x_1)$ and $\sin(x_2)$ are identified with only approximate values of their coefficients, i.e., $\hat{y}_1 = (0.88 - 0.3x_1^2 + 0.01x_1^4) + (0.97 - 0.2x_2^3)$, which is reflected in the significantly high reconstruction error of the order of 30%. In both bases where trigonometric functions are enlisted, the correct terms $\cos(x_1)$ and $\sin(x_2)$ are identified with an excellent reconstruction error, that is $\geq 10^{-7}$. Note that the lowest reconstruction error is obtained for the library U 2, which has the least number of operations and, consequently, the lowest number of coefficients.

The function $y_2 = x_1^2 + \cos(x_2)$ is also tested. For the pure polynomial basis, the reconstructed function $\hat{y}_2 = x_1^2 + (0.83 + 0.49x_2 - x_2^2)$ predicts approximate values with a reconstruction error of $\leq 1\%$. An excellent prediction is made for both of the other bases, which enlist both operations in $y_2(x_1, x_2)$.

In the same exercise, a more complicated function form is tested that includes mixed terms, i.e., $y_3 = x_1(1 + x_2) + \cos(x_1) * \sin(x_2)$. The difference between the true and the predicted function is illustrated in Fig. 6. The linear approach performs similarly for all three library bases. A low reconstruction error is obtained because the operation term $\cos(x_1) * \sin(x_2)$ in y_3 is not enlisted in any of the libraries, showing an important limitation of the current approach.

4.2 Multidimensional case

The target mathematical expression comprises m components, i.e., $Y = [y_1, \cdots, y_m]$, and the goal is to learn the coefficients of a system of linear equations rather than one mathematical expression. Each component (y_j) is described by:

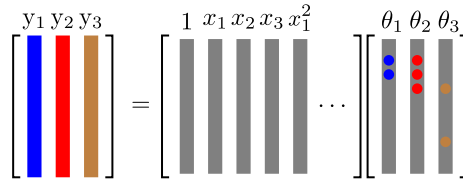$$y_j = f_j(x) = \sum_k \theta_{jk} h_k(x) \tag{16}$$

Fig. 7 Schematic of the system of Eq. 11 for the Lorenz system defined by $y_1 = \sigma(x_2 - x_1)$, $y_2 = x_1(\rho - x_3) - x_2$, $y_3 = x_1 x_2 - \beta x_3$. A library U(X) of nonlinear functions of the input is constructed. The marked entries in the θs vectors denote the non-zero coefficients determining which library functions are active for each of the three variables $\{y_1, y_2, y_3\}$

In this case, there exist m sparse vectors of coefficients, i.e., $\Theta = \begin{bmatrix} \theta_1 & \cdots & \theta_m \end{bmatrix}$. Consider the Lorenz system, which is a set of ordinary differential equations that captures nonlinearities in the dynamics of fluid convection. It consists of three variables $\{x_1, x_2, x_3\}$ and their first-order derivatives with respect to time $\{\frac{dx_1}{dt}, \frac{dx_2}{dt}, \frac{dx_3}{dt}\}$, which we will refer to as $\{y_1, y_2, y_3\}$. Using the library of Eq. 13, the system of linear equations is represented in a matrix form as follows:

$$\begin{bmatrix} y_1 & y_2 & y_3 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 & & \exp(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} \quad (17)$$

Here, $Y \in \mathbb{R}^{n \times 3}$, $U(X) \in \mathbb{R}^{n \times k}$ and $\Theta \in \mathbb{R}^{k \times 3}$, where n is the size of the input data and k is the number of columns in the library matrix U. The jth-component of the Y vector is given by:

$$y_j = \theta_{j,0} + \theta_{j,1} x_1 + \theta_{j,2} x_2 + \theta_{j,3} x_1^2 + \cdots + \theta_{j,k} \exp(x_2) \quad (18)$$

Equation 17 can be written in a compact form as:

$$y_k = U(x^T)\theta_k \quad (19)$$

The application presented in Champion et al. (2019) uses this approach, where the authors aim to learn differential equations that govern the dynamics of a given system, such as a nonlinear pendulum and the Lorenz system. The approach successfully learned the exact weights, allowing them to recover the correct governing equations.

An exemplary schematic is illustrated in Fig. 7 for the Lorenz system defined by $\dot{x} = \sigma(y - x)$, $\dot{y} = x(\rho - z) - y$, $\dot{z} = xy - \beta z$. Here x, y, and z are physical variables and $\dot{x}$, $\dot{y}$, and $\dot{z}$ are their respective time-derivatives. Only coefficients associated with functions $\{x_1, x_1 x_2, \}$ should be non-zero and equal to the factors shown in the Lorenz system's set of equations.

In summary, the linear approach is only successful in particular cases and can not be generalized. Its main limitation is in predefining the model's structure as a linear combination of nonlinear functions, reducing the SR problem to solve a system of linear equations. In contrast, the main mission of SR is to learn the model's structure and parameters. A direct consequence of this limitation is that the linear approach fails to learn expressions in many cases: (i) composition of functions (e.g., $f(x) = f_1(x) * f_2(x)$); (ii) multivariate
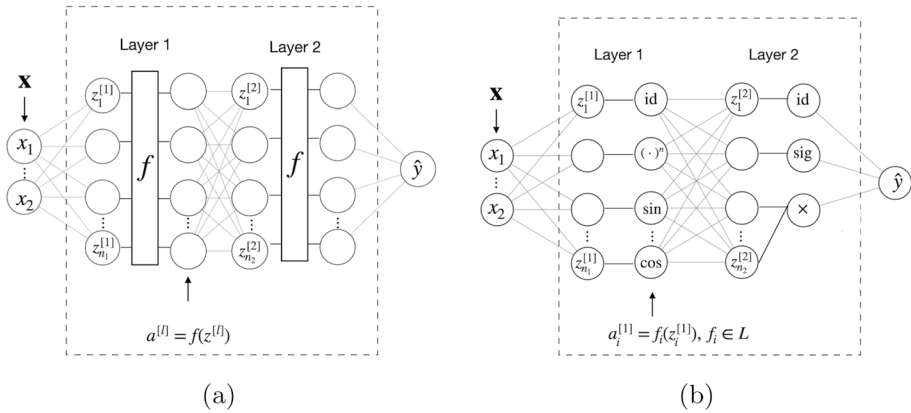
Fig. 8 Exemplary setup of a standard NN (**a**) and EQL-NN (**b**) with input **x**, output $\hat{y}$ and two hidden layers. In **a**, f denotes the activation function usually chosen among {RELU, tanh, sigmoid} while in EQL each node has a specific activation function drawn from the function class $\mathcal{F}$

functions (e.g., $\exp(x * y), \tan(x + y)$, etc.); and (iii) functions including multiplicative or additive factors to their arguments (e.g., $\exp(\lambda x)$). Finally the dimension of the library matrix can be challenging in computing resources for extended libraries and high-dimensional data sets.

5 Nonlinear symbolic regression

The nonlinear method uses deep neural networks (DNN), known for their great ability to detect and learn complex patterns directly from data.

DNN has the advantage of being fully differentiable in its free parameters allowing end-to-end training using back-propagation. This approach searches the target expression by replacing the standard activation functions in a neural network with elementary mathematical operations. Figure 8 shows an NN-based architecture for SR called the Equation Learner (EQL) network proposed by Martius and Lampert (2016) in comparison with a standard NN. Only two hidden layers are shown for simple visualization, but the network's deepness is controlled as per the case study.

The EQL network uses a multi-layer feed-forward NN with one output node. A linear transformation $z^{[l]}$ is applied at every hidden layer (l), followed by a nonlinear transformation $a_i^{[l]}$ using unary (i.e., one argument) and binary (i.e., two arguments) activation functions as follows

$$
\begin{aligned}
z^{[l]} &= W^{[l]} \cdot a^{[l-1]} + b^{[l]} \\
a_i^{[l]} &= f_i(z_i^{[l]})
\end{aligned}
\tag{20}
$$

where $\{W, b\}$ denote the weight parameters and f_i denotes individual activation function from the library $L = \{\text{identity}, (\cdot)^n, \cos, \sin, \exp, \log, \text{sigmoid}\}$. In a standard NN,
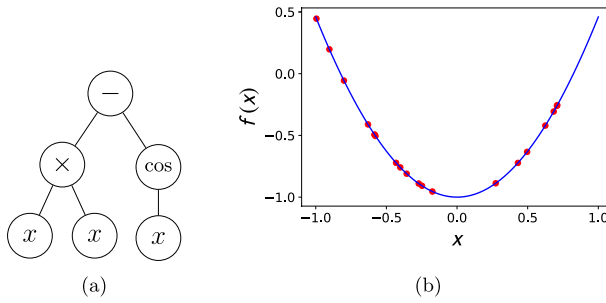
Fig. 9 **a** Expression-tree structure of $f(x) = x^2 - \cos(x)$. **b** $f(x)$ as a function of x (blue curve) and data points (red points) generated using $f(x)$

the same activation function is applied to all hidden units and is typically chosen among {RELU, tanh, sigmoid, softmax, etc.}.

The problem reduces to learn the correct weight parameters $\{W^{[l]}, b^{[l]}\}$, whereas the operators of the target mathematical expression are selected during training. To overcome the interpretability limitation of neural network-based architectures and to promote simple over complex solutions as a typical formula describing a physical process, sparsity is enforced by adding a regularization term l_1 to the l_2 loss function such that,

$$\ell = \frac{1}{N} \sum_{i=1}^{N} \|\hat{y}(x_i) - y_i\|^2 + \lambda \sum_{l=1}^{L} |W^{[l]}|_1 \tag{21}$$

Where N denotes the number of data entries and L denotes the number of layers. Whereas this method is end-to-end differentiable in NN parameters and scales well to high dimensional problems, back-propagation through activation functions such as division or logarithm requires simplifications to the search space, thus limiting its ability to produce simple expressions involving divisions (e.g., $\frac{\sin(x/y)}{x}$). An extended version EQL$^{\div}$ (Sahoo et al. 2018) includes only the division, whereas exponential and logarithm activation functions are not included because of numerical issues.

6 Tree expression

This section discusses SR methods in which a mathematical expression is regarded as a unary-binary tree consisting of internal nodes and terminals. Every tree node represents a mathematical operation (e.g., $+, -, \times, \sin, \log,$ etc.) that is drawn from a pre-defined library and every tree terminal node (or leaf) represents an operand, i.e., variable or constant, as illustrated for the example shown in Fig. 9. Expression tree-based methods include genetic programming, transformers, and reinforcement learning.

6.1 Genetic programming

Genetic programming (GP) is an evolutionary algorithm in computer science that searches the space of computer programs to solve a given problem. Starting with a
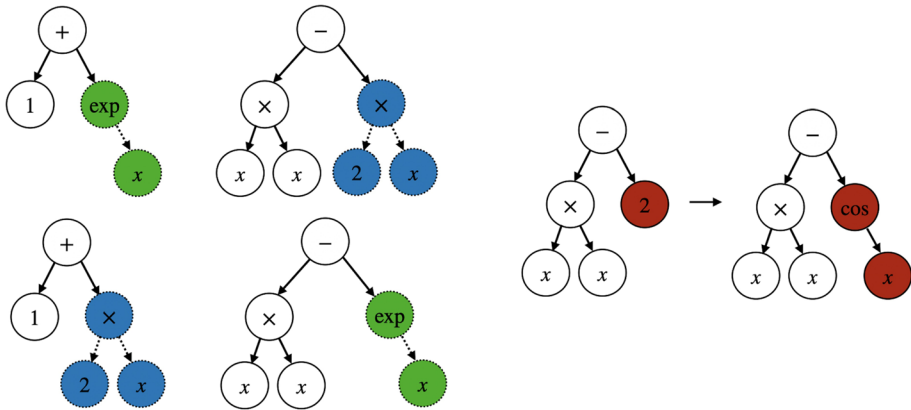
Fig. 10 Crossover (left) and mutation (right) operations on exemplary expression trees in genetic programming

"population" (set) of "individuals" (trees) that is randomly generated, GP evolves the initial population $\mathcal{T}_{GP}^{(0)}$ using a set of evolutionary "transition rules" (operations) $\{r_i : f \rightarrow f \mid i \in \mathbb{N}\}$ that is defined over the tree space. GP evolutionary operations include mutation, crossover, and selection. The mutation operation introduces random variations to an individual by replacing one subtree with another randomly generated subtree (Fig. 10, right). The crossover operation involves exchanging content between two individuals, for example, by swapping one random subtree of one individual with another random subtree of another individual (Fig. 10, left). Finally, the selection operation is used to select which individuals from the current population persist onto the next population. A common selection operator is tournament selection, in which a set of k candidate individuals are randomly sampled from the population, and the individual with the highest fitness i.e., a minimum loss is selected. In a GP algorithm, a single iteration corresponds to one generation. The application of one generation of GP on a population $\mathcal{T}_{GP}^{(i)}$ produces a new, augmented population $\mathcal{T}_{GP}^{(i+1)}$. In each generation, each individual has a probability of undergoing a mutation operation and a probability of undergoing a crossover operation. The selection is applied when the dimension of the current population is the same as the previous one. Throughout M_k iterations, the following steps are undertaken: (1) transition rules are applied to the function set $F^k = \{f_1^k, \cdots, f_{M_k}^k\}$ such that $f^{k+1} = r_i(f^k)$ where k denotes the iteration index; (2) the loss function $\ell(F^k)$ is evaluated for the set; and (3) an elite set of individuals is selected for the next iteration step. The GP algorithm repeats this procedure until a pre-determined accuracy level is achieved.

Whereas GP allows for large variations in the population resulting in improved performance for out-of-distribution data, GP-based methods do not scale well to high dimensional data sets and are highly sensitive to hyperparameters (Petersen 2019).
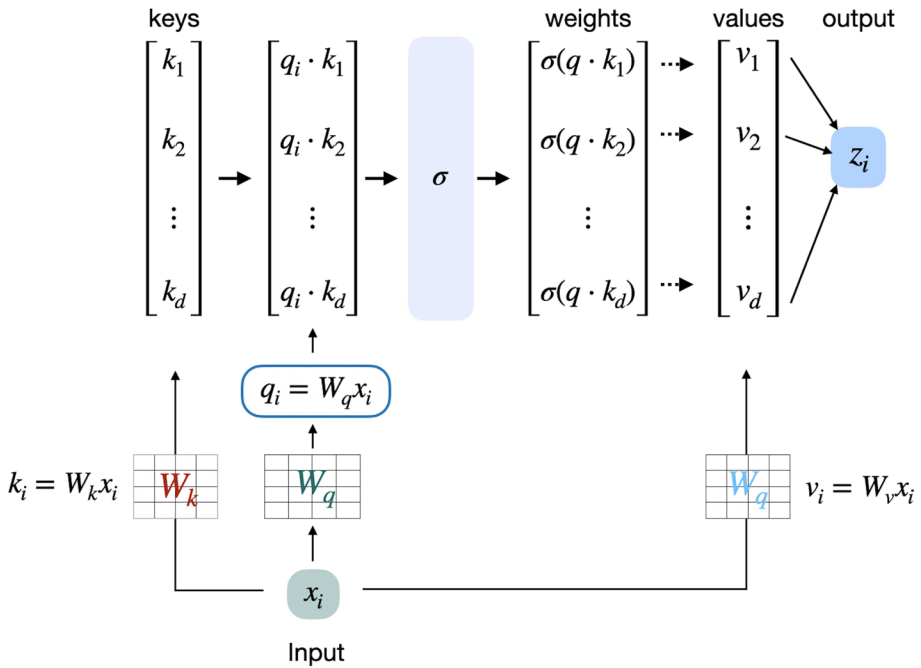
Fig. 11 Evaluation of Attention$(q, \mathcal{S})$ (Eq. 22) for a query q_i, computed using the input vector embedding x_i

6.2 Transformers

Transformer neural network (TNN) is a novel NN architecture introduced by Vaswani et al. (2017) in natural language processing (NLP) to model sequential data. TNN is based on the attention mechanism that aims to model long-range dependencies in a sequence. Consider the English-to-French translation of the two following sentences:

En: The kid did not go to school because **it** was closed.
Fr: L'enfant n'est pas allé à l'école parce qu'elle était fermée.
En: The kid did not go to school because **it** was cold.
Fr: L'enfant n'est pas allé à l'école parce qu'il faisait froid.

The two sentences are identical except for the last word, which refers to the school in the first sentence (i.e., "**closed**") and to the weather in the second one (i.e., "**cold**"). Transformers create a context-dependent word embedding that it pays particular attention to the terms (of the sequence) with high weights. In this example, the noun that the adjective of each sentence refers to has a significant weight and is therefore considered for translating the word "it". Technically, an embedding x_i is assigned to each element of the input sequence, and a set of m key-value pairs is defined, i.e., $\mathcal{S} = \{(k_1, v_1), \cdots, (k_m, v_m)\}$. For each query, the attention mechanism computes a linear combination of values $\sum_j \omega_j v_j$, where the attention weights ($\omega_j \propto q \cdot k_j$) are derived using the dot product between the query (q) and all keys (k_j), as follows:

Fig. 12 Structure of a TNN encoder (Vaswani et al. 2017). It comprises an attention layer and a feed-forward neural network
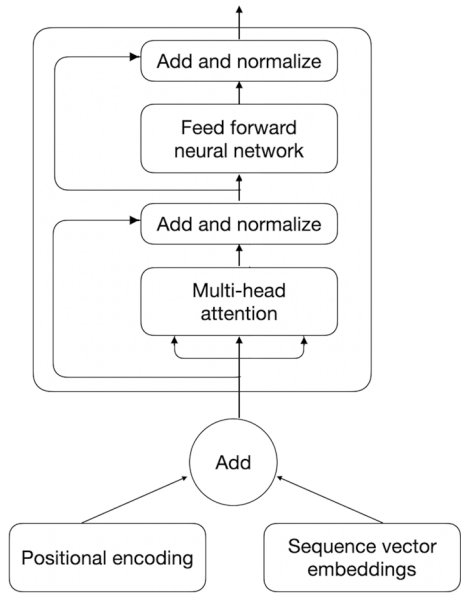
$$\text{Attention}(q, \mathcal{S}) = \sum_j \sigma(q \cdot k_j) v_j \tag{22}$$

Here, $q = x W_q$ is a query, $k_i = x_i W_k$ is a key, $v_i = x_i W_v$ is a value, and W_q, W_k, W_v are the learnable parameters. The architecture of the self-attention mechanism is illustrated in Fig. 11.

In the context of SR, both input data points $\{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i \in \mathbb{N}_n\}$ and mathematical expressions f are encoded as sequences of symbolic representations as discussed in Sect. 2.2. The role of the transformer is to create the dependencies at two levels, first between numerical and symbolic sequences and between tokens of symbolic sequence. Consider the mathematical expression $f(x, y, z) = \sin(x/y) - \sin(z)$, which can be written as a sequence of tokens following the polish notation:

$$\boxed{-}\;\boxed{\sin}\;\boxed{\div}\;\boxed{x}\;\boxed{y}\;\boxed{\sin}\;\boxed{z}$$

Each symbol is associated with an embedding such that:

$$x_1 : -\quad x_2 : \sin\quad x_3 : \div\quad x_4 : x\quad x_5 : y\quad x_6 : \sin\quad x_7 : z$$

In this particular example, for query $(x_7 : z)$, the attention mechanism will give a higher weight for the binary operator $(x_1 : -)$ than for the variable $(x_5 : y)$ or the division operator $(x_3 : \div)$.

Transformers consist of an encoder-decoder structure; each block comprises a self-attention layer and a feed-forward neural network. TNN inputs a sequence of embeddings $\{x_i\}$ and outputs a "context-dependent" sequence of embeddings $\{y_i\}$ one at a time, through a latent representation z_i. TNN is an auto-regressive model, i.e., sampling each symbol is
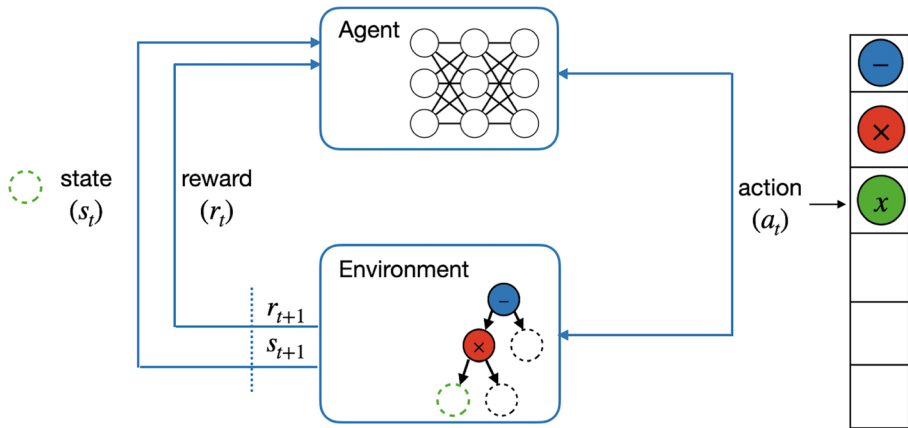
Fig. 13 Exemplary sketch of a general RL-based SR method. s_t, a_t, and $r_t = \mathcal{R}(s_t, a_t)$ denote the state, action, and reward at time step t. $(t + 1)$ denotes the next time step
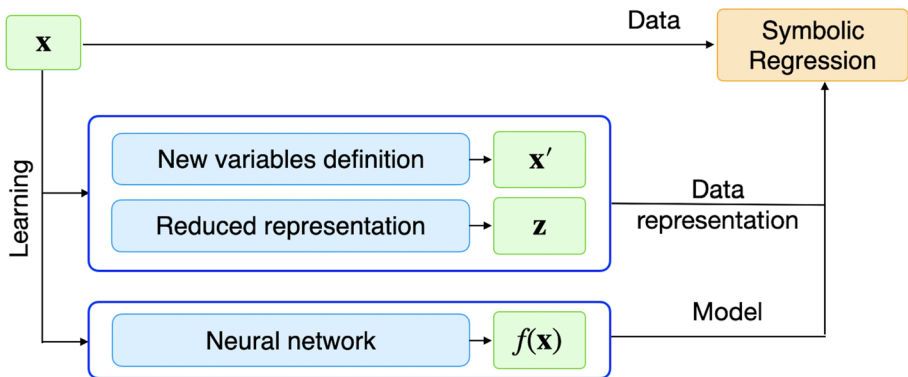

Fig. 14 Strategies for solving SR problem. An SR algorithm has three types of input: data (x), a new or reduced representation of the data (x′ or z), or a model ($f(x)$) learned from the data

conditioned by the previously sampled symbols and the latent sequence. An example of a TNN encoder is shown in Fig. 12.

In symbolic regression case, the encoder and the decoder do not share the same vocabulary because the decoder has a mixture of symbolic and numeric representations, while the encoder has only numeric representations. There exist two approaches to solving SR problems using transformers. First is the skeleton approach (Biggio et al. 2021; Valipour et al. 2021) where the transformer conducts the two-steps procedure: (1) the decoder predicts a skeleton f_e, a parametric function that defines the general shape of the target expression up to a choice of constants, using the function class $\mathcal{F}$ and (2) the constants are fitted using optimization techniques such as the non-linear optimization solver BFGS. For example, if $f = \cos(2x_1) - 0.1\exp(x_2)$, then the decoder predicts $f_e = \cos(\circ\, x_1) - \circ\exp(x_2)$ where $\circ$ denotes an unknown constant.

The second is an end-to-end (E2E) approach (Kamienny et al. 2022) where both the skeleton and the numerical values of the constants are simultaneously predicted. Both approaches are further discussed in Sect. 7.

6.3 Reinforcement learning

Reinforcement learning provides a framework for learning and decision-making by trial and error (Sutton and Barto 2018). An RL Setting consists of four components $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ in a Markov decision process. In this setting, an agent observes a state $s \in \mathcal{S}$ of the environment and, based on that, takes action $a \in \mathcal{A}$, which results in a reward $r = \mathcal{R}(s, a)$, and the environment then transitions to a new state $s' \in \mathcal{S}$. The interaction goes on in time steps until a terminal state is reached. The aim of the agent is to learn the policy $\mathcal{P}$ (also called transition dynamics), which is a mapping from states to actions that maximize the expected cumulative reward. An exemplary sketch of an RL-based SR method is illustrated in Fig. 13.

SR problem can be framed in RL as follows: the agent (NN) observes the environment (parent and sibling in a tree) and, based on the observation, takes an action (predict the next token of the sequence) and transitions into a new state. In this view, the NN model is like a policy, the parent and sibling are like observations, and sampled symbols are like actions.

7 Applications

Most existing algorithms for solving SR are GP-based, whereas many others, and more recent, are deep learning (DL)-based. There exist two different strategies to solve SR problems, as illustrated in the taxonomy of Fig. 14.

The first is a one-step approach, where data points are directly fed into an SR algorithm. A second is a two-step approach involving a process which either learns a new representation of data or learns a *"blackbox"* model, which will be then fed into SR algorithm as described below:

1. Learn a new representation of the original data set through defining new features (reducing the number of independent variables) or a reduced representation using specific NN architectures such as principal component analysis and autoencoders.
2. Learn a *"blackbox"* model either using regular NN or using conceptual NN such as graph neural network (GNN). In this case, an SR algorithm is applied to the learned model or parts of it.

We group the applications based on the categories presented in Sect. 3, and we summarize them in Table 5.

GP-based applications will not be reviewed here; they are listed in the living review (Makke and Chawla 2022), along with DL-based applications. State-of-the-art GP-based methods are discussed in detail in La Cava et al. (2016). Among GP-based applications is the commercial software Eureqa (Dubcakova 2011), the most well-known

Table 5 Table summarizing symbolic regression applications

Application	Ref	Name/description(code)	Year	Method	Strategy
SINDY	Brunton et al. (2016)	Sparse Identification of Nonlinear Dynamics faculty.washington.edu/sbrunton/sparsedynamics.zip	2016	Linear	D
SINDY-AE	Champion et al. (2019)	Data-driven discovery of coordinates and governing equations (https://github.com/kpchamp/SindyAutoencoders)	2019	Linear	1
EQL	Martius and Lampert (2016)	Equation learner (https://github.com/KristofPusztai/EQL)	2016	non linear	D
EQL÷	Sahoo et al. (2018)	Equation learner division (https://github.com/martius-lab/EQL)	2018	Non linear	D
Eureqa	Dubcakova (2011)	Commercial software	2011	GP	D
FFX	McConaghy (2011)	Fast function extraction (https://github.com/natekupp/ffx/tree/master/ffx)	2011	GP	D
ITEA	de França and Aldeia (2019)	Interaction-Transformation Evolutionary Algorithm for SR (https://github.com/folivetti/ITEA/)	2019	GP	D
MRGP	Arnaldo et al. (2014)	Multiple Regression GP (https://github.com/flexgp/gp-learners)	2014	GP	D
E2ESR	Kamienny et al. (2022)	End-to-end SR with transformers (https://github.com/facebookresearch/symbolicregression)	2022	TNN	D
NeSymReS	Biggio et al. (2021)	Neural SR that scales (https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales)	2021	TNN	D
DSR	Petersen (2019)	Deep symbolic regression (https://github.com/brendenpetersen/deep-symbolic-regression)	2019	RNN,RL	D
NGPPS	Mundhenk et al. (2021)	SR via Neural-Guided GP population seeding (https://github.com/brendenpetersen/deep-symbolic-regression)	2021	RNN,GP,RL	D
AIFeynman	Udrescu and Tegmark (2019)	Physics-inspired method for SR (https://github.com/SJ001/AI-Feynman)	2019	Physics-informed	1
SM	Alaa and Schaar (2019)	Symbolic Metamodel (https://bitbucket.org/mvdschaar/mlforhealthlabpub)	2019	Mathematics	2
GNN	Cranmer et al. (2020)	Discovering Symbolic Models from DL with Inductive Biases (https://github.com/MilesCranmer/symbolic_deep_learning)	2020	GNN	2
HEAL	https://github.com/heal-research	Heuristic and Evolutionary Algorithms Laboratory (https://github.com/heal-research/HeuristicLab)	–	Heuristic	D

D refers to data input, 1 refers to data representation input, and 2 refers to model input to SR

Fig. 15 Example of a Meijer G-function $G^{2,2}_{1,1}\binom{a,a}{a,b}|x)$ for different values of a and b (Alaa and Schaar 2019)
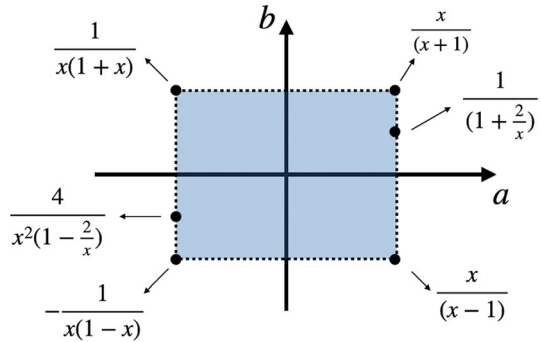
GP-based method that uses the algorithm proposed by Schmidt and Lipson (2009). Eureqa is used as a baseline SR method in several research works.

SINDY-AE (Champion et al. 2019) is a hybrid SR method that combines autoencoder network (Rumelhart et al. 1986) with linear SR (Brunton et al. 2016). The novelty of this approach is in simultaneously learning sparse dynamical models and reduced representations of coordinates that define the model using snapshot data. Given a data set $\mathbf{x}(t) \in \mathbb{R}^n$, this method seeks to learn coordinate transformations from original to intrinsic coordinates $\mathbf{z} = \phi(\mathbf{x})$ (encoder) and back via $\mathbf{x} = \psi(\mathbf{z})$ (decoder), along with the dynamical model associated with the set of reduced coordinates $\mathbf{z}(t) \in \mathbb{R}^d$ ($d \ll n$):

$$\frac{d}{dt}\mathbf{z}(t) = \mathbf{g}(\mathbf{z}(t)) \tag{23}$$

through a customized loss function $\mathcal{L}$, defined as a sum of four terms:

$$\mathcal{L} = \underbrace{\|\mathbf{x} - \psi(\phi(\mathbf{x}))\|_2^2}_{\text{reconstruction error}} + \lambda_1 \underbrace{\|\dot{\mathbf{z}} - \dot{\mathbf{z}}_{\text{pred}}\|_2^2}_{\text{encoder loss}} + \lambda_2 \underbrace{\|\dot{\mathbf{x}} - \dot{\mathbf{x}}_{\text{pred}}\|_2^2}_{\text{decoder loss}} + \underbrace{\lambda_3 \|\Theta\|_1}_{\text{regularizer loss}} \tag{24}$$

Here the derivative of the reduced variables $\mathbf{z}$ are computed using the derivatives of the original variable $\mathbf{x}$, i.e. $\dot{\mathbf{z}} = \nabla_{\mathbf{x}}\phi(\mathbf{x})\dot{x}$. Predicted coordinates denoted as $\mathbf{a}_{\text{pred}}$ represent NN outputs and are expressed in terms of coefficient vector Θ and library matrix $\mathbf{U}(\mathbf{x})$ following Eq. 19, i.e., $\mathbf{z}_{\text{rec}} = \mathbf{U}(\mathbf{z}^T)\Theta = \mathbf{U}(\phi(\mathbf{x})^T)\Theta$. The library is specified before training, and the coefficients Θ are learned with the NN parameters as part of the training procedure.

A case study is the nonlinear pendulum motion whose dynamics are governed by a second-order differential equation given by $\ddot{x} = -\sin(x)$. The data set is generated as a series of snapshot images from a simulated video of a nonlinear pendulum. After training, the SINDY autoencoder correctly identified the equation $\ddot{z} = -0.99 \sin z$, which is the dynamical model of a nonlinear pendulum in the reduced representation. This approach is particularly efficient when the dynamical model may be dense in terms of functions of the original measurement coordinates $\mathbf{x}$. This method and similar works (Chen et al. 2021) make the path to "Gopro physics" where researchers point a camera on an event and get back an equation capturing the underlying phenomenon using an algorithm.

Despite successful applications involving partial differential equations, still, one main limitation of this method is in its linear SR part. For example, a model expressed as $f(\mathbf{x}) = x_1 x_2 - 2x_2 \exp(-x_3) + \frac{1}{2} \exp(-2x_1 x_3)$ is discovered only if each term of this

expression is comprised in the library, e.g., $\exp(-2x_1x_2)$. The presence of the exponential function, i.e., $\exp(x)$, is not sufficient to discover the second and the third terms.

Symbolic metamodel (Alaa and Schaar 2019) (SM) is a *model-of-a-model* method for interpreting "blackbox" model predictions. It inputs a learned *"blackbox"* model and outputs a symbolic expression. Available post-hoc methods aim to explain ML model predictions, i.e., they can explain some aspects of the prediction but can not offer a full model interpretation. In contrast, SM is interpretable because it uncovers the functional form that underlies the learned model. The symbolic metamodel is based on Meijer G-function (Meijer 1946; Beals and Szmigielski 2013), which is a special univariate function characterized by a set of indices, i.e., $G_{p,q}^{m,n}(\mathbf{a}_p, \mathbf{b}_q|x)$, where $\mathbf{a}$ and $\mathbf{b}$ are two sets of real-values parameters. An instance of the Meijer G-function is specified by $(\mathbf{a}, \mathbf{b})$, for example the function $G_{2,2}^{1,2}(_{a,b}^{a,a}|x)$ takes different forms for different settings of the parameters a and b, as illustrated in Fig. 15.

In the context of SR problem solving, the target mathematical expression is defined as a parameterization of the Meijer function, i.e., $\{g(x) = G(\theta, \mathbf{x}) \mid \theta = (\mathbf{a}, \mathbf{b})\}$, thus reducing the optimization task to a standard parameter optimization problem that can be efficiently solved using gradient descent algorithms $\theta^{k+1} := \theta^k - \gamma \sum_i l(G(\mathbf{x}_i, \theta), f(\mathbf{x}_i))|_{\theta=\theta^k}$. The parameters $\mathbf{a}$ and $\mathbf{b}$ are learned during training, and the indices (m, n, p, q) are regarded as hyperparameters of the model. SM was tested on both synthetic and real data and was deployed in two modes spanning (1) only polynomial expressions (SM^p) and (2) closed-form expressions (SM^c), in comparison to a GP-based SR method. SM^p produces accurate polynomial expressions for three out of four tested functions (except the Bessel function), whereas SM^c produces the correct ground-truth expression for all four functions and significantly outperforms GP-based SR.

More generally, consider a problem in a critical discipline such as healthcare. Assuming a feature vector comprising (age, gender, weight, blood pressure, temperature, disease history, profession, etc.) with the aim to predict the risk of a given disease. Predictions made by a *"blackbox"* could be highly accurate. However, the learned model does not provide insights into why the risk is high or low for a patient and what parameter is the most critical or weightful in the prediction. Applying the symbolic metamodel to the learned model outputs a symbolic expression, e.g., $f(x_1, x_2) = x_1(1 - \exp(-x_2))$, where x_1 is the blood pressure and x_2 is the age. Here, we can learn that only two features (out of many others) are crucial for the prediction and that the risk increases with high blood pressure and decreases with age. This is an ideal example showing the difference between *"blackbox"* and interpretable models. In addition, it is worth mentioning that methods applied for model interpretation only exploit part of the prediction and can not unveil how the model captures nonlinearities in the data. Thus model interpretation methods are insufficient to provide full insights into why and how model predictions are made and are not by any means equivalent to interpretable models.

End-to-end symbolic regression (Kamienny et al. 2022) (E2ESR) is a transformer-based method that uses end-to-end learning to solve SR problems. It is made up of three components: (1) an embedder that maps each input point (x_i, y_i) to a single embedding, (2) a fully-connected feedforward network, and (3) a transformer that outputs a mathematical expression. What distinguishes E2ESR from other transformer-based applications is the use of an end-to-end approach without resorting to skeletons, thus using both symbolic representations for the operators and the variables and numeric representations for the constants. Both input data points $\{(\mathbf{x}_i, y_i) \mid i \in \mathbb{N}_n\}$ and mathematical expressions f are encoded as sequences of symbolic representations following the description in Sect. 2.2. E2ESR is
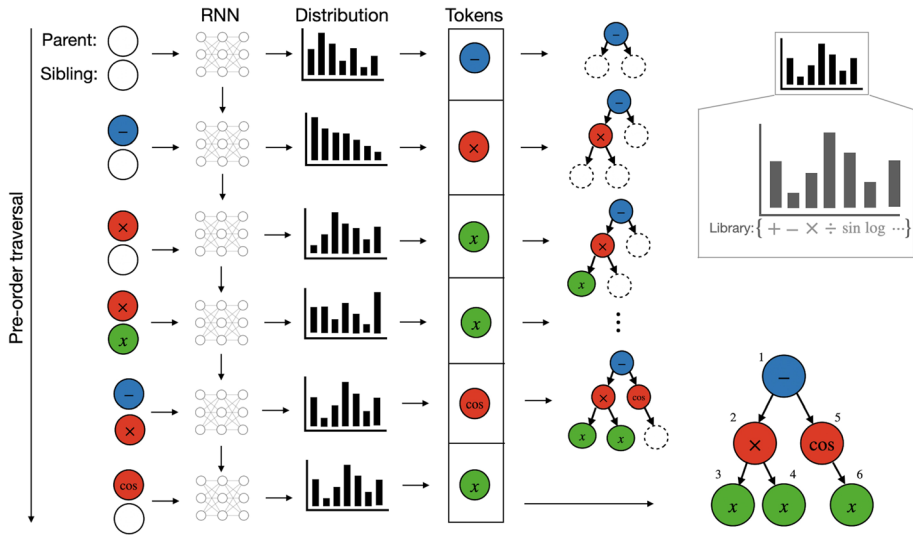
Fig. 16 Exemplary sketch of RNN generating a mathematical expression $x^2 - \cos(x)$

tested and compared to several GP-based and DL-based applications on SR benchmarks. Results are reported in terms of mean accuracy, formula complexity, and inference time, and it was shown E2ESR achieves very competitive results for SR and outperforms previous applications.

AIFeynman (Udrescu and Tegmark 2019) is a physics-inspired SR method that recursively applies a set of solvers, i.e., dimensional analysis,[3] polynomial fit, and brute-force search to solve an SR problem. If the problem is not solved, the algorithm searches for simplifying intrinsic properties in data (e.g. invariance, factorization) using NN and deploys them to recursively simplify the dataset into simpler sub-problems with fewer independent variables. Each sub-problem is then tackled by a symbolic regression method of choice. The authors created the Feynman SR database (see Sect. 8) to test their approach. All the basic equations and 90% of the bonus equations were solved by their algorithm, outperforming Eureqa.

Deep symbolic regression (DSR) (Petersen 2019) is an RL-based search method for symbolic regression that uses a generative recurrent neural network (RNN). RNN defines a probability distribution ($p(\theta)$) over mathematical expressions (τ), and batches of expressions $\mathcal{T} = \{\tau^{(i)}\}_{i=1}^{N}$ are stochastically generated. An exemplary sketch of how RNN generates an expression (e.g., $x^2 - \cos(x)$) is shown in Fig. 16. Starting with the first node following the pre-order traversal (Sect. 2.2) of an expression tree, RNN is initially fed with empty placeholders tokens (a parent and a sibling) and produces a categorical distribution, i.e., outputs the probability of selecting every token from the defined library $L = \{+, -, \times, \div, \sin, \cos, \log, \text{etc.}\}$. The sampled token is fed into the first node, and the number of siblings is determined based on whether the operation is unary (one sibling) or

---

[3] Dimensional analysis is a well-known technique in physics that uses set of units of measurements to solve an equation and/or to check the correctness of a given equation.

Fig. 17 Neural-guided genetic programming population seeding method overview (Mundhenk et al. 2021)
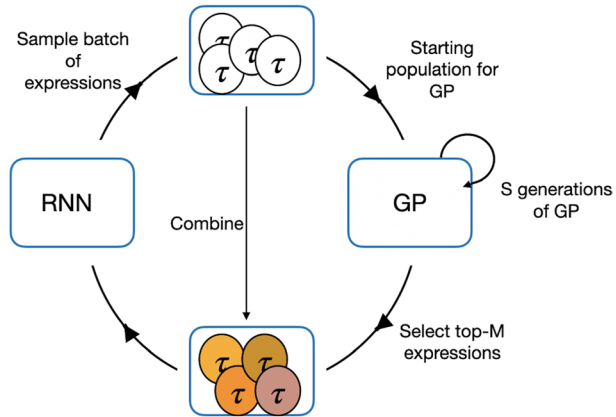
binary (two siblings). The second node is then selected, and the RNN is fed with internal weights along with the first token and outputs a new (and potentially different) categorical distribution. This procedure is repeated until the expression is complete. Expressions are then evaluated with a reward function $R(\tau)$ to test the goodness of the fit to the data $\mathcal{D}$ for each candidate expression (f) using normalized root-mean-square error, $R(\tau) = 1/\left(1 + \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(X_i))^2}\right)$.

To generate better expressions (f), the probability distribution $p(\tau|\theta)$ needs to be optimized. Using a gradient-based approach for optimization requires the reward function $R(\tau)$ to be differentiable with respect to the RNN parameter θ, which is not the case. Instead, the learning objective is defined as the expectation of the reward under expressions from the policy, i.e., $J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)]$, and reinforcement learning is used to maximize $J(\theta)$ by means of the "standard policy gradient":

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)] = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)\nabla_\theta \log p(\tau|\theta)] \tag{25}$$

This reinforcement learning trick, called REINFORCE (Williams 1992), can be derived using the definition of the expectation $\mathbb{E}[\cdot]$ and the derivative of $\log(\cdot)$ function as follows:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)] &= \nabla_\theta \int R(\tau) p(\tau|\theta) d\theta \\
&= \int R(\tau) \nabla_\theta p(\tau|\theta) d\theta \\
&= \int R(\tau) \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)} p(\tau|\theta) d\theta \\
&= \int R(\tau) \log(p(\tau|\theta) p(\tau|\theta) d\theta \\
&= \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)\nabla_\theta \log p(\tau|\theta)]
\end{aligned} \tag{26}$$

The importance of this result is that it allows estimating the expectation using samples from the distribution. More explicitly, the gradient of $J(\theta)$ is estimated by computing the mean over a batch of N sampled expressions as follows:
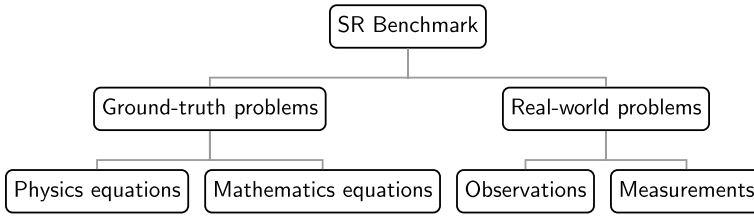
```
                        ┌──────────────────┐
                        │   SR Benchmark   │
                        └──────────────────┘
              ┌───────────────────┴────────────────────┐
    ┌─────────────────────┐                  ┌─────────────────────┐
    │ Ground-truth problems│                  │ Real-world problems │
    └─────────────────────┘                  └─────────────────────┘
      ┌──────────┴───────────┐              ┌────────┴─────────┐
┌──────────────┐ ┌────────────────────┐ ┌──────────────┐ ┌──────────────┐
│Physics equations│ │Mathematics equations│ │ Observations │ │ Measurements │
└──────────────┘ └────────────────────┘ └──────────────┘ └──────────────┘
```

Fig. 18 Taxonomy based on the type of SR benchmark problems

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} R(\tau^{(i)}) \nabla_\theta \log p(\tau^{(i)}|\theta) \tag{27}$$

The standard policy gradient (Eq. 25) permits optimizing a policy's average performance over all samples from the distribution. Since SR requires maximizing best-case performance, i.e., to optimize the gradient over the top ϵ fraction of samples from the distribution found during training, a new learning objective is defined as a conditional expectation of rewards above the $(1 - \epsilon)$-quantile of the distribution of rewards, as follows:

$$J_{\text{risk}}(\theta, \epsilon) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau) \mid R(\tau) \geq R_\epsilon(\theta)] \tag{28}$$

where $R_\epsilon(\theta)$ represent the samples from the distribution below the ϵ-threshold. The gradient of the new learning objective is given by:

$$\nabla_\theta J_{\text{risk}}(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[(R(\tau) - R_\epsilon(\theta)) \cdot \nabla_\theta \log p(\tau|\theta) \mid R(\tau) \geq R_\epsilon(\theta)] \tag{29}$$

DSR was essentially evaluated on the Nguyen SR benchmark and several additional variants of this benchmark. An excellent recovery rate was reported for each set, and DSR solved all mysteries except the Nguyen-12 benchmark given by $x^4 - x^3 + \frac{1}{2}y^2 - y$. More details on SR data benchmarks can be found in Sect. 8.

Neural-guided genetic programming population seeding (Mundhenk et al. 2021) (NGPPS) is a hybrid method that combines GP and RNN (Petersen 2019) and leverages the strengths of each of the two components. Whereas GP begins with random starting populations, the authors in Mundhenk et al. (2021) propose to use the batch of expressions sampled by RNN as a staring population for GP: $\mathcal{T}_{GP}^{(0)} = \mathcal{T}_{RNN}$. Each iteration of the proposed algorithm consists of 4 steps: (1) The batch of expressions sampled by RNN is passed as a starting population to GP, (2) S generations of GP are performed and result in a final GP population $\mathcal{T}_{GP}^S$, (3) An elite set of top-performing GP samples is selected $\mathcal{T}_{GP}^E$ and passed to the gradient update of RNN (Fig. 17).

Neural symbolic regression that scales (Biggio et al. 2021) (NeSymReS) is a transformer-based algorithm that emphasizes large-scale pre-training. It comprises a pre-training and test phase. Pre-training includes data generation and model training. Hundreds of millions of training examples are generated for every minibatch in pre-training. Each training example consists of a symbolic equation f_e and a set of n input–output pairs $\{\mathbf{x}_i, y_i = f(\mathbf{x}_i)\}$ where n can vary across examples, and the number of independent input variables is at most three. In the test phase, a set of input–output pairs $\{x_i, y_i\}$ is fed into the encoder that maps it into a latent vector z, and the

Table 6 Table summarizing ground-truth problems for symbolic regression

Type	Benchmark	Number of problems	Year	Reference
Physics-related	Feynman Database	119	2019	Udrescu and Tegmark (2019)
	Strogatz Repository	10	2011	La Cava et al. (2016)
Mathematics-related	Koza	3	1994	Koza (1994)
	Keijzer	15	2003	Keijzer (2003)
	Vladislavleva	8	2009	Vladislavleva et al. (2009)
	Nguyen	12	2011	Uy et al. (2010)
	Korns	15	2011	Korns (2011)
	R	3	2013	Krawiec and Pawlak (2013)
	Jin	6	2019	Jin et al. (2019)
	Livermore	22	2021	Petersen (2019)

decoder iteratively samples candidates' skeletons. What distinguishes this method is the learning task, i.e., it improves over time with experience, and there is no need to be retrained from scratch on each new experiment. It was shown that NeSymReS outperforms selected baselines (including DSR) in time and accuracy by a large margin on all datasets (AI-Feynman, Nguyen, and strictly out-of-sample equations (SOOSE) with and without constants). NeSymReS is more than three orders of magnitudes faster at reaching the same maximum accuracy as GP while only running on CPU.

GNN (Cranmer et al. 2020) is a hybrid scheme performing SR by training a Graph Neural Network (GNN) and applying SR algorithms on GNN components to find mathematical equations.

A case study is Newtonian dynamics which describes the dynamics of particles in a system according to Newton's laws of motion. $\mathcal{D}$ consists of an N-body system with known interaction (force law F such as electric, gravitation, spring, etc.), where particles (nodes) are characterized by their attributes (mass, charge, position, velocity, and acceleration) and their interaction (edges) are assigned the attribute of dimension 100. The GNN functions are trained to predict instantaneous acceleration for each particle using the simulated data and then applied to a different data sample. The study shows that the most significant edge attributes, say $\{e_1, e_2\}$, fit to a linear combination of the true force components, $\{F_1, F_2\}$, which were used in the simulation showing that edge attributes can be interpreted as force laws. The most significant edge attributes were then passed into Eureqa to uncover analytical expressions that are equivalent to the simulated force laws. The proposed approach was also applied to datasets in the field of cosmology, and it discovered an equation that fits the data better than the existing hand-designed equation.

The same group has recently succeeded in inferring Newton's law for gravitational force using GNN and *PySR* for symbolic regression task (Lemos et al. 2022). GNN was trained using observed trajectories (position) of the Sun, planets, and moons of the solar system collected during 30 years. The SR algorithm could correctly infer Newton's formula that describes the interaction between masses, i.e., $F = -GM_1M_2/r^2$, and the masses and the gravitational constant as well.

8 Datasets

For symbolic regression purposes, there exist several benchmark data sets that can be categorized into two main groups: (1) ground-truth problems (or synthetic data) and (2) real-world problems (or real data), as summarized in Fig. 18. In this section, we describe each category and discuss its main strength and limitations.

Ground-truth regression problems are characterized by known mathematical equations, they are listed in Table 6. These include (1) physics-inspired equations (Udrescu and Tegmark 2019; La Cava et al. 2016) and (2) real-valued symbolic equations (Koza 1994; Keijzer 2003; Vladislavleva et al. 2009; Korns 2011; Uy et al. 2010; Jin et al. 2019; Petersen 2019; Krawiec and Pawlak 2013).

The Feynman Symbolic Regression Database (Tegmark 2019) is the largest SR database that originates from Feynman lectures on Physics series (Feynman et al. 2011, 2006) and is proposed in Udrescu and Tegmark (2019). It consists of 119[4] physics-inspired equations that describe static physical systems and various physics processes. The proposed equations depend on at least one variable and, at most, nine variables. Each benchmark (corresponding to one equation) is generated by randomly sampling one million entries. Each entry is a row of randomly generated input variables, which are sampled uniformly between 1 and 5. This range of sampling was slightly adjusted for some equations to avoid unphysical results (e.g., division by zero or the square root of a negative number). The output is evaluated using function f, e.g. $\mathcal{D} = \{\mathbf{x}_i \in \mathbb{R}^d, y_i = f(x_1, \cdots, x_d)\}$.

This benchmark is rich in proposing various theoretical formulae. Still, it suffers a few limitations: (1) there is no distinction between variables and constants, i.e., constants are randomly sampled and, in some cases, in domains extremely far from physical values. For example, the speed of light is sampled from a uniform distribution $\mathcal{U}(1, 20)$ whereas its physical value is orders of magnitude higher, i.e., $c = 2.988 \times 10^8$ m/s, and the gravitational constant is sampled from $\mathcal{U}(1, 2)$ whereas its physical value is orders of magnitude smaller, $G = 6.6743 \times 10^{-11}$ m^3 kg^{-1} s^{-2}, among others (e.g., vacuum permittivity $\epsilon \sim 10^{-12}$, Boltzmann constant $k_b \sim 10^{-23}$, Planck constant $h \sim 10^{-34}$). (2) Some variables are sampled in nonphysical ranges. For example, the gravitational force is defined between two masses distant by r as $F = Gm_1m_2/r^2$. This force is weak unless defined between significantly massive objects (e.g., the mass of the earth is $M_e = 5.9722 \times 10^{24}$ kg) whereas m_1 and m_2 are sampled in $\mathcal{U}(1, 5)$ in the Feynman database. (3) Some variables are treated as floats while they are integers, and (4) many equations are duplicates of each other (e.g., a multiplicative function of two variables $f(x, y) = x * y$) or have similar functional forms.

The ODE-Strogatz repository (La Cava et al. 2016) consists of ten physics equations that describe the behavior of dynamical systems which can exhibit chaotic and/or nonlinear behavior. Each dataset is one state of a two-state system of ordinary differential equations.

Within the same category, there exist several benchmarks (Koza 1994; Keijzer 2003; Vladislavleva et al. 2009; Korns 2011; Uy et al. 2010; Jin et al. 2019; Petersen 2019) consisting of real-valued symbolic functions. The majority of these benchmarks are proposed for GP-based methods and grouped into four categories: polynomial, trigonometric, logarithmic, exponential, and square-root functions, and a combination of univariate and bivariate functions. The suggested functions do not have any physical meaning, and most depend

---

[4] The equation number II.11.17 is missing in the benchmark repository.

either on one or two independent variables. Datasets are generally generated by randomly sampling either 20 or 100 points in narrow ranges. The most commonly known is the so-called Nguyen benchmark, which consists of 12 symbolic functions taken from Keijzer (2003); Hoai et al. (2002); Johnson (2009). Only four equations have the scalars {1,2,1/2} as constants therein. Each benchmark is defined by a ground-truth expression, training, and test datasets. The equations proposed in these benchmarks can not be found in a single repository. Therefore we list them in the Appendix in Tables 7, 8, and 9 and Tables 10 and 11 for completeness and for easy comparison.

Real-world problems are characterized by an unknown model that underlies data. This category comprises two groups: observations and measurements. Data sets in the observations category can be originating from any domain such as health informatics, environmental science, business, commerce, etc. Data could be collected online or offline from reports or studies. A wide range of problems can be assessed from the following repositories: the PMLB (Olson et al. 2017), the OpenML (Vanschoren et al. 2013), and the UCI (Dua and Graff 2017). An exemplary application in this category is wind speed forcasting (Abdellaoui and Mehrkanoon 2021). Measurements represent sets of data points that are collected (and sometimes analysed) in physics experiments. Here the target model is either an underlying theory than can be derived from first principles or not. In the first case, symbolic regression would either infer the correct model structure and parameters or contribute to the theory development of the studied process, whereas in the second case, the symbolic regression output could be the awaited theory.

9 Discussion

SR is a growing area of ML and is gaining more attention as interpretability is increasingly promoted (Rudin 2019) in AI applications. SR is propelled by the fact that ML models are becoming very big in parameters at the expense of making accurate predictions. An exemplary application is the chatGPT-4, a large language model comprising hundreds of billions of parameters and trained on hundreds of terabytes of textual data. Such big models are very complicated networks. ChatGPT-4, for example, is accomplishing increasingly complicated and intelligent tasks to the point that it is showing emergent properties (Wei et al. 2022). However, it is not straightforward to understand when it works and, more importantly, when it does not. In addition, its performance improves with increasing the number of parameters, highlighting that its prediction accuracy depends on the size of the training data set. Therefore, a new paradigm is needed, especially in scientific disciplines, such as physical sciences, where problems are of causal hypothesis-driven nature. SR is by far the most potential candidate to fulfill the interpretability requirements and is expected to play a central role in the future of ML.

Despite the significant advances made in this subfield and the high performance of most deep learning-based SR methods proposed in the literature, still, SR methods fail to recover relatively simple relationships. A case in point is the Nguyen-12 expression, i.e., $f(x, y) = x^4 - x^3 + y^2/2 - y$, where x and y are uniformly sampled in the range [0, 1]. The NGPPS method could not recover this particular expression using the library basis $L = \{+, -, \times, \div, \sin, \cos, \exp, \log, x, y\}$. A variant of this expression, Nguyen-12*, consisting of the same equation but defined over a larger domain, i.e., data points sampled in [0, 10], was successfully covered using the same library, with a recovery rate of 12%. This result is significantly below the perfect performance

on all other Nguyen expressions. A similar observation is made for the Livermore-5 whose expression is $f(x, y) = x^4 - x^3 + x^2 - y$. We ran NGPPS on Nguyen-12 with two libraries, a pure polynomial basis $L_1 = \{+, -, \times, \div, (\cdot)^2, (\cdot)^3, (\cdot)^4, x, y\}$ and a mixed basis $L_2 = L_1 \cup \{\sin, \cos, \exp, \log, \text{sqrt}, \text{expneg}\}$. The algorithm succeeds in recovering Nguyen-12 only using a pure polynomial basis with a recovery rate of 3%. The same observation is made by applying linear SR on Nguyen-12. This highlights how strongly the predicted expression depends on the set of allowable mathematical operations. A practical way to encounter this limitation is to implement basic domain knowledge in SR applications whenever possible. For example, astronomical data collected by detecting the light curves of astronomical objects exhibit periodic behavior. In such cases, periodic functions such as trigonometric functions should be part of the library basis.

Most SR methods are only applied to synthetic data for which the input–output relationship is known. This is justified because the methods must be cross-checked, and their performance must be evaluated using ground-truth expressions. However, the reported results are for synthetic data only. To the best of our knowledge, only one physics application (Lemos et al. 2022) succeeded in extracting New's laws of gravitation by applying SR to astronomical data. The absence of such applications leads us to state that SR is still a relatively nascent area with the potential to make a big impact. Physics in general, and physical sciences in particular, represent a very broad field for SR development purposes and are very rich both in data and expressions, e.g., areas such as astronomy and high-energy physics are very rich in data. In addition, lots of our acquired knowledge in physics can be used for SR methods test purposes because underlying phenomena and equations are well known. All that is needed is greater effort and investment.

10 Conclusion

This work presents an in-depth introduction to the symbolic regression problem and an expansive review of its methodologies and state-of-the-art applications. Also, this work highlights a number of conclusions that can be made about symbolic regression methods, including (1) linear symbolic regression suffer many limitations, all originating from pre-defining the model structure, (2) neural network-based methods lead to numerical issues and the library can not include all mathematical operations, (3) expression tree-based methods are yet the most powerful in terms of model performance on synthetic data, in particular transformer-based ones, (4) model predictions strongly depend on the set of allowable operations in the library basis, and (5) generally, deep learning-based methods are performing better than other ML-based methods.

Symbolic regression represents a powerful tool for learning interpretable models in a data-driven manner. Its application is likely to grow in the future because it balances prediction accuracy and interpretability. Despite the limited SR application to real data, the few existing ones are very promising. A potential path to boost progress in this subfield is to apply symbolic regression to experimental data in physics.

Appendix: Datasets benchmarks equations

Table 7 Ground-truth expressions for Koza ([1994](#)), Nguyen (Uy et al. [2010](#)), Jin (Jin et al. [2019](#)), Keijzer (Keijzer [2003](#)) and R Krawiec and Pawlak ([2013](#)) benchmarks

Dataset	Expression	Variables	Data range
Koza-1[a]	$x^4 + x^3 + x^2 + x$	1	$U[-1, 1, 20]$
Koza-2	$x^5 - 2x^3 + x$	1	$U[-1, 1, 20]$
Koza-3	$x^6 - 2x^4 + x^2$	1	$U[-1, 1, 20]$
Nguyen-1	$x^3 + x^2 + x$	1	$U(-1,1,20)$
Nguyen-2	$x^4 + x^3 + x^2 + x$	1	$U(-1,1,20)$
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	1	$U(-1,1,20)$
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	$U(-1,1,20)$
Nguyen-5	$\sin(x^2)\cos(x) - 1$	1	$U(-1,1,20)$
Nguyen-6	$\sin(x) + \sin(x + x^2)$	1	$U(-1,1,20)$
Nguyen-7	$\log(x + 1) + \log(x^2 + 1)$	1	$U(0,2,20)$
Nguyen-8	$\sqrt{x}$	1	$U(0,4,20)$
Nguyen-9	$\sin(x) + \sin(y^2)$	2	$U(-1,1,100)$
Nguyen-10	$2\sin(x)\cos(y)$	2	$U(-1,1,100)$
Nguyen-11	x^y	2	
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	2	
Jin-1	$2.5x^4 - 1.3x^3 + 0.5y^2 - 1.7y$	2	$U(-3,3,100)$
Jin-2	$8.0x^2 + 8.0y^3 - 15.0$	2	$U(-3,3,100)$
Jin-3	$0.2x^3 + 1.5y^3 - 1.2y - 0.5x$	2	$U(-3,3,100)$
Jin-4	$1.5\exp(x) + 5.0\cos(y)$	2	$U(-3,3,100)$
Jin-5	$6.0\sin(x)\cos(y)$	2	$U(-3,3,100)$
Jin-6	$1.35xy + 5.5\sin((x - 1.0)(y - 1.0))$	2	$U(-3,3,100)$
Keijzer-1	$0.3x\sin(2\pi x)$	1	$E[-1, 1, 0.1]$
Keijzer-2	$0.3x\sin(2\pi x)$	1	$E[-2, 2, 0.1]$
Keijzer-3	$0.3x\sin(2\pi x)$	1	$E[-3, 3, 0.1]$
Keijzer-4	$x^3 e^{-x}\cos(x)\sin(x)(\sin^2(x)\cos(x) - 1)$	1	$E[0, 10, 0.05]$
Keijzer-5	$30xz/(x - 10)y^2$	3	$x, z:\ U[-1,1,1000]$
			$y:\ U[1,2,1000]$
Keijzer-6	$\sum_1^x i$	1	$E[1, 50, 1]$
Keijzer-7	$\log x$	1	$E[1, 100, 1]$
Keijzer-8	$\sqrt{x}$	1	$E[0, 100, 1]$
Keijzer-9	$\operatorname{arcsinh}(x) = \log(x + \sqrt{x^2 + 1})$	1	$E[0, 100, 1]$
Keijzer-10	x^y	2	$U[0, 1, 100]$
Keijzer-11	$xy + \sin((x - 1)(y - 1))$	2	$U[-3, 3, 20]$
Keijzer-12	$x^4 - x^3 + y^2/2 - y$	2	$U[-3, 3, 20]$
Keijzer-13	$6\sin(x)\cos(y)$	2	$U[-3, 3, 20]$
Keijzer-14	$8/(2 + x^2 + y^2)$	2	$U[-3, 3, 20]$
Keijzer-15	$x^3/5 + y^3/2 - y - x$	2	$U[-3, 3, 20]$
R1	$(x + 1)^3/(x^2 - x + 1)$	1	$E[-1,1,20]$
R2	$(x^5 - 3x^3 + 1)/(x^2 + 1)$	1	$E[-1,1,20]$
R3	$(x^6 + x^5)/(x^4 + x^3 + x^2 + x + 1)$	1	$E[-1,1,20]$

[a]Same as Nguyen-2

Table 8 Ground-truth expressions for Korns (2011) and Livermore (Petersen 2019) benchmarks

Dataset	Expression	Variables	Data range
Korns-1	$1.57 + (24.3v)$	1	U[− 50, 50, 10000]
Korns-2	$0.23 + 14.2 \frac{v+y}{3\omega}$	3	U[− 50, 50, 10000]
Korns-3	$-5.41 + 4.9 \frac{v-x+y/w}{3\omega}$	4	U[− 50, 50, 10000]
Korns-4	$-2.3 + 0.13 \sin(z)$	1	U[− 50, 50, 10000]
Korns-5	$3 + 2.13 \ln(\omega)$	1	U[− 50, 50, 10000]
Korns-6	$1.3 + 0.13 \sqrt{x}$	1	U[− 50, 50, 10000]
Korns-7	$213.80940889(1 - e^{-0.54723748542x})$	1	U[− 50, 50, 10000]
Korns-8	$6.87 + 11 \sqrt{7.23\, x\, v\, \omega}$	3	U[− 50, 50, 10000]
Korns-9	$\frac{\sqrt{x}}{\ln(y)} \frac{e^z}{v^2}$	4	U[− 50, 50, 10000]
Korns-10	$0.81 + 24.3 \frac{2y+3z^2}{4v^3+5\omega^4}$	4	U[− 50, 50, 10000]
Korns-11	$6.87 + 11 \cos(7.23x^3)$	1	U[− 50, 50, 10000]
Korns-12	$2 - 2.1 \cos(9.8x) \sin(1.3\omega)$	2	U[− 50, 50, 10000]
Korns-13	$32 - 3 \frac{\tan(x)}{\tan(y)} \frac{\tan(z)}{\tan(v)}$	4	U[− 50, 50, 10000]
Korns-14	$22 - 4.2(\cos(x) - \tan(y)) \frac{\tanh(z)}{\sin(v)}$	4	U[− 50, 50, 10000]
Korns-15	$12 - 6 \frac{\tan(x)}{e^y}(\ln(z) - \tan(v))$	4	U[− 50, 50, 10000]
Livermore-1	$1/3 + x + \sin(x^2)$	1	U[− 10,10,1000]
Livermore-2	$\sin(x^2) \cos(x) - 2$	1	U[− 1,1,20]
Livermore-3	$\sin(x^3) \cos(x^2) - 1$	1	U[− 1,1,20]
Livermore-4	$\log(x + 1) + \log(x^2 + 1) + \log(x)$	1	U[0,2,20]
Livermore-5	$x^4 - x^3 + x^2 - y$	2	U[0,1,20]
Livermore-6	$4x^4 + 3x^3 + 2x^2 + x$	1	U[− 1,1,20]
Livermore-7	$\sinh(x)$	1	U[− 1,1,20]
Livermore-8	$\cosh(x)$	1	U[− 1,1,20]
Livermore-9	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	U[− 1,1,20]
Livermore-10	$6 \sin(x) \cos(y)$	2	U[0,1,20]
Livermore-11	$x^2 y^2 / (x + y)$	2	U[− 1,1,50]
Livermore-12	x^5 / y^3	2	U[− 1,1,50]
Livermore-13	$x^{1/3}$	1	U[0,4,20]
Livermore-14	$x^3 + x^2 + x + \sin(x) + \sin(x^2)$	1	U[− 1,1,20]
Livermore-15	$x^{1/5}$	1	U[0,4,20]
Livermore-16	$x^{2/5}$	1	U[0,4,20]
Livermore-17	$4 \sin(x) \cos(y)$	2	U[0,1,20]
Livermore-18	$\sin(x^2) \cos(x) - 5$	1	U[− 1,1,20]
Livermore-19	$x^5 + x^4 + x^2 + x$	1	U[− 1,1,20]
Livermore-20	$\exp(-x^2)$	1	U[− 1,1,20]
Livermore-21	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	U[− 1,1,20]
Livermore-22	$\exp(-0.5x^2)$	1	U[− 1,1,20]

Table 9 Ground-truth expressions for Vladislavleva et al. (2009) benchmark

Dataset	Expression	Variables	Data range
Vladislavleva-1	$\dfrac{e^{-(x-1)^2}}{1.2+(y-2.5)^2}$	1	U[0.3, 4, 100]
Vladislavleva-2	$e^{-x}x^3(\cos x \sin x)(\cos x \sin^2 x - 1)$	2	E[0.5, 10, 0.1]
Vladislavleva-3	$e^{-x}x^3(\cos x \sin x)(\cos x \sin^2 x - 1)(y-5)$	2	x : E[0.05,10,0.1]
			y : E[0.05,10.05,2]
Vladislavleva-4	$\dfrac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$	5	U[0.05, 6.05, 1024]
Vladislavleva-5	$30(x-1)\dfrac{(z-1)}{y^2(x-10)}$	3	x : U[0.05, 2, 300]
			y : U[1, 2, 300]
			z : U[0.05, 2, 300]
Vladislavleva-6	$6\sin(x)\cos(y)$	2	U[0.1, 5.9, 30]
Vladislavleva-7	$(x-3)(y-3) + 2\sin((x-4)(y-4))$	2	U[0.05, 6.05, 300]
Vladislavleva-8	$\dfrac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	2	U[0.05, 6.05, 50]

Table 10 Feynman physics equation (Udrescu and Tegmark 2019)

Function form	#v	Function form	#v
$f = \exp(-\theta^2/2)/\sqrt{(2\pi)}$	1	$n = n_0\exp(-mgx/(k_bT))$	6
$f = \exp(-(\theta/\sigma)^2/2)/(\sqrt{(2\pi)}\sigma)$	2	$L_{rad} = \hbar\omega^3/(\pi^2c^2(\exp(\hbar\omega/(k_bT))-1))$	5
$f = \exp(-((\theta-\theta_1)/\sigma)^2/2)/(\sqrt{(2\pi)}\sigma)$	3	$v = mu_{drift}qV_e/d$	4
$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$	4	$D = \mu_ek_bT$	3
$F = \frac{Gm1m2}{((x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2)}$	9	$\kappa = 1/(\gamma-1)k_bv/A$	4
$m = \frac{m_0}{\sqrt{1-v^2/c^2}}$	3	$E = nk_bT\ln(\frac{V2}{V1})$	5
$A = x_1y_1+x_2y_2+x_3y_3$	6	$c = \sqrt{(\gamma pr/\rho)}$	3
$F = \mu N_n$	2	$E = mc^2/\sqrt{1-v^2/c^2}$	3
$F = q_1q_2/(4\pi\epsilon r^2)$	4	$x = x_1(\cos(\omega t)+\alpha\cos(\omega t)^2)$	4
$E_f = q_1r/(4\pi\epsilon r^3)$	3	$P = \kappa(T_2-T_1)A/d$	5
$F = q_2E_f$	2	$F_E = Pwr/(4\pi r^2)$	2
$F = q(E_f+Bv\sin(\theta))$	5	$V_e = q/(4\pi\epsilon r)$	3
$K = 1/2m(v^2+u^2+w^2)$	4	$V_e = \frac{1}{4\pi\epsilon}p_d\cos(\theta)/r^2$	4
$U = Gm1m2(\frac{1}{r_2}-\frac{1}{r_1})$	5	$E_f = \frac{3}{4\pi\epsilon}p_dz/r^5\sqrt{x^2+y^2}$	6
$U = mgz$	3	$E_f = \frac{3}{4\pi\epsilon}p_d\cos(\theta)\sin(\theta)/r^3$	4
$U = \frac{1}{2}k_{spring}x^2$	2	$E = \frac{3}{5}q^2/(4\pi\epsilon d)$	3
$x' = (x-ut)/\sqrt{1-u^2/c^2}$	4	$E_{den} = \epsilon E f^2/2$	2
$t' = (t-ux/c^2)/\sqrt{1-u^2/c^2}$	4	$E_f = \sigma_{den}/\epsilon 1/(1+\chi)$	3
$p = m_0v/\sqrt{1-v^2/c^2}$	3	$x = qE_f/(m(\omega_0^2-\omega^2))$	5
$v' = (u+v)/(1+uv/c^2)$	3	$n = n_0(1+p_dE_f\cos(\theta)/(k_bT))$	6
$r = (m_1r_1+m_2r_2)/(m1+m2)$	4	$P_\star = n_{rho}p_d^2E_f/(3k_bT)$	5
$\tau = rF\sin(\theta)$	3	$P_\star = n\alpha/(1-(n\alpha/3))\epsilon E_f$	4
$L = mrv\sin(\theta)$	4	$\theta = 1+n\alpha/(1-(n\alpha/3))$	2
$E = \frac{1}{4}m(\omega^2+\omega_0^2)x^2$	4	$B = 1/(4\pi\epsilon c^2)2I/r$	4
$V_e = q/C$	2	$\rho_c = \rho_{c_0}/\sqrt{1-v^2/c^2}$	3
$\theta_1 = \arcsin(n\sin(\theta 2))$	2	$j = \rho_{c_0}v/\sqrt{1-v^2/c^2}$	3
$f_f = 1/(\frac{1}{d_1}+\frac{n}{d_2})$	3	$E = -\mu_M B\cos(\theta)$	3
$k = \omega/c$	2	$E = -p_dE_f\cos(\theta)$	3
$x = \sqrt{x_1^2+x_2^2-2x_1x_2\cos(\theta_1-\theta_2)}$	4	$V_e = q/(4\pi\epsilon r(1-v/c))$	5
$I_\star = I_{0\star}\sin^2(n\theta/2)/\sin^2(\theta/2)$	3	$k = \sqrt{\omega^2/c^2-\pi^2/d^2}$	3
$\theta = \arcsin(\lambda/nd)$	3	$F_E = \epsilon cE_f^2$	3
$P = q^2a^2/(6\pi\epsilon c^3)$	4	$E_{den} = \epsilon E_f^2$	2
$P = (1/2\epsilon cE_f^2)(8\pi r^2/3)(\omega^4/(\omega^2-\omega_0^2)^2)$	6	$I = qv/(2\pi r)$	3
$\omega = qvB/p$	4	$\mu_M = qvr/2$	3
$\omega = \omega_0/(1-v/c)$	3	$\omega = gqB/(2m)$	4
$\omega = (1+v/c)/\sqrt{1-v^2/c^2}\omega_0$	3	$\mu_M = q\hbar/(2m)$	3
$E = \hbar\omega$	2	$E = g\mu_M BJ_z/\hbar$	5
$I_\star = I_1+I_2+2\sqrt{I_1I_2}\cos(\delta)$	3	$M = n_{rho}\mu_M\tanh(\mu_M B/(k_bT))$	5
$r = 4\pi\epsilon\hbar^2/(mq^2)$	4	$f = \mu_m B/(k_bT)+(\mu_m\alpha)/(\epsilon c^2k_bT)M$	8
$E = \frac{3}{2}p_FV$	2	$E = \mu_M(1+\chi)B$	6

Table 10 (continued)

Function form	#v	Function form	#v
$E = 1/(\gamma - 1)p_F V$	3	$F = YA_x/d$	4
$P_F = nkb_T/V$	4	$\mu_S = Y/(2(1 + \sigma))$	2
		$E = \bar{h}\omega/(\exp(\bar{h}\omega/(k_b T)) - 1)$	4

Table 11 Feynman physics equation (Udrescu and Tegmark 2019)

Function form	# variables
$n = 1/(\exp(\bar{h}\omega/(k_b T)) - 1)$	4
$n = n_0/(\exp(\mu_m B/(k_b T)) + \exp(-\mu_m B/(k_b T)))$	
$\omega = 2\mu_M B/\bar{h}$	3
$p_\gamma = \sin(E_n t/\bar{h})^2$	3
$p_\gamma = (p_d E_f t/\bar{h})\sin((\omega - \omega_0)t/2)^2/((\omega - \omega_0)t/2)^2$	6
$E = \mu_M\sqrt{B_x^2 + B_y^2 + B_z^2}$	3
$L = n\bar{h}$	2
$v = 2E_n d^2 k/\bar{h}$	4
$I = I_0(\exp(qV_e/(k_b T)) - 1)$	5
$E = 2U(1 - \cos(kd))$	3
$m = \bar{h}^2/(2E_n d^2)$	3
$k = 2\pi\alpha/(nd)$	3
$f = \beta(1 + \alpha\cos(\theta))$	3
$E = -mq^4/(2(4\pi\epsilon)^2\bar{h}^2)(1/n^2)$	4
$j = -\rho_{c_0}qA_{vec}/m$	4

References

Abdellaoui IA, Mehrkanoon S (2021) Symbolic regression for scientific discovery: an application to wind speed forecasting. In: 2021 IEEE symposium series on computational intelligence (SSCI), 01–08

Alaa AM, van der Schaar M (2019) Demystifying black-box models with symbolic metamodels. In: Wallach H, Larochelle H, Beygelzimer A, d' Alché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems, vol 32. Curran Associates Inc, New York

Arnaldo I, Krawiec K, O'Reilly U-M (2014) Multiple regression genetic programming. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation. GECCO '14. Association for Computing Machinery, New York, NY, USA, pp 879–886. https://doi.org/10.1145/2576768.2598291

Batra R, Song L, Ramprasad R (2020) Emerging materials intelligence ecosystems propelled by machine learning. Nat Rev Mater 6(8):655–678. https://doi.org/10.1038/s41578-020-00255-y

Beals R, Szmigielski J (2013) Meijer g-functions: a gentle introduction. Not Am Math Soc 60:866–873

Biggio L, Bendinelli T, Neitz A, Lucchi A, Parascandolo G (2021) Neural symbolic regression that scales. CoRR arXiv:2106.06427

Brunton SL, Proctor JL, Kutz JN (2016) Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc Natl Acad Sci 113(15):3932–3937. https://doi.org/10.1073/pnas.1517384113

Cava WGL, Singh TR, Taggart J, Suri S, Moore JH (2018) Stochastic optimization approaches to learning concise representations. CoRR arXiv:1807.00981

Cava WGL, Orzechowski P, Burlacu B, de França FO, Virgolin M, Jin Y, Kommenda M, Moore JH (2021) Contemporary symbolic regression methods and their relative performance. CoRR arXiv:2107.14351

Champion K, Lusch B, Kutz JN, Brunton SL (2019) Data-driven discovery of coordinates and governing equations. Proc Natl Acad Sci 116(45):22445–22451. https://doi.org/10.1073/pnas.1906995116

Chen B, Huang K, Raghupathi S, Chandratreya I, Du Q, Lipson H (2021) Discovering state variables hidden in experimental data. https://doi.org/10.48550/ARXIV.2112.10755

Cranmer MD, Sanchez-Gonzalez A, Battaglia PW, Xu R, Cranmer K, Spergel DN, Ho S (2020) Discovering symbolic models from deep learning with inductive biases. CoRR arXiv:2006.11287

de França FO, Aldeia GSI (2019) Interaction-transformation evolutionary algorithm for symbolic regression. CoRR arXiv:1902.03983

Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml

Dubcakova R (2011) Eureqa: software review. Genet Program Evolvable Mach 12(2):173–178. https://doi.org/10.1007/s10710-010-9124-z

Falkenhainer BC, Michalski RS (1986) Integrating quantitative and qualitative discovery: the abacus system. Mach Learn 1(4):367–401. https://doi.org/10.1023/A:1022866732136

Feynman RP, Leighton RB, Sands ML, Gottlieb MA (2006) The Feynman lectures on physics, vol 2. Pearson/Addison-Wesley, Boston

Feynman RP, Leighton RB, Sands M (2011) The Feynman lectures on physics, vol I: The New Millennium Edition: Mainly mechanics, radiation, and heat. The Feynman lectures on physics. Basic Books, New York

Gerwin D (1974) Information processing, data inferences, and scientific generalization. Syst Res Behav Sci 19:314–325

Hernandez A, Balasubramanian A, Yuan F, Mason S, Mueller T (2019) Fast, accurate, and transferable many-body interatomic potentials by symbolic regression

Heuristic, Laboratory, E.A. https://github.com/heal-research

Hoai NX, McKay RI, Essam D, Chau R (2002) Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. In: Proceedings of the 2002 Congress on evolutionary computation. CEC'02 (Cat. No.02TH8600), vol. 2, pp 1326–13312. https://doi.org/10.1109/CEC.2002.1004435

Jin Y, Fu W, Kang J, Guo J, Guo J (2019) Bayesian symbolic regression. https://doi.org/10.48550/ARXIV.1910.08892

Johnson CG (2009) Genetic programming crossover: does it cross over? In: Vanneschi L, Gustafson S, Moraglio A, De Falco I, Ebner M (eds) Genetic programming. Springer, Berlin, pp 97–108

Kamienny P-A, d'Ascoli S, Lample G, Charton F (2022) End-to-end symbolic regression with transformers. arXiv:2204.10532

Keijzer M (2003) Improving symbolic regression with interval arithmetic and linear scaling. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E (eds) Genetic programming. Springer, Berlin, pp 70–82

Kepler J (1953) Epitome astronomiae copernicanae. In: Noscemus Wiki. http://wiki.uibk.ac.at/noscemus/Epitome_astronomiae_Copernicanae

Korns MF (2011). In: Riolo R, Vladislavleva E, Moore JH (eds) Accuracy in symbolic regression. Springer, New York, pp 129–151. https://doi.org/10.1007/978-1-4614-1770-5_8

Koza JR (1989) Hierarchical genetic algorithms operating on populations of computer programs. In: Proceedings of the 11th International joint conference on artificial intelligence, vol 1, IJCAI'89. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 768–774

Koza JR (1990) Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford, CA, USA

Koza JR (1994) Genetic programming as a means for programming computers by natural selection. Proc Natl Acad Sci 4(2):87–112. https://doi.org/10.1007/BF00175355

Krawiec K, Pawlak T (2013) Approximating geometric crossover by semantic backpropagation. In: Proceedings of the 15th annual conference on genetic and evolutionary computation. GECCO '13. Association for Computing Machinery, New York, NY, USA, pp 941–948. https://doi.org/10.1145/2463372.2463483

La Cava W, Danai K, Spector L (2016) Inference of compact nonlinear dynamic models by epigenetic local search. Eng Appl Artif Intell 55:292–306. https://doi.org/10.1016/j.engappai.2016.07.004

Langley PW (1979) Bacon: a production system that discovers empirical laws. https://www.ijcai.org/Proceedings/77-1/Papers/057.pdf

Langley P (1981) Data-driven discovery of physical laws. Cogn Sci 5(1):31–54. https://doi.org/10.1016/S0364-0213(81)80025-0

Langley P, Simon HA, Bradshaw GL, Zytkow JM (1987) Scientific discovery: computational explorations of the creative process. MIT Press, Cambridge

Lemos P, Jeffrey N, Cranmer M, Ho S, Battaglia P (2022) Rediscovering orbital mechanics with machine learning. Mach Learn Sci Technol 4:045002. https://doi.org/10.48550/ARXIV.2202.02306

Makke N, Chawla S (2022) A living review of symbolic regression. https://github.com/nmakke/SR-LivingReview

Martinez-Gil J, Chaves-Gonzalez JM (2020) A novel method based on symbolic regression for interpretable semantic similarity measurement. Expert Syst Appl 160:113663. https://doi.org/10.1016/j.eswa.2020.113663

Martius G, Lampert CH (2016) Extrapolation and learning equations. CoRR arXiv:1610.02995

McConaghy T (2011). In: Riolo R, Vladislavleva E, Moore JH (eds) FFX: fast, scalable, deterministic symbolic regression technology. Springer, New York, pp 235–260. https://doi.org/10.1007/978-1-4614-1770-5_13

Meijer C (1946) On the G-function. North-Holland, Amsterdam

Mozaffari-Kermani M, Sur-Kolay S, Raghunathan A, Jha NK (2015) Systematic poisoning attacks on and defenses for machine learning in healthcare. IEEE J Biomed Health Inform 19(6):1893–1905. https://doi.org/10.1109/JBHI.2014.2344095

Mundhenk TN, Landajuela M, Glatt R, Santiago CP, Faissol DM, Petersen BK (2021) Symbolic regression via neural-guided genetic programming population seeding. CoRR arXiv:2111.00053

Newton I, Motte A, Machin J (1729) The mathematical principles of natural philosophy, vol 1. B. Motte, London

Olson RS, La Cava W, Orzechowski P, Urbanowicz RJ, Moore JH (2017) Pmlb: a large benchmark suite for machine learning evaluation and comparison. BioData Min 10(36):1–13. https://doi.org/10.1186/s13040-017-0154-4

O'Reilly U-M (1994) Genetic programming II: automatic discovery of reusable programs. Artif Life 1(4):439–441. https://doi.org/10.1162/artl.1994.1.4.439

Petersen BK (2019) Deep symbolic regression: recovering mathematical expressions from data via policy gradients. CoRR arXiv:1912.04871

Robinson R (1958) Jan Łukasiewicz: Aristotle's syllogistic from the standpoint of modern formal logic. second edition enlarged. pp. xvi 222. Oxford: Clarendon Press, 1957. cloth, 305. net. The Class Rev 8(3–4):282–282. https://doi.org/10.1017/S0009840X00168337

Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. MIT Press, Cambridge, pp 318–362

Sahoo SS, Lampert CH, Martius G (2018) Learning equations for extrapolation and control. CoRR arXiv:1806.07259

Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324(5923):81–85. https://doi.org/10.1126/science.1165893

Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. A Bradford book. The MIT Press, Cambridge

Tegmark M (2019) The Feynman symbolic regression database. https://space.mit.edu/home/tegmark/aifeynman.html

Udrescu S-M, Tegmark M (2019) AI Feynman: a physics-inspired method for symbolic regression. Sci Adv https://doi.org/10.48550/ARXIV.1905.11481

Uy NQ, Hoai NX, O'Neill M, McKay RI, López EG (2010) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet Program Evolvable Mach 12:91–119

Valipour M, You B, Panju M, Ghodsi A (2021) Symbolicgpt: a generative transformer model for symbolic regression. CoRR arXiv:2106.14131

Vanschoren J, van Rijn JN, Bischl B, Torgo L (2013) Openml: networked science in machine learning. SIG-KDD Explor 15(2):49–60. https://doi.org/10.1145/2641190.2641198

Vapnik V (1991) Principles of risk minimization for learning theory. In: Moody J, Hanson S, Lippmann RP (eds) Advances in neural information processing systems, vol 4. Morgan-Kaufmann, Cambridge

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. CoRR arXiv:1706.03762

Virgolin M, Pissis SP (2022) Symbolic regression is NP-hard. arXiv: 2207.01018

Virgolin M, Alderliesten T, Witteveen C, Bosman PAN (2019) A model-based genetic programming approach for symbolic regression of small expressions. CoRR arXiv:1904.02050

Virgolin M, Wang Z, Alderliesten T, Bosman PAN (2020) Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction. J Med Imaging 7(4):046501. https://doi.org/10.1117/1.JMI.7.4.046501

Vladislavleva E, Smits G, den Hertog D (2009) Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans Evol Comput 13:333–349

Wang Y, Wagner N, Rondinelli JM (2019) Symbolic regression in materials science. MRS Commun 9(3):793–805. https://doi.org/10.1557/mrc.2019.85

Wei J, Tay Y, Bommasani R, Raffel C, Zoph B, Borgeaud S, Yogatama D, Bosma M, Zhou D, Metzler D, Chi EH, Hashimoto T, Vinyals O, Liang P, Dean J, Fedus W (2022) Emergent abilities of large language models

Weng B, Song Z, Zhu R, Yan Q, Sun Q, Grice CG, Yan Y, Yin W-J (2020) Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts. Nat Commun 11:3513

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8(3–4):229–256. https://doi.org/10.1007/BF00992696