TRANSFORMER-BASED MODEL FOR SYMBOLIC REGRESSION VIA JOINT SUPERVISED LEARNING

Wenqiang Li[1,2,4] **Weijun Li**[1,3,4,*] **Linjun Sun**[1,3,4] **Min Wu**[1,3,4] **Lina Yu**[1,3,4]
Jingyi Liu[1,3,4] **Yanjie Li**[1,3,4] **Songsong Tian**[1,2,4]

[1]Institute of Semiconductors, Chinese Academy of Sciences, Beijing, China
[2]School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, China
[3]School of Integrated Circuits, University of Chinese Academy of Sciences, Beijing, China
[4]Beijing Key Laboratory of Semiconductor Neural Network Intelligent Sensing and Computing Technology, Beijing, China

ABSTRACT

Symbolic regression (SR) is an important technique for discovering hidden mathematical expressions from observed data. Transformer-based approaches have been widely used for machine translation due to their high performance, and are recently highly expected to be used for SR. They input the data points, then output the expression skeleton, and finally optimize the coefficients. However, recent transformer-based methods for SR focus more attention on large scale training data and ignore the ill-posed problem: the lack of sufficient supervision, i.e., expressions that may be completely different have the same supervision because of their same skeleton, which makes it challenging to deal with data that may be from the same expression skeleton but with different coefficients. Therefore, we present a transformer-based model for SR with the ability to alleviate this problem. Specifically, we leverage a feature extractor based on *pure residual MLP* networks to obtain more information about data points. Furthermore, the core idea is that we propose a joint learning mechanism combining supervised contrastive learning, which makes features of data points from expressions with the same skeleton more similar so as to effectively alleviates the ill-posed problem. The benchmark results show that the proposed method is up to 25% higher with respect to the recovery rate of skeletons than typical transformer-based methods. Moreover, our method outperforms state-of-the-art SR methods based on reinforcement learning and genetic programming in terms of the coefficient of determination (R^2).

1 INTRODUCTION

Exploring mathematical expressions that can be fitted to real-world observed data is the core of expressing scientific discoveries. The correct expression would not only provide us with useful scientific insights simply by inspection but would also allow us to forecast how the process will change in the future. The task of finding such an interpretable mathematical expression from observed data is called *symbolic regression*. More specifically, given a dataset (X, y), where each feature $X_i \in \mathbb{R}^n$ and target $y_i \in \mathbb{R}$, the goal of symbolic regression is to identify a function f (i.e., $y \approx f(X) : \mathbb{R}^n \to \mathbb{R}$) that best fits the dataset.

Symbolic regression is NP-hard because the search space of an expression grows exponentially with the length of the expression, and the presence of numeric constants further exacerbates its difficulty (Lu et al., 2016). Considering this issue, genetic programming (GP) as the most common approach is leveraged to tackle the symbolic regression problems (Forrest, 1993; Koza, 1994; Schmidt & Lipson, 2009; Staelens et al., 2013; Arnaldo et al., 2015; Błądek & Krawiec, 2019). GP-based methods iteratively "evolves" each generation of mathematical expressions through selection, crossover, and mutation. Although this approach can be effective, the expression it yields

[*]Corresponding author.

is complex, and it is also known to be computationally expensive and to exhibit high sensitivity to hyperparameters. A more recent line of research has made use of the neural network to tackle the aforementioned shortcomings. Martius & Lampert (2016) propose a simple fully-connected neural network called "EQL", where elementary functions ($\sin$, $\cos$, $+$, ...) are used as activation functions. The limitation of EQL is the existence of vanishing gradient and exploding gradient, and the depth of the network limits the complexity of the predicted equation. More recently, deep symbolic optimization (DSO) (Petersen et al., 2021) trains the RNN using the reinforcement learning algorithm based on a risk-seeking policy gradient to generate expressions. They take the output from RNN as an initial population for a genetic algorithm to find the target expression. Albeit the above two approaches show promising results, they still handle symbolic regression as an instance-based problem, training a model from scratch on each new input dataset for a regression task.

Inspired by the successes of large scale pre-training, recent efforts in symbolic regression have focused on using the transformer-based model and training with a large amount of data (Valipour et al., 2021; Biggio et al., 2021). They all approach the symbolic regression problem as a machine translation problem, mapping the input data to latent representations via encoders, and then outputting the skeleton of expressions without constants by decoders. These transformer-based methods (Valipour et al., 2021; Biggio et al., 2021) for symbolic regression exists two main drawbacks: (i) A natural question is what architecture of the encoder is optimally suited for symbolic regression. It is clear that the decoder's ability to sample expressions efficiently is severely constrained by the encoder's ability to extract the features of the input data. The idea is that the encoder should not just encode the points, but also represent the expression on a high level such that the decoder only prints the representation as a sequence of symbols. (ii) They use the single character of the expression's string (Valipour et al., 2021) and the pre-order traversal of the expression tree (Biggio et al., 2021) as supervision information, respectively, which is an ill-posed problem that does not provide sufficient supervision: different instances of the same skeleton can have very different shapes, and instances of very different skeletons can be very close. To alleviate these issues, we proposed a transformer-based method for symbolic regression using a new feature extractor and a joint supervised learning mechanism.

In summary, we introduce the main contributions in this study as follows:

- We leverage a *pure residual MLP* feature extractor for extracting the local and global features of observed data targeting symbolic regression tasks, which aids the expression generator in producing more correct expression skeletons.
- We propose a joint learning mechanism combining supervised contrastive learning that combines the supervision of the whole expression skeleton with the supervision of the pre-order traversal of its expression tree, which alleviates the ill-posed problem effectively.
- Empirically, the proposed method is up to 25% better than recent transformer-based methods with respect to the recovery rate of expression skeleton. Moreover, our method outperforms several strong baseline methods in terms of R^2.

2 Related Work

Genetic programming (GP) for symbolic regression. Traditionally, the approaches to symbolic regression are based on genetic algorithms (Forrest, 1993). Later, the symbolic regression task is seen as an optimization problem for the search space (Koza, 1994). By far the most popular commercial software Eureqa (Dubčáková, 2011) is the most successful application based on GP methods. A limitation of the genetic algorithms-based methods is that they need to train for each equation from scratch, which is slow, computationally expensive and highly randomized. The models tend to generate more complex equations and they are sensitive to the choice of hyperparameters (Petersen et al., 2021).

Neural network for symbolic regression. Symbolic regression based on neural network approaches can be broadly classified into three categories. First, the methods based on equation learner (EQL) (Martius & Lampert, 2016; Sahoo et al., 2018; Werner et al., 2021) are trained by replacing the activation function of the neural network with arithmetic operators, which inherits the ability of neural networks to deal with high-dimensional data and scales well with the number of input-output pairs (Biggio et al., 2021). Nevertheless, the existence of exponential and logarithmic activation

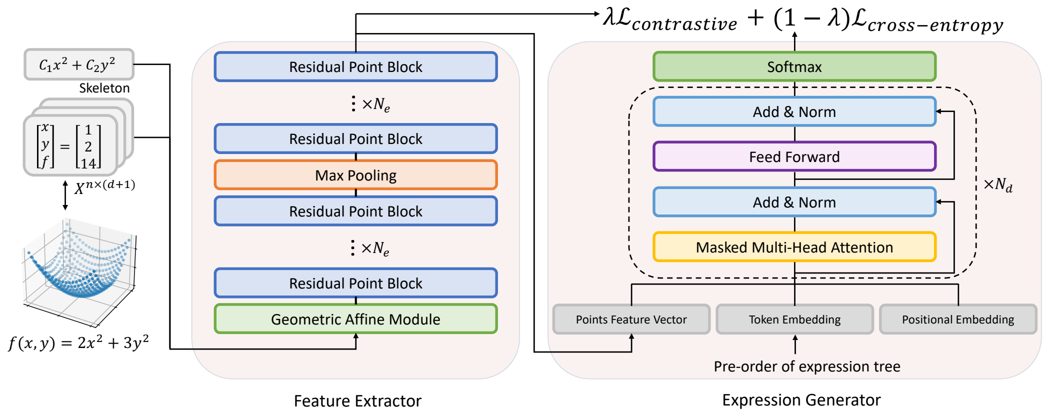


Figure 1: Schematic diagram of training. The data inputs and expression skeletons' labels are passed through the feature extractor. Then, given the feature vectors, token embedding, and positional embedding, the expression generator produces expression skeletons in parallel. Finally, the model jointly computes the supervised contrastive learning loss and cross-entropy loss.

functions leads to gradient instability. Also, the complexity of predicted expression depends on the depth of the EQL network.

Reinforcement learning for symbolic regression. The second approach is the autoregressive model based on reinforcement learning. Petersen et al. (2021) uses reinforcement learning based on a risk-seeking policy gradient to train a RNN to generate a probability distribution over the space of mathematical expressions. For such symbolic regression tasks, they proposed a new objective function based on risk-seeking policy gradients, that focuses on learning only on *maximizing best-case performance* rather than the average performance of a policy. Genetic programming and neural-guided search are mechanistically dissimilar, yet both have proven to be effective solutions to symbolic regression. Mundhenk et al. (2021) proposed a more novel approach, combining the two approaches to leverage each of their strengths. They take the output from the RNN as an initial population for a genetic algorithm. The method represents a significant step forward in the application of deep learning to symbolic regression (Biggio et al., 2021). The promising results make it the currently recognized state-of-the-art approach to symbolic regression tasks. Nevertheless, the limitations of this method are obvious, namely, the network has to be retrained from scratch for each new equation and the RNN is never directly conditioned on the data it is trained to model (Biggio et al., 2021).

Large scale transformer-based models for symbolic regression. The third approach is to train a large scale transformer-based model by using a large amount of data. More recently, SymbolicGPT (Valipour et al., 2021) trained a GPT (Radford et al., 2019) model to construct a mapping of pairs of points and symbolic output. They first input the data points into T-net (Qi et al., 2017) to get a potential representation of the data points and then input it to the GPT (Radford et al., 2019) for generating expression strings. They generate the expression at the character level and finally concatenate it into an expression. In general, they explore an alternative approach to symbolic regression by considering it as a task in language modeling. Symbolic mathematics behaves as a language in its own right, with well-formed mathematical expressions treated as valid "sentences" in this language (Valipour et al., 2021). Furthermore, NeSymReS (Biggio et al., 2021) proposed a similar method, where they use the encoder from the Set transformer (Lee et al., 2019) and the decoder from the original transformer architecture (Vaswani et al., 2017). Their greatest contribution is to show that their approach is able to improve performance as the size of the dataset increases.

3 Methods

The proposed joint learning mechanism is shown in Figure 1. First, we leverage a permutation-invariant feature extractor based on residual MLP networks to obtain feature vectors of data points. Then, given the feature vectors, the expression generator autoregressively generates individual mathematical symbols until we obtain the entire skeleton of the expression. In the forward propagation, we compute the contrastive loss with respect to the expression skeleton categories and the cross-entropy loss with respect to the mathematical symbol categories separately. The parameters of the network are jointly updated in backpropagation. Training can efficiently process each sequence in a single forward pass of the network thanks to the masked attention and teacher forcing (Vaswani et al., 2017). During inference, multiple predictions are sampled from the model using the beam search strategy and we select the prediction with the lowest error.

3.1 Extracting effective feature of data points

As mentioned in section 1, the feature extracted from data points affects generating expressions through the decoder. Valipour et al. (2021) obtain the latent representation of data using the T-net (Qi et al., 2017). Albeit efficient, local feature loss caused by non-locality and non-hierarchy degrades the representational quality of details for point cloud (Ma et al., 2022). As a similar work, NeSymReS (Biggio et al., 2021) use the Set Transformer (Lee et al., 2019) based on self-attention to extract data points features. However, it focuses too much on local feature extraction and lacks global information, which is not suitable for expression data points. We explore and visualize the similarity of the features extracted from data points by these methods and compare the expression skeletons generated by the decoder, which confirms the above problems. The result is shown in section 4. The performance of the feature extractor has to be considered, therefore, we opt for the framework of PointMLP (Ma et al., 2022) based on *pure residual MLP* as our feature extractor in order to obtain the local and overall information of the data points.

Given a set of data points $\mathcal{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n} \in \mathbb{R}^{n\times(d+1)}$, where n indicates the number of points and d denotes the dimension of the variable. PointMLP learns hierarchical features of data points by stacking multiple learning stages. In each stage, N_s points are re-sampled by the farthest point sampling (FPS) algorithm, where s indexes the stage and K neighbors are employed for each sampled point (Ma et al., 2022). Conceptually, the kernel operation of PointMLP can be formulated as:

$$O_i = \mathrm{POS}\left(\mathrm{MaxPool}\left(\mathrm{PRE}\left(f_{i,j}\right), |j = 1, \cdots, K\right)\right)$$

where $f_{i,j}$ is the j-th neighbor point feature of i-th sampled point. $\mathrm{POS}(\cdot)$ and $\mathrm{PRE}(\cdot)$ are residual point MLP blocks: the shared $\mathrm{PRE}(\cdot)$ is designed to learn shared weights from a local region while the $\mathrm{POS}(\cdot)$ is leveraged to extract deep aggregated features. $\mathrm{POS}(\cdot)$ and $\mathrm{PRE}(\cdot)$ consist of several residual MLP blocks: $\mathrm{MLP}(x) + x$. We use the max pooling layer to aggregate global features. After MLP blocks, we add a dropout layer (Srivastava et al., 2014).

Following (Ma et al., 2022), we also leverage a lightweight geometric affine module to tackle the problem that is less robust and caused by the sparse and irregular geometric structures in local regions. Let $\{f_{i,j}\}_{j=1,\cdots,k} \in \mathbb{R}^{k\times d}$ be the grouped local neighbors of $f_i \in \mathbb{R}^d$ containing k points, and each neighbor point $f_{i,j}$ is a d-dimensional vector. We transform the local neighbor points by the following formulation:

$$\{f_{i,j}\} = \alpha \odot \frac{\{f_{i,j}\} - f_i}{\sigma + \epsilon} + \beta, \quad \sigma = \sqrt{\frac{1}{k\times n\times d}\sum_{i=1}^{n}\sum_{j=1}^{k}\left(f_{i,j} - f_i\right)^2}$$

where $\alpha \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ are learnable parameters, $\odot$ indicates Hadamard production, and $\epsilon = 1e^{-5}$ is a small number for numerical stability. Note that σ is a scalar that describes the feature deviation across all local groups and channels. By doing so, we transform the local points to a normal distribution while maintaining original geometric properties.

3.2 Training with joint supervision information

Recent transformer-based face an ill-posed problem because use insufficient supervision information: they use the single character of the expression's string (Valipour et al., 2021) and the pre-order traversal of the expression (Biggio et al., 2021) as supervision information, respectively, and train the model by minimizing the cross-entropy loss. In this work, we propose a joint objective function that combines cross-entropy (CE) loss and supervised contrastive learning (CL) loss. On the basis of using symbol labels as supervision, we treat the skeleton of expressions as category labels to enrich the supervisory information. The CL loss with respect to the expression skeleton categories and the

CE loss with respect to the mathematical symbol categories are separately calculated in the forward propagation. In backpropagation, the network's parameters are concurrently updated. As the auxiliary loss, the CL loss is meant to capture the similarities of feature vectors between expressions with the same skeletons and contrast them with others. The promising results are described in section 4. The overall loss is a weighted average of CE loss and proposed CL loss, as given in equation (1). The canonical definition of the CE loss that we use is given in equation (2). The novel CL loss is given in equation (3). The overall loss is then given in the following:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_{CE} + \lambda\mathcal{L}_{CL} \tag{1}$$

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \cdot \ln \hat{y}_{i,c} \tag{2}$$

$$\mathcal{L}_{CL} = -\sum_{i=1}^{N}\frac{1}{N_{\ell_i}+\epsilon}\sum_{j=1}^{N}\mathbf{1}_{i\neq j}\mathbf{1}_{\ell_i=\ell_j}\ln\frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{N}\mathbf{1}_{i\neq k}\mathbf{1}_{\ell_i\neq\ell_k}\exp\left(s_{i,k}/\tau\right)} \tag{3}$$

Here, λ is a scalar weighting hyperparameter that we tune for the training stage; N represents the mini-batch size; C denotes the size of the pre-specified tokens library; $y_{i,c}$ denotes the symbol label and $\hat{y}_{i,c}$ denotes the expression generator output for the probability of the ith token belonging to the class c; N_{l_i} is the total number of examples in the batch that have the same skeleton label as ℓ_i; ϵ is a very small scalar preventing devision by zero; $\mathbf{1}_{i\neq j}$, $\mathbf{1}_{\ell_i=\ell_j}$ and $\mathbf{1}_{\ell_i\neq\ell_k}$ are similar indicator functions; $s_{i,j} = \boldsymbol{v}_i \cdot \boldsymbol{v}_j / \|\boldsymbol{v}_i\| \|\boldsymbol{v}_j\|$ denotes the cosine similarity between the sample i and the sample j, where $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ represent the high-level feature vectors of the sample i and the sample j respectively from the feature extractor; $\tau > 0$ is an adjustable scalar temperature parameter that controls the separation of classes.

3.3 Generating expressions with a transformer-based model

We leverage a framework of the GPT language model (Radford et al., 2019) as the expression generator, which is an autoregressive language model based on the decoder of transformer (Vaswani et al., 2017). During inference, we generate an expression τ one token at a time along the pre-order traversal. For example, the expression $f(x_1, x_2) = \sin(x_1) + \log(x_2)$ is encoded as $[+, \sin, x_1, \log, x_2]$. We denote the i^{th} token of the traversal as τ_i and each token has a value within a given library $\mathcal{L}$ of possible tokens, e.g., $\{+, -, \times, \div, \sin, \cos, \log, x_1, x_2\}$. Specifically, the i^{th} output of the generator with parameters θ and feature vector $\boldsymbol{v}$ extracted from data $\mathcal{D}$ pass through a softmax layer to produce vector $\psi^{(i)}$, which defines the probability distribution for selecting the i^{th} token τ_i. The likelihood of the entire generated expression is simply the product of the likelihoods of its tokens: $p(\tau|\theta, \boldsymbol{v}) = \prod_{i=1}^{|\tau|} p(\tau_i|\tau_{1:(i-1)}; \theta, \boldsymbol{v}) = \prod_{i=1}^{|\tau|} \psi^{(i)}_{\mathcal{L}(\tau_i)}$. Note that the output sequence from the generator does not contain any numerical constants. Instead, we use a special placeholder $\langle C \rangle$ denoting the presence of a constant that will be optimized at a later stage.

3.4 Learning Constants using BFGS by restarting multiple times

At inference time, Mundhenk et al. (2021); Valipour et al. (2021); Biggio et al. (2021) all use BFGS optimization algorithm (Fletcher, 1984) on the mean squared error to fit the constants. BFGS is a quasi-newton method for solving unconstrained nonlinear convex optimization problems. However, for the symbolic regression task, the loss function minimized by BFGS may be highly non-convex, which is likely to be falling into several local minima: even when the skeleton is perfectly predicted, the correct constants are not guaranteed to be found. To ameliorate this issue, we use a simple and hardly any space cost method: restart the BFGS algorithm by initializing the different starting points multiple times, and the global optimal point is achieved as much as possible. The result is shown in section 4.

4 Experiments and Results

4.1 Generating datasets

We generate mathematical expressions following the framework as described by (Lample & Charton, 2019). The framework starts by generating a random unary-binary tree and filling the nodes with operators and the leaves with independent variables or constants. The unnormalized probabilities of each operation and operator are given in Appendix A. We generate the training set containing approximately $100,000$ unique expression skeletons. For each expression, we re-sample its constant values for 10, 20, 30, 40, and 50 times. Once an expression tree has been generated, we can represent the expression using a pre-order traversal of the tree. We opt for scalar functions of up to two independent variables (i.e., $d = 2$ and $y = f(x_1, x_2)$) and three numerical constants each. Specifically, each generated expression's constant values are replaced with the constant placeholder $\langle C \rangle$. Then, the additive and multiplicative constant values are sampled uniformly from the interval $[-2, 2]$ to fill in the placeholder. After this, the entire equation will be simplified using the rules built in the symbolic manipulation library SymPy (Meurer et al., 2017). Finally, we sample uniformly at random 100 input points $X = \{x_i\}_{i=1}^{n}$ from the interval $[-10, 10]$ and evaluate the expressions on the X to obtain the corresponding outputs Y. X will be re-sampled if produce non-finite values (NaN or $\pm\infty$) and we discard the expression that cannot be sampled completely in 30 seconds.

Creating the SSDNC benchmark. We generate a more challenging test set to discover the performance of transformer-based methods, which includes approximately 100 unique expression skeletons and 10 re-sampled numerical constants for each skeleton. We sample random support of 100 points from the uniform distribution $\mathcal{U}(-10, 10)$, for each independent variable. We call it SSDNC, for the same skeletons with different numerical coefficients.

4.2 Training and Inference

We train the feature extractor and expression generator jointly to minimize the objective loss combined with cross-entropy loss and supervised contrastive loss. More specifically, we train the model using the Adam optimizer (Kingma & Ba, 2014) on 4 NVIDIA V100 GPUs. More detailed hyperparameters are reported in Appendix 5, which were found empirically and not fine-tuned for maximum performance. *Note that we use the same dataset for training to facilitate quantitative benchmarking* when evaluating the feature extraction and other capabilities with SymbolicGPT (Valipour et al., 2021) and NeSymReS (Biggio et al., 2021). At inference, most of the expressions of benchmark test sets are not seen during training, and we resampled all data points to avoid the possible overfitting problem.

4.3 Metrics

We have selected the coefficient of determination (R^2) to assess the quality of our method. The R^2 (Glantz & Slinker, 2001) is defined as follows:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{k}(y_i - \bar{y})^2}$$

where y_i and $\hat{y}_i$ are the ground-truth and predicted values for point i, respectively. $\bar{y}$ is the average of y_i over all the points. k is the number of test points. The advantage of using R^2 is its nice interpretation. if $R^2 > 0$, then means the prediction is better than predicting just the average value, and if $R^2 = 1$, then we get a perfect prediction. However, due to the presence of the max operator, R^2 is sensitive to outliers, and hence to the number of points considered at test time (more points entail a higher risk of an outlier). To circumvent this, we discard the 5% worst predictions for all methods used, following (Biggio et al., 2021).

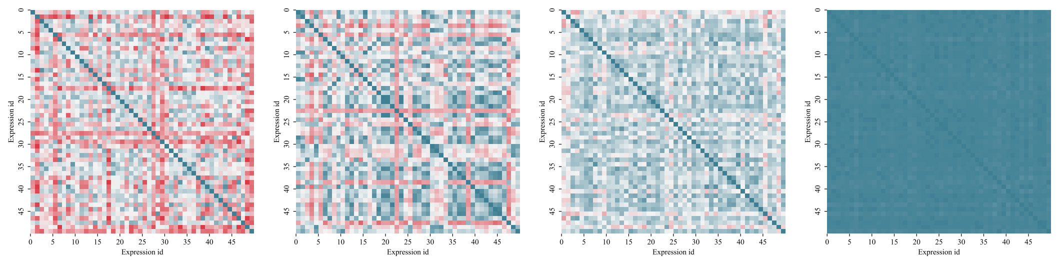
4.4 Baselines

We compare the performance of our method with four strong symbolic regression baselines:

- **SymbolicGPT (Valipour et al., 2021)** A recent novel transformer-based language model for symbolic regression. We use the open-source implementation provided by the authors.[1]
- **Neuaral Symbolic Regression that Scales (NeSymReS) (Biggio et al., 2021)** Recently proposed transformer-based symbolic regression model on the large training data. We use the open-source implementation provided by the authors. [2]

[1]`https://github.com/mojivalipour/symbolicgpt`

[2]`https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales`



(a) SymbolicGPT (b) NeSymReS (c) Ours w/o CL (d) Ours w/ CL

Figure 2: For the expression skeleton $c_1 \sin(x_1) + c_2 \cos(x_2) + c_3$, four heat maps of cosine similarity between the fifty different feature vectors from different methods, where the redder color means the cosine similarity is closer to 0, and the greener color means the cosine similarity is closer to 1.

- **Deep Symbolic Optimization (DSO).** A symbolic regression method based on RNN and reinforcement learning search strategy (Mundhenk et al., 2021). We use the open-source implementation provided by the authors.[3]
- **Genetic Programming.** Standard GP-based symbolic regression (Koza, 1994) based on the open-source Python library `gplearn`.[4]

All details for baselines are reported in Appendix D.

4.5 Feature Extraction Performance

For transformer-based methods in symbolic regression, we empirically demonstrate that the performance of the feature extractor plays a critical role in the overall training and evaluation stages. We evaluate the feature extraction performance of recent transformer-based methods, i.e., SymbolicGPT (Valipour et al., 2021) and NeSymReS (Biggio et al., 2021), and our method with/without CL on SSDNC benchmark. After training, we input the data points corresponding to the specific expression skeleton with different constants into the feature extractor, then we compute the cosine similarity between the different feature vectors. As shown in Figure 2, SymbolicGPT (Valipour et al., 2021) and NeSymReS (Biggio et al., 2021) all produce dissimilar feature vectors for the data points of different expressions belonging to the same skeleton. The reason is that the feature extractor they used focuses more on local features and loses key information, which is not appropriate for the data points of symbolic regression. By manual inspection, we find that this problem can adversely affect the expression generator to produce an expression skeleton. The results in section 4.6 illustrate the high correlation between the feature extractor and expression generating. Benefiting from using the architecture of *pure residual MLP*, our feature extractor is able to obtain more similar feature vectors when facing the same skeleton, even without CL. After training jointly, we make the feature vectors of data points from the same skeleton more similar, which effectively alleviates the ill-posed problem and improves the performance of generating expressions with the same skeletons.

4.6 Evaluation on benchmarks

Recovery rate of skeletons. In inference, since transformer-based methods first generate expression skeletons based on features of data points, we first compare the performance of these methods in recovering expression skeletons. we evaluate the recovery rate of expression skeletons on the SSDNC benchmark. As shown in Table 1, our method with the more effective feature extractor and the joint leaning mechanism can better guide the generator in generating expressions, thus outperforming other transformer-based methods. Generating the right expression skeleton is crucial to the final

Table 1: Revovery rate of expression skeletons on the SSDNC benchmark.

Method	Recovery rate (%)
SymbolicGPT	50.3 ± 1.7
NeSymReS	63.4 ± 1.1
Ours w/o CL	69.7 ± 0.9
Ours w/ CL	$\mathbf{75.2} \pm \mathbf{1.3}$

[3]`https://github.com/brendenpetersen/deep-symbolicregression`
[4]`https://gplearn.readthedocs.io/en/stable/`

Table 2: Results comparing our method with CL with state-of-the-art methods on several benchmarks. Our method, SymbolicGPT, and NeSymReS all use the beam search strategy with the beam size equaling 128. We report the average value of R^2 for each benchmark.

	Ours	SymbolicGPT	NeSymReS	DSO	GP
Benchmark	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$
Nguyen	**0.99999**	0.64394	0.97538	0.99489	0.89019
Constant	**0.99998**	0.69433	0.84935	0.99927	0.90842
Keijzer	**0.98320**	0.59457	0.97500	0.96928	0.90082
R	**0.99999**	0.71093	0.99993	0.97298	0.83198
AI-Feynman	**0.99999**	0.64682	**0.99999**	**0.99999**	0.92242
SSDNC	**0.94782**	0.74585	0.85792	0.93198	0.88913
Overall avg.	**0.98850**	0.67274	0.94292	0.97806	0.89049

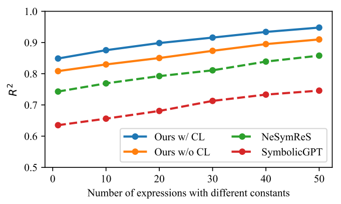


Figure 3: Training on different datasets that contain various numbers of expressions with different constants. Inference on SSDNC benchmark.

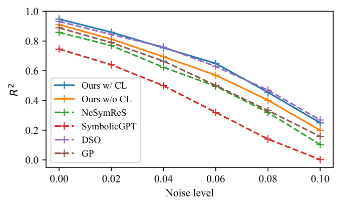


Figure 4: R^2 vs gaussian noisy data. Error bar represent standard error. Inference on SSDNC benchmark.

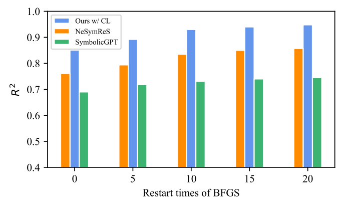


Figure 5: R^2 for different restart times of BFGS in the constant optimization stage. Inference on SSDNC benchmark.

result because optimizing constants is relatively simple. This is reflected in the comparison for R^2 in the following results.

Statistics of fitting accuracy We evaluate our method and current state-of-the-art approaches on the widely used public benchmarks, i.e., the Nguyen benchmark (Uy et al., 2011), Constant, Keijzer (Keijzer, 2003), R rationals (Krawiec & Pawlak, 2013), AI-Feynman database (Udrescu & Tegmark, 2020) and our SSDNC test set. Nguyen was the main benchmark used in Petersen et al. (2021). There are terms that appear in three ground truth equations that are not included in the set of equations that our model can fit, specifically x^6 and x^y, but we can find approximate expressions. AI-Feynman database is extracted from the popular *Feynman Lectures on Physics* series and contains expressions with up to nine variables. In our study, we consider all the expressions with up to two variables. The complete benchmark functions are given in Appendix 4. From the results in Table 2, our method outperforms all baseline methods in terms of average R^2 on six benchmarks.

Performance under different training sets sizes. Following subsection 4.1, we generate the different size of data set that contains the various number of expressions with different constants and the same skeletons. Since DSO (Mundhenk et al., 2021) and GP-based methods are trained from scratch for specific problems, they are not included in this comparison. We train our model, SymbolicGPT and NeSymReS on our training sets separately. As shown in Figure 3, our method with/without CL all outperforms these two baseline methods in terms of R^2 on the SSDNC benchmark. SymbolicGPT and NeSymReS can also improve performance simply by increasing the size of the dataset, which shows that the data-driven approach for SR can continuously improve performance as the size of the dataset increases.

Performance under noisy data. We evaluated the robustness of our and baseline methods to noisy data by adding independent Gaussian noise to the dependent variable, with mean zero and standard deviation proportional to the root-mean-square of the dependent variable in the training data (Mundhenk et al., 2021). In Figure 4, we varied the proportionality constant from 0 (noiseless) to 10^{-1}, following (Mundhenk et al., 2021), and evaluated each algorithm across the SSDNC benchmark.

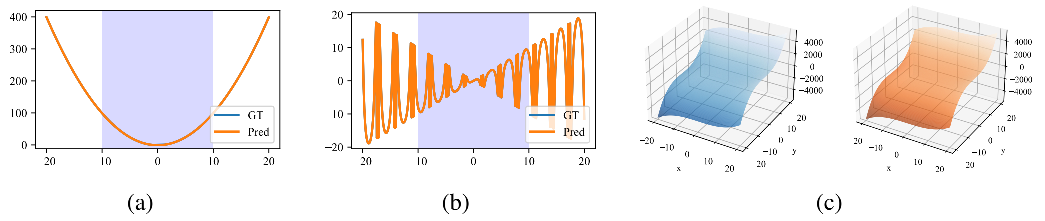


Figure 6: Examples of model predictions using beam search with beam size equaling to 128. The shaded area represents the sampling range. For all functions, x and y were sampled from $[-10, 10]$. 'GT' denotes ground-truth and 'Pred', the model prediction. Specifically,
(a): GT: $\frac{(x^6+x^5)}{(x^4+x^3+x^2+x+1)}$, Pred: $x \cdot (x - \sin(\frac{1}{x}))$;
(b): GT: $x \cdot \cos(\tan(x))$, Pred: $x \cdot \cos(\tan(x))$;
(c): GT: $0.2x^3 + 0.5y^3 - y - x$, Pred: $0.1853x^3 + 0.4974y^3 - 0.8608y$.

Our method with CL is competitive compared with DSO based on reinforcement learning, not overfitting the noise when adding a small amount of noise.

Optimizing constants via restarting BFGS multiple times. We try to reach the global optimum rather than the local optimum by restarting BFGS several times, each time using a different initialization. We set the restart times to 5, 10, 15 and 20. 0 means that the optimization only once. Figure 5 shows that R^2 improves as the number of BFGS restarts increases. We can conclude that multiple initializations can effectively jump out of local optimum in the constant optimization process so as to achieve better fitting accuracy.

Finding mathematically equivalent expressions. By manual observation, we find that our method can generate more expressions with the same symbol as the target expression, which benefits from the high skeleton recovery rate. Interestingly, we also find that the model sometimes predicts a more simple expression compared to the ground truth with fairly high fitting accuracy. For example, for the Keijzer-4 expression $x^3 \cdot \exp(-x) \cdot \cos(x) \cdot \sin(x) \cdot (\sin(x)^2 \cdot \cos(x) - 1)$, our method predicts $-0.634193499960418x^3 \cdot \exp(-x) \cdot \sin(1.83246993155237x + 10.9750345372947)$, which is simpler in the terms of skeleton and achieving a high R^2 through the constant optimization. Additionally, the model can learn some transformation relations of trigonometric functions, e.g., for Constant-7 expression $2\sin(1.3x_1) \cdot \cos(x_2)$, our method predicts $2\cos(1.3x_1 - \frac{\pi}{2}) \cdot \cos(x_2)$, which implicates the transformation relation, i.e., $\sin(x) = \cos(x - \frac{\pi}{2})$.

Out-of domain performance. To test the out-of domain performance of our method, we first run the inference on the points sampled from the training range and then evaluate these predicted functions on points outside the sampling range. In Figure 6, we visualized some of the model predictions about unary and binary expressions. The experimental results show that our model can have good generalization performance outside the sampling domain when predicting more complex expressions, without overfitting the sampled data.

5 Conclusion

We propose a transformer-based method for symbolic regression using a new feature extractor and a joint supervised learning mechanism. Specifically, we leverage a *pure residual MLP* feature extractor to obtain more valuable features on input data points. In order to fundamentally alleviate the ill-posed problem, we propose a joint supervised learning mechanism combining supervised contrastive learning, which strengthens the similarity of feature vectors from the same skeleton. The expression skeleton recovery rate of the proposed method is up to 25% higher than that of recent transformer-based methods. Evaluated on six benchmarks, the results show that our method outperforms current state-of-the-art methods based on reinforcement learning and genetic programming in terms of R^2. It is worth noting that our method is competitive with the reinforcement learning-based method in terms of robustness. Finally, by evaluating the performance of our method outside the sampling range, we showed that it has good extrapolation capabilities.

ACKNOWLEDGMENTS

This work was supported by the Key-Area Research and Development Program of Guangdong Province (No.2019B010107001), and the Key Research Program of the Chinese Academy of Sciences (No.XDPB22).

REFERENCES

Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. Building predictive models via feature synthesis. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pp. 983–990, 2015.

Rohit Batra, Le Song, and Rampi Ramprasad. Emerging materials intelligence ecosystems propelled by machine learning. *Nature Reviews Materials*, 6(8):655–678, 2021.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. PMLR, 2021.

Iwo Błądek and Krzysztof Krawiec. Solving symbolic regression problems with formal constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 977–984, 2019.

Renáta Dubčáková. Eureqa: software review, 2011.

R. Fletcher. Practical methods of optimization. *SIAM Review*, 26(1):143–144, 1984.

Stephanie Forrest. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

Stanton A Glantz and Bryan K Slinker. *Primer of applied regression & analysis of variance, ed*, volume 654. McGraw-Hill, Inc., New York, 2001.

Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*, pp. 70–82. Springer, 2003.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

Krzysztof Krawiec and Tomasz Pawlak. Approximating geometric crossover by semantic backpropagation. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 941–948, 2013.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

Christian Loftis, Kunpeng Yuan, Yong Zhao, Ming Hu, and Jianjun Hu. Lattice thermal conductivity prediction using symbolic regression and machine learning. *The Journal of Physical Chemistry A*, 125(1):435–450, 2020.

Qiang Lu, Jun Ren, and Zhiguang Wang. Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational intelligence and neuroscience*, 2016, 2016.

Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual MLP framework. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=3Pbra-_u76D.

Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

A. Meurer, C. P. Smith, M. Paprocki, O Čertík, and Anthony Scopatz. Sympy: Symbolic computing in python. *PeerJ Computer Science*, 3(3):e103, 2017.

T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen. Symbolic regression via neural-guided genetic programming population seeding. 2021.

Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *Proc. of the International Conference on Learning Representations*, 2021.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Nicolas Staelens, Dirk Deschrijver, Ekaterina Vladislavleva, Brecht Vermeulen, Tom Dhaene, and Piet Demeester. Constructing a no-reference h. 264/avc bitstream-based video quality metric using genetic programming-based symbolic regression. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(8):1322–1333, 2013.

Sheng Sun, Runhai Ouyang, Bochao Zhang, and Tong-Yi Zhang. Data-driven discovery of formulas by symbolic regression. *MRS Bulletin*, 44(7):559–564, 2019.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

Baicheng Weng, Zhilong Song, Rilong Zhu, Qingyu Yan, Qingde Sun, Corey G Grice, Yanfa Yan, and Wan-Jian Yin. Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts. *Nature communications*, 11(1):1–8, 2020.

Matthias Werner, Andrej Junginger, Philipp Hennig, and Georg Martius. Informed equation learning. *arXiv preprint arXiv:2105.06331*, 2021.

Tailin Wu and Max Tegmark. Toward an artificial intelligence physicist for unsupervised learning. *Physical Review E*, 100(3):033311, 2019.

A Details for dataset generation

In this section, we describe the configurations used to generate the training set, validation set, and SSDNC benchmark. We sample each non-leaf node following the unnormalized weighted distribution shown in Table 3 Our base data set approximately contains 100,000 unique expressions' skeletons. Then the different training sets are generated according to the number of expressions with different coefficients. We set this number to 10, 20, 30, 40 and 50. The validation set contains 1K skeletons that are randomly sampled from the base data set and assigned constants that differ from the training set. We generate the more challenging test benchmark SSDNC, which includes approximately 100 unique expression skeletons and 10 re-sampled numerical constants for each skeleton. Source code is available at `https://github.com/AILWQ/Joint_Supervised_Learning_for_SR`.

Table 3: Unnormalised probabilities of unary and binary operators used by the dataset generator.

Operation	Mathematical meaning	Unnormalized probability
add	$+$	10
mul	$\times$	10
sub	$-$	5
div	$\div$	5
sqrt	$\sqrt{\cdot}$	4
exp	$\exp\cdot$	4
ln	$\ln\cdot$	4
sin	$\sin\cdot$	4
cos	$\cos\cdot$	4
tan	$\tan\cdot$	4
pow2	$(\cdot)^2$	4
pow3	$(\cdot)^3$	2
pow4	$(\cdot)^4$	1
pow5	$(\cdot)^5$	1

B Bechmark functions

This section describes the exact functions used to compare our method with the current state-of-the-art methods. In Table 4, we show the name of the benchmark and corresponding expressions.

Table 4: Benchmark functions that we have used in our experiments. Input variables are denoted by x and/or y. We have restricted ourselves only to the univariate and bivariate functions.

Name	Expression
Nguyen-1	$x^3 + x^2 + x$
Nguyen-2	$x^4 + x^3 + x^2 + x$
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$
Nguyen-5	$\sin(x^2)\cos(x) - 1$
Nguyen-6	$\sin(x) + \sin(x + x^2)$
Nguyen-7	$\ln(x+1) + \ln(x^2+1)$
Nguyen-8	$\sqrt{x}$
Nguyen-9	$\sin(x) + \sin(y^2)$
Nguyen-10	$2\sin(x)\cos(y)$
Nguyen-11	x^y
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$
Constant-1	$3.39x^3 + 2.12x^2 + 1.78x$
Constant-2	$\sin x^2 \cdot \cos x - 0.75$
Constant-3	$\sin(1.5x) \cdot \cos(0.5y))$
Constant-4	$2.7x^y$

Constant-5	$\sqrt{1.23x}$
Constant-6	$x^{0.426}$
Constant-7	$2\sin(1.3x)\cdot\cos y$
Constant-8	$\ln(x+1.4)+\ln(x^2+1.3)$
Keijzer-3	$0.3\cdot x\cdot\sin(2\cdot\pi\cdot x)$
Keijzer-4	$x^3\exp(-x)\cos(x)\sin(x)(\sin x^2\cos x-1)$
Keijzer-6	$\frac{x\cdot(x+1)}{2}$
Keijzer-7	$\ln x$
Keijzer-8	$\sqrt{x}$
Keijzer-9	$\ln(x+\sqrt{(x^2+1)})$
Keijzer-10	x^y
Keijzer-11	$xy+\sin((x-1)(y-1))$
Keijzer-12	$x^4-x^3+\frac{y^2}{2}-y$
Keijzer-13	$6\sin(x)\cdot\cos(y)$
Keijzer-14	$\frac{8}{2+x^2+y^2}$
Keijzer-15	$\frac{x^3}{5}+\frac{y^3}{2}-y-x$
R-1	$\frac{(x+1)^3}{x^2-x+1}$
R-2	$\frac{x^5-3x^3+1}{x^2+1}$
R-3	$\frac{x^6+x^5}{x^4+x^3+x^2+x+1}$
Feynman-1	$\frac{\exp(\frac{-x^2}{2})}{\sqrt{2\cdot\pi}}$
Feynman-2	$\frac{\exp(\frac{-(xy^{-1})^2}{2})}{\sqrt{(2\cdot\pi)y}}$
Feynman-3	$x\cdot y$
Feynman-4	$x\cdot y$
Feynman-5	$\frac{1}{2}\cdot x\cdot y^2$
Feynman-6	$\frac{x}{y}$
Feynman-7	$\frac{\sin(\theta_1)}{\sin(\theta_2)}$
Feynman-8	$\frac{x}{y}$
Feynman-9	$\frac{x\cdot y}{2\cdot\pi}$
Feynman-10	$1.5\cdot x\cdot y$
Feynman-11	$\frac{x}{4\cdot\pi\cdot y^2}$
Feynman-12	$\frac{x\cdot y^2}{2}$
Feynman-13	$x\cdot y^2$
Feynman-14	$\frac{x}{2\cdot(1+y)}$
Feynman-15	$\frac{x\cdot y}{2\cdot\pi}$

C Model Hyperparameters

In this section, we give more details about the hyperparameters of our models with/without CL. The full set of hyperparameters can be seen in Table 5.

D Baselines details

Transformer-based methods. For SymbolicGPT (Valipour et al., 2021) and NeSymReS (Biggio et al., 2021), we use the standard hyperparameters provided in the open-source implementation of these methods [5][6]. Some parameters such as the dimension of input, vocabulary size, and so on, are manually adjusted as the dataset changes. Note that we do not change model hyperparameters that may affect performance.

[5] https://github.com/mojivalipour/symbolicgpt
[6] https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales

Table 5: Hyperparameters for our models.

Encoder	
Number of stages	4
Number of pre-blocks for each stage	2
Number of pos-blocks for each stage	2
Original dimension of embedding	64
Dimension expansion for each stage	2
Number of neighbors for FPS	5
Dropout rate	0.1
Decoder	
Dimension of model	512
Number of heads	8
Dropout rate	0.1
Number of layers	8
Dimension of output	60
Training	
Batch size	128
Learning rate	0.0001
Scale weight (w/ CL) λ	0.2
Temperature parameter	0.5
Max norm of gradients	1.0
Inference	
Beam size	128
Re-start times of BFGS	20

Deep symbolic Optimization (DSO). For DSO (Mundhenk et al., 2021), we use the standard parameter settings in the open-source implementation[7]. DSO depends on two main hyper-parameters namely the entropy coefficient λ_H and the risk factor ϵ, and hyperparameters related to genetic programming hybrid methods. The λ_H is used to weight a bonus proportional to the entropy of the sampled expression which is added to the main objective. The intervention in the definition of the final objective depends on the $(1-\epsilon)$ quantile of the distribution of rewards under the current policy. According to the open-source implementation, the chosen hyperparameters are listed in Table 6.

Genetic Programming (GP). For GP-based methods, we opt for the function `SymbolicRegressor` of open-source Python library `gplearn`[8]. Our choices for the hyperparameters are mostly the default values indicated in the library documentation. The detailed settings are reported in Table 7.

Table 6: Hyperparameters for DSO.

Parameter name	Value
Entropy coefficient λ_H	0.05
Risk factor ϵ	0.005
Generations	20
Population size	1000
Crossover probability	0.5
Mutation probability	0.5

Table 7: Hyperparameters for GP.

Parameter name	Value
Population size	1000
Generations	20
Tournament size	20
Crossover probability	0.9
Mutation probability	0.01
Const range	$(-5, 5)$

[7]`https://github.com/brendenpetersen/deep-symbolicregression`
[8]`https://gplearn.readthedocs.io/en/stable/`

E EXPLORING PHYSIC LAWS

Symbolic regression is used by many research communities to advance the study of numerous scientific fields, e.g., Physics (Wu & Tegmark, 2019; Udrescu & Tegmark, 2020), Chemistry (Batra et al., 2021), and Materials (Sun et al., 2019; Wang et al., 2019; Weng et al., 2020; Loftis et al., 2020). Our method shows great potential in recovering some of the laws of physics. As reported in Table 8, we successfully recover all the expressions of two variables in Feynman benchmark test sets. Since π is not included in the dictionary during training, the corresponding value in the recovered expression is predicted to be a decimal.

Symbolic regression algorithms are getting better. Our work will be useful for future data-driven symbolic regression methods. We look forward to the day when a computer helps physicists discover the basic laws of physics, even just like Kepler, discovers a useful and hitherto unknown physics expression through symbolic regression.

Table 8: Recovery expressions on each of the Feynman benchmark of our methods. $\mathcal{U}(a, b, c)$ denotes c random points uniformly sampled between a and b for each input variable. R^2 values are rounded to 5 decimals.

ID	**Expression**	**Constant**	**Dataset**	**Ours**	**Accuracy**(R^2)
I.6.2a	$f = \frac{\exp\left(\frac{-\theta^2}{2}\right)}{\sqrt{2 \cdot \pi}}$	π	$\mathcal{U}(1, 3, 100)$	$0.39894 \exp(-0.5\theta^2)$	0.99999
I.6.2	$f = \frac{\exp\left(\frac{-(\theta\sigma^{-1})^2}{2}\right)}{\sqrt{(2 \cdot \pi)\sigma}}$	π	$\mathcal{U}(1, 3, 100)$	$\frac{0.39888 \exp\left(\frac{-0.49996\theta^2}{\sigma^2}\right)}{\sigma}$	0.99999
I.12.1	$F = \mu N_n$	None	$\mathcal{U}(1, 5, 100)$	μN_n	1.0
I.12.5	$F = q_2 E_f$	None	$\mathcal{U}(1, 5, 100)$	$q_2 E_f$	1.0
I.14.4	$U = \frac{k_{spring} x^2}{2}$	None	$\mathcal{U}(1, 5, 100)$	$\frac{k_{spring} x^2}{2}$	1.0
I.25.13	$V = \frac{q}{C}$	None	$\mathcal{U}(1, 5, 100)$	$\frac{q}{C}$	1.0
I.26.2	$n = \frac{\sin(\theta_1)}{\sin(\theta_2)}$	None	$\mathcal{U}(0, \frac{\pi}{2}, 100)$	$\frac{\sin(\theta_1)}{\sin(\theta_2)}$	1.0
I.29.4	$k = \frac{\omega}{c}$	c	$\mathcal{U}(1, 10, 100)$	$\frac{\omega}{c}$	1.0
I.34.27	$W = \frac{h}{2\pi}\omega$	π, h	$\mathcal{U}(1, 5, 100)$	$0.15924 h\omega$	0.99999
I.39.1	$U = \frac{3}{2} p_r V$	None	$\mathcal{U}(1, 5, 100)$	$\frac{3}{2} p_r V$	1.0
II.3.24	$h = \frac{P_{wr}}{4\pi r^2}$	π	$\mathcal{U}(1, 5, 100)$	$\frac{0.079617 P_{wr}}{r^2}$	1.0
II.8.31	$u = \frac{\epsilon E_f^2}{2}$	ϵ	$\mathcal{U}(1, 5, 100)$	$\frac{\epsilon E_f^2}{2}$	1.0
II.27.18	$u = \epsilon E^2$	ϵ	$\mathcal{U}(1, 5, 100)$	ϵE^2	1.0
II.38.14	$\mu = \frac{Y}{2(1+\sigma)}$	None	$\mathcal{U}(1, 5, 100)$	$\frac{0.5Y}{1+\sigma}$	1.0
III.12.43	$J = \frac{mh}{2\pi}$	π, h	$\mathcal{U}(1, 5, 100)$	$0.15923 mh$	0.99999