A Neural-Guided Dynamic Symbolic Network for Exploring Mathematical Expressions from Data

Wenqiang Li [1 2 3] **Weijun Li** [1 2 3 4 *] **Lina Yu** [1 2 3] **Min Wu** [1 2 3] **Linjun Sun** [1 2 3] **Jingyi Liu** [1 2 3] **Yanjie Li** [1 2 3] **Shu Wei** [1 2 3] **Yusong Deng** [1 2 3] **Meilan Hao** [1 2 3]

Abstract

Symbolic regression (SR) is a powerful technique for discovering the underlying mathematical expressions from observed data. Inspired by the success of deep learning, recent deep generative SR methods have shown promising results. However, these methods face difficulties in processing high-dimensional problems and learning constants due to the large search space, and they don't scale well to unseen problems. In this work, we propose DySymNet, a novel neural-guided **Dy**namic **Sym**bolic **Net**work for SR. Instead of searching for expressions within a large search space, we explore symbolic networks with various structures, guided by reinforcement learning, and optimize them to identify expressions that better-fitting the data. Based on extensive numerical experiments on low-dimensional public standard benchmarks and the well-known SRBench with more variables, DySymNet shows clear superiority over several representative baseline models. Open source code is available at https://github.com/AILWQ/DySymNet.

1. Introduction

Numerous phenomena in the natural world, such as physical laws, can be precisely described using mathematical expressions. Symbolic regression (SR) is an effective machine-learning technique that involves discovering mathematical expressions that describe a dataset with accuracy. Unlike polynomial or neural network-based regression, SR aims to unveil the fundamental principles underlying the data generation process. This method is analogous to how physicists use explicit mathematical models to explain physical phenomena. For instance, Isaac Newton's laws of motion provided a mathematical framework for describing object motion, while Albert Einstein's theory of relativity introduced new equations to explain the behavior of objects in motion. 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 SR is to identify a function f (i.e., $y \approx f(X) : \mathbb{R}^n \to \mathbb{R}$) that best fits the dataset, where the functional form of f is a short closed-form mathematical expression.

SR is a challenging task because the search space of expressions grows exponentially with the length of the expression, while the position and value of numeric further exacerbate its difficulty. The traditional SR methods mainly involve heuristic search methods based on genetic programming (GP) (Forrest, 1993; Koza, 1994; Schmidt & Lipson, 2009; Staelens et al., 2013; Arnaldo et al., 2015; Bładek & Krawiec, 2019). They represent the expression as a binary tree and find high-fitness solutions through iterative evolutions in the large functional search space. While GP-based SR methods have the capability to solve nonlinear problems, they often yield complex expressions and are computationally expensive. Moreover, these methods are known to exhibit high sensitivity to hyperparameters, which can complicate the optimization process.

A more recent line of research has made use of the neural network to tackle the aforementioned shortcomings. Sahoo et al. (2018) proposed an equation learner (EQL), a fully-connected network where elementary functions are used as activation functions. They try to constrain the search space by optimizing a pre-defined network. The limitation of EQL is that the pre-defined architecture of the network limits the complexity of the predicted expression and is not flexible for different specific problems. Recently, reinforcement learning (RL)-based methods (Petersen et al., 2020; Mundhenk et al., 2021) for SR have shown promising

*Corresponding author. [1]AnnLab, Institute of Semiconductors, Chinese Academy of Sciences, Beijing, China [2]School of Electronic, Electrical and Communication Engineering & School of Integrated Circuits, University of Chinese Academy of Sciences, Beijing, China [3]Beijing Key Laboratory of Semiconductor Neural Network Intelligent Sensing and Computing Technology, Beijing, China [4]Center of Materials Science and Optoelectronics Engineering, University of Chinese Academy of Sciences, Beijing, China. Correspondence to: Weijun Li <wjli@semi.ac.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

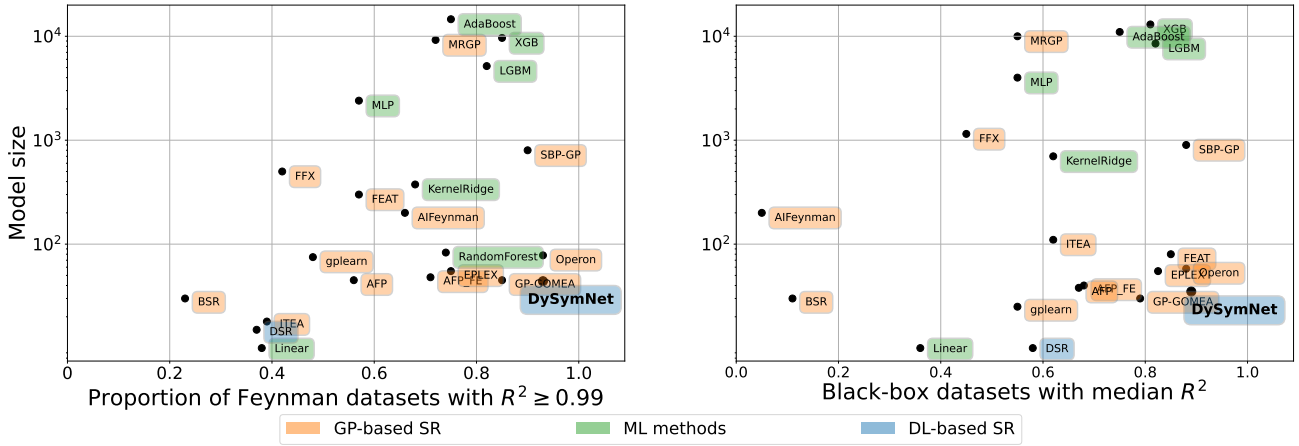


Figure 1. **DySymNet outperforms previous DL-based and GP-based SR methods in terms of fitting accuracy while maintaining a relatively small symbolic model size.** Pareto plot comparing the average test performance and model size of our method with baselines provided by the SRBench benchmark (La Cava et al., 2021), both on *Feynman* dataset (left) and *Black-box* dataset (right). We use the colors to distinguish three families of models: **deep-learning based SR**, **genetic programming-based SR** and **classic machine learning methods** (which do not provide interpretable solutions).

results. They directly search expressions in the large functional space guided by RL. Although it is effective in dealing with low-dimensional problems without constants, they face difficulties in handling high-dimensional problems and constant optimization due to the large search space. Inspired by the success of large-scale pre-training, there has been a growing interest in the SR community for transformer-based models (Valipour et al., 2021; Biggio et al., 2021; Kamienny et al., 2022; Li et al., 2023). These approaches are inductive: they are pre-trained on a large-scale dataset to generate a pre-order traversal of the expression tree in a single forward pass for any new dataset. Thus, transformer-based SR methods possess the advantage of generating expressions quickly. However, they may encounter problems in inference when the given data is out-of-distribution compared to the synthetic data, and can not generalize to unseen input variables of a higher dimension from those seen during pre-training.

Overall, the current mainstream SR methods are mainly generating expression trees from scratch or experience. EQL provides a good idea for reducing the search space and inferencing high-dimensional problems, despite the lack of flexibility. To address these issues, we propose DySymNet, a novel neural-guided **Dy**namic **Sym**bolic **Net**work for SR. We use a controller for sampling various architectures of symbolic network guided by policy gradients, which reduce the vast search space of expressions to that of symbolic network structures while retaining powerful formulaic representation capability.

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

- Proposed DySymNet offers a new search paradigm for SR that searches the symbolic network with various architectures instead of searching expression trees in the vast functional space.
- Proposed DySymNet is adaptable and can converge to a compact and interpretable symbolic expression, with promising capabilities for effectively solving high-dimensional problems that are lacking in mainstream SR methods.
- We demonstrated that DySymNet outperforms several representative baselines across various SR standard benchmarks and the well-known SRBench with more variables. Its state-of-the-art performance surpasses that of the baselines in terms of accuracy and robustness to noise.

2. Related Work

Symbolic regression from scratch SR methods can be broadly classified into two categories: the first category comprises methods that start from scratch for each instance, while the second category involves transformer methods based on large-scale supervised learning. Traditionally, genetic programming (GP) algorithms (Forrest, 1993) are commonly employed to search the optimal expression for given observations (Koza, 1994; Dubčáková, 2011). However, these methods tend to increase in complexity without much performance improvement, and it is also problematic to tune expression constants only by using genetic operators. Recently, the neural networks were used for SR. Martius & Lampert (2016) leverage a pre-defined fully-connected neu-

ral network to identify the expression, which constrains the search space but is not flexible. Petersen et al. (2020) propose deep symbolic regression (DSR), using reinforcement learning (RL) to guide a policy network to directly output a pre-order traversal of the expression tree. Based on DSR, a combination of GP and RL (Mundhenk et al., 2021) was presented, where the policy network is used to seed the GP's starting population. Xu et al. (2023) proposed a Reinforcement Symbolic Regression Machine (RSRM) that masters the capability of uncovering complex math expressions from only scarce data. They employ a Monte Carlo tree search (MCTS) agent and combined with RL to explore optimal expression trees based on measurement data. Although the promising results make it the currently recognized state-of-the-art approach to SR tasks. Nevertheless, the limitations of this method and others from scratch are obvious, namely, the large search space of the expression trees makes them face difficulties in handling high-dimensional problems and constants optimization.

Transformer-based model for symbolic regression In recent years, the transformer (Vaswani et al., 2017) has gained considerable attention in the field of natural language processing. For instance, in machine translation, the transformer model has been extensively employed due to its remarkable performance. Recently, transformer-based models have been highly anticipated for SR. For example, Valipour *et al.* (Valipour et al., 2021) proposed SymbolicGPT that models SR as a machine translation problem. They established a mapping between the data point space and the expression space by encoding the data points and decoding them to generate an expression skeleton on the character level. Similarly, Biggo *et al.* (Biggio et al., 2021) introduces NeSymReS that can scale with the amount of synthetic training data and generate expression skeletons. Kamienny *et al.* (Kamienny et al., 2022) proposed an end-to-end framework based on the transformer that predicts the expression along with its constants. These methods encode the expression skeleton or the complete expression into a sequence, corresponding to the pre-order traversal of the expression tree, then trained on token-level cross-entropy loss. Despite showing promising results, their training approach suffers from the problem of insufficient supervised information for SR. This is because the same expression skeleton can correspond to multiple expressions with different coefficients, leading to ill-posed problems. To address this issue, Li et al. (2023) proposed a joint supervised learning approach to alleviate the ill-posed problem. Overall, transformer-based model for SR has a clear advantage in inference speed compared to methods starting from scratch. Additionally, researchers have made efforts to improve the accuracy of Transformer-based SR methods in searching for expressions (Holt et al., 2022; Liu et al., 2023; Wu et al., 2023; Li et al., 2024). However, their limitations are poor flexibility and model performance dependence on the training set. They may encounter problems when inferring expressions outside the training set distribution.

3. Methodology

The overall approach is visualized in Figure 2. Pseudocode for DySymNet is shown in Appendix A.

3.1. Identify expression from DySymNet

DySymNet architecture In this work, DySymNet is flexible and adaptable in architecture, rather than being fixed and unchanging. The architecture of the symbolic network is controlled by a recurrent neural network (RNN), with further details explained in Section 3.2. Each symbolic network is a fully connected feed-forward network with units representing the building blocks of algebraic expressions. Each layer of the DySymNet is automatically designed by the RNN for a specific SR task, and it is subject to *evolve* over time. In a DySymNet with L layers, there are $L-1$ hidden layers that consist of a linear mapping followed by non-linear transformations. The linear mapping at level l maps the n'-dimensional input $\boldsymbol{h}^{(\ell-1)}$ to the m-dimensional intermediate representation $\boldsymbol{z}$ given by

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)},$$

where $\boldsymbol{h}^{(\ell-1)} \in \mathbb{R}^{n'}$ is the output of the previous layer, with the convention $\boldsymbol{h}^{(0)} = \boldsymbol{x}$. The weight matrix $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{m\times n'}$ and the bias vector $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d}$ are free parameters that learned during training. In practice, we predefine a function library $\mathcal{F}$ of available operators, e.g., $\{\mathrm{Id}, +, -, \times, \sin(\cdot), \cos(\cdot), \exp(\cdot)\}$, to be selected in generating DySymNet. Suppose that we have presently instantiated a specific symbolic network, whereby its ℓ-th layer contains u unary operator units, $f_i : \mathbb{R} \to \mathbb{R}$, for $i = 1, \ldots, u$, and v binary operator units, $g_i : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ for $j = 1, \ldots, v$. The ℓ-th layer output $\boldsymbol{h}^{(\ell)}$ is formed by the concatenation of units outputs

$$\boldsymbol{h}^{(\ell)} := \left(f_1\left(\boldsymbol{z}_1^{(\ell)}\right), f_2\left(\boldsymbol{z}_2^{(\ell)}\right), \ldots, f_u\left(\boldsymbol{z}_u^{(\ell)}\right), \right.$$
$$\left. g_1\left(\boldsymbol{z}_{u+1}^{(\ell)}, \boldsymbol{z}_{u+2}^{(\ell)}\right), \ldots, g_v\left(\boldsymbol{z}_{u+2v-1}^{(\ell)}, \boldsymbol{z}_{u+2v}^{(\ell)}\right)\right).$$

Specifically, the non-linear transformation stage has $n = u + v$ outputs and $m = u + 2v$ inputs. The unary units, $f_1, \ldots, f_u$, $f_i \in \{\mathrm{Id}, \sin(\cdot), \cos(\cdot), \exp(\cdot)\}$, receive the respective component, $z_1, \ldots, z_u$ as inputs. The binary units $g_1, \ldots, g_v$, $g_j \in \{+, -, \times, \div\}$, receive the remaining component, $z_{u+1}, \ldots, z_{u+2v}$, as input in pairs of two. For example, the multiplication unit computes the product of their two input values: $g_j(\boldsymbol{z}_{u+2j-1}, \boldsymbol{z}_{u+2j}) := \boldsymbol{z}_{u+2j-1} \cdot \boldsymbol{z}_{u+2j}$, for $j = 1, \ldots, v$.

The last layer computes the regression values by a linear

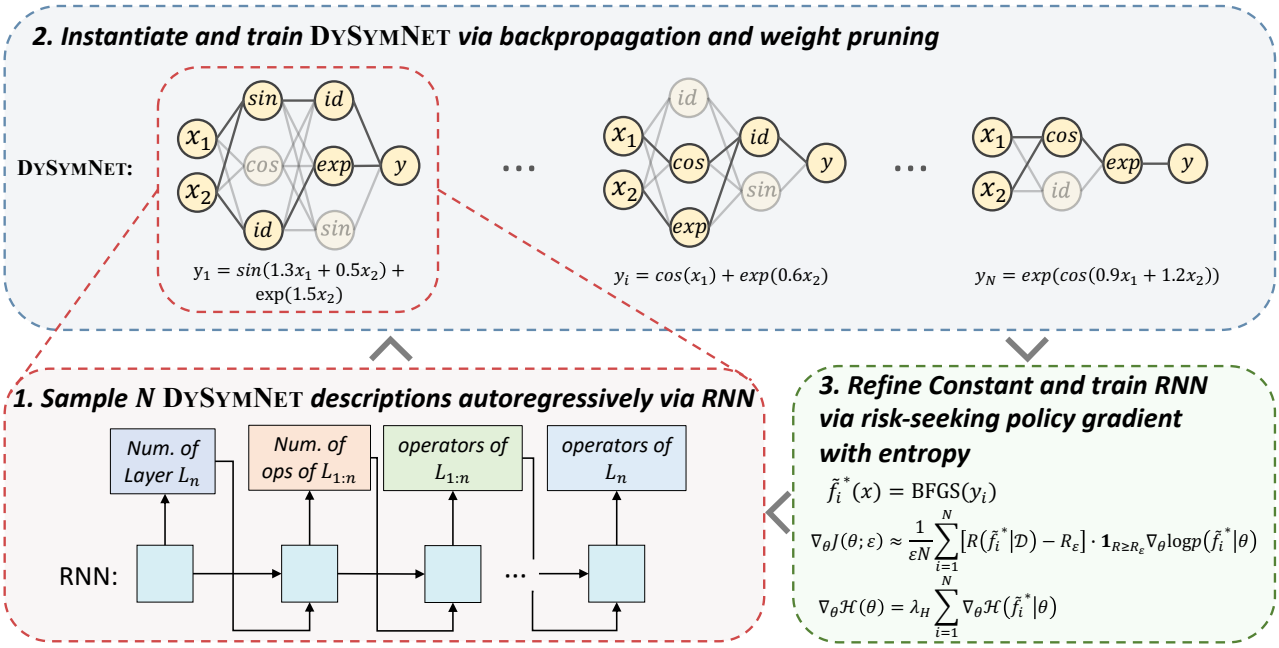


Figure 2. **Overview of neural-guided DySymNet.** First, we sample batch descriptions of DySymNet architecture autoregressively via RNN. Then, we instantiate and train DySymNet through backpropagation and weight pruning. Finally, we use BFGS to refine the constants and train RNN via risk-seeking policy gradient with entropy.

read-out without any operator units

$$\boldsymbol{y}^{(L)} := \boldsymbol{W}^{(L)}\boldsymbol{h}^{(L)} + \boldsymbol{b}^{(L)}.$$

The architecture of the symbolic network is illustrated in Figure 2. We define the expression denoted from the particular symbolic network as $\tilde{f}(\cdot)$.

DySymNet training Similar to conventional fully-connected neural networks, the symbolic network possesses its own set of free parameters, denoted as $\boldsymbol{\Theta} = \{\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)}\}$, which is predetermined and can undergo end-to-end training through backpropagation. While it may be possible to fit a more accurate expression that incorporates a greater number of terms, for a successful SR method, the goal is to find the simplest expression that accurately explains the dataset. This requires a trade-off between accuracy and complexity. To achieve this, two approaches are adopted. First, similar to the transformer-based methods that impose constraints on the pre-order traversal length of the expression tree, we limit the maximum depth of the symbolic network to 5 layers and the maximum width to 6 neurons prior to training. Second, regularization techniques are applied during training to sparsify the weights, followed by pruning according to specific rules to obtain a simplified mathematical expression that accurately explains the dataset. More specifically, given the dataset $\mathcal{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$, the training objective is then given in the following:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|\tilde{f}(\boldsymbol{x}_i) - y_i\|^2 + \lambda \sum_{\ell=1}^{L} |\boldsymbol{\Theta}^{(\ell)}|_r$$

where n indicates the size of training data, the last term represents the regularization term, λ is a switch factor of the regularization term, with a value of $\{0, 1\}$, and L is the number of hidden layers.

Furthermore, we propose an adaptive gradient clipping strategy to enhance the stability of the training process. Specifically, the gradient norm at each time step is calculated as the moving average of the gradient norm within a certain time window. Then, this average value is used to adjust the threshold γ for gradient clipping, which is applied to the gradients of DySymNet. The gradient clipping threshold with window size w and weight factor c during the n-th epoch of training DySymNet is given by: $\gamma = \frac{c}{w}\sum_{i=1}^{w}\sum_{\ell=1}^{L}\|\Theta^{(\ell)}\|_2$, where we set the window size w is 50 and the weight factor c is 0.1. In the ablation experiments in Section 5, we demonstrated the effectiveness of this technique.

Regularization and prune The purpose of regularization is to obtain a sparse symbolic network, which is consistent

with the goal of SR to obtain an interpretable and concise expression. The ideal scenario would involve directly minimizing the L_0 norm, wherein the regularization term penalizes the presence of non-zero weights irrespective of their magnitude, effectively pushing the solution towards sparse representations. However, the utilization of L_0 regularization presents a combinatorial conundrum that is NP-hard to solve, thus rendering it incompatible with gradient descent methods that are typically employed for optimizing neural networks. L_1 regularization has long been recognized as a prominent sparsity technique, which is used in the original EQL (Sahoo et al., 2018). Despite its effectiveness in promoting sparsity, since L_1 regularization also penalizes the magnitude of the weights, $L_{0.5}$ has been proposed to enforce sparsity more strongly without penalizing the magnitude of the weights as much as L_1 (Xu et al., 2010; Zong-Ben et al., 2012). Following (Kim et al., 2020), we use a smoothed version of $L^*_{0.5}$ proposed in (Wu et al., 2014) that has been shown to avoid singularity in the gradient, a challenge often encountered in gradient descent-based optimization when the weights approach zero. The $L^*_{0.5}$ regularizer employs a piecewise function to smooth out the function at small magnitudes:

$$L^*_{0.5}(w) = \begin{cases} |w|^{1/2} & |w| \geq a \\ \left(-\frac{w^4}{8a^3} + \frac{3w^2}{4a} + \frac{3a}{8}\right)^{1/2} & |w| < a \end{cases},$$

where $a \in \mathbb{R}^+$ is the transition point between the standard $L_{0.5}$ function and the smoothed function. After training, we keep all weights $w \in \boldsymbol{W}^{1\ldots L}$ that are close to 0 at 0, i.e., if $|w| < 0.01$ then $w = 0$ to obtain the sparse structure.

3.2. Generate DySymNet with a controller recurrent neural network

We leverage a controller to generate architecture descriptions of DySymNet. The controller is implemented as an RNN. We can use the RNN to generate architecture parameters as a sequence of tokens sequentially. Once the RNN finishes generating an architecture, a DySymNet with this architecture is instantiated and trained. At convergence, we obtain the compact symbolic expression and evaluate it with a reward function. The reward function is not differentiable with respect to the parameters of the controller RNN, θ_c; thus, we optimize θ_c via policy gradient in order to maximize the expected reward of the proposed architectures. Next, we will elaborate on the specific process of generating architecture parameters of DySymNet using the RNN, the reward function definition, and the training algorithm.

Generative process We adopt a Markov Decision Process (MDP) (Sutton & Barto, 2018) modeling the process of generating architectures. An RL setting consists of four components $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ in a MDP. In this view, the list of tokens that the controller RNN predicts can be viewed as a list of actions $a_{1:T} \in \mathcal{A}$ to design an architecture for a DySymNet. Hence, the process of generating architectures can be framed in RL as follows: the agent (RNN) emits a distribution over the architecture $p(a_{1:T}|\theta_c)$, observes the environment (current architecture) and, based on the observation, takes an action (next available architecture parameter) and transitions into a new state (new architecture). The architecture parameters of the DySymNet mainly consist of three parts: **the number of layers**, **the number of operators for each layer**, and **the type of each operator**. As shown in Figure 2, for a specific process, the RNN first samples the number of layers in the network. Then, for each layer, the RNN samples the number of operators in the i-th layer and the type of operators sequentially. At each time step, the input vector $\boldsymbol{x}_t$ of the RNN is obtained by embedding the previous parameter a_{t-1}. Each episode refers to the complete process of using the RNN to sample a DySymNet, instantiating it, and training it to obtain a symbolic expression. This process is performed iteratively until the resulting expression meets the desired performance criteria.

Reward definition As the expression $\tilde{f}(\cdot)$ identified from the DySymNet can only be evaluated at the end of the episode, the reward R equals 0 until the final step is reached. Then, we use the nonlinear optimization algorithm BFGS (Fletcher, 1984) to refine the constants to get a better-fitting expression $\tilde{f}^*(\cdot)$. The constants in the expression $\tilde{f}(\cdot)$ are used as the initial values for the BFGS algorithm. To align with the training objective of the DySymNet, we first calculate the standard fitness measure mean squared error (MSE) between the ground-truth target variable y and the predicted target variable $\tilde{y} = \tilde{f}^*(\boldsymbol{x})$. That is, given a dataset $\mathcal{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$ of size n and candidate expression $\tilde{f}^*(\cdot)$, $\text{MSE}(\tilde{f}^*(\boldsymbol{x}), y) = \frac{1}{n}\sum_{i=1}^n (\tilde{f}^*(\boldsymbol{x}_i) - y_i)^2$. Then, to bound the reward function, we apply a squashing function: $R(\tilde{f}^*|\mathcal{D}) = 1/(1 + \text{MSE})$.

Training the RNN using policy gradients We use the accuracy of $\tilde{f}^*(\cdot)$ as the reward signal and use RL to train the controller RNN. More concretely, to find the optimal architecture of DySymNet, we first consider the standard REINFORCE policy gradient (Williams, 1992) objective to maximize $J_{\text{std}}(\theta_c)$, defined as the expectation of a reward function $R(\tilde{f}^*|\mathcal{D})$ under expressions from the policy:

$$J_{\text{std}}(\theta_c) \doteq \mathbb{E}_{\tilde{f} \sim p(a_{1:T}|\theta_c)} \left[R(\tilde{f}^*|\mathcal{D}) \right]$$

The standard policy gradient objective, $J_{\text{std}}(\theta_c)$, is the desired objective for control problems in which one seeks to optimize the average performance of a policy. However, the final performance in domains like SR is measured by the single or few best-performing samples found during training.

For SR, $J_{\text{std}}(\theta_c)$ is not an appropriate objective, as there is a mismatch between the objective being optimized and the final performance metric. To address this disconnect, we adopt the *risk-seeking policy gradient* proposed in (Petersen et al., 2020), with a new learning objective that focuses on learning only on *maximizing best-case performance*. The learning objective $J_{\text{risk}}(\theta_c;\varepsilon)$ is parameterized by ε:

$$J_{\text{risk}}(\theta_c;\varepsilon) \doteq \mathbb{E}_{\tilde{f}\sim p(a_{1:T}|\theta_c)}\left[R(\tilde{f}^*|\mathcal{D}) \mid R(\tilde{f}^*|\mathcal{D}) \geq R_\varepsilon(\theta_c)\right],$$

where $r_\varepsilon(\theta_c)$ is defined as the $(1-\varepsilon)$-quantile of the distribution of rewards under the current policy. This objective aims to increase the reward of the top ε fraction of samples from the distribution, without regard for samples below that threshold. Lastly, in accordance with the maximum entropy RL framework (Haarnoja et al., 2018), we add an entropy term $\mathcal{H}$ weighted by $\lambda_{\mathcal{H}}$, to encourage exploration:

$$J_{\text{entropy}}(\theta_c;\varepsilon) \doteq \mathbb{E}_{\tilde{f}\sim p(a_{1:T}|\theta_c)}[\mathcal{H}(\tilde{f}|\theta_c) \mid R(\tilde{f}^*|\mathcal{D}) \geq R_\varepsilon(\theta_c)].$$

4. Experimental Settings

In this section, we present our experimental settings and results. We evaluate our proposed DYSYMNET by answering the following research questions (**RQs**):

RQ1: Does DYSYMNET perform better than other SR algorithms on high-dimensional problems?

RQ2: Do individual components of DYSYMNET contribute to overall performance?

RQ3: Does DYSYMNET provide better robustness to noise than other SR algorithms?

RQ4: Can DYSYMNET be used in real life to discover the physical laws?

4.1. Metrics

We assess our method and baselines using the coefficient of determination (R^2):

$$R^2 = 1 - \frac{\sum_i^{N_{\text{test}}}(y_i - \tilde{y}_i)^2}{\sum_i^{N_{\text{test}}}(y_i - \bar{y})^2},$$

where N_{test} is the number of observations, y_i is the ground-truth value for the i-th observation, $\tilde{y}_i$ is the predicted value for the i-th observation, and $\bar{y}$ is the averaged value of the ground-truth. R^2 measures the goodness of fit of a model to the data, and it ranges from 0 to 1, with higher values indicating a better fit between the model and the data.

4.2. Baselines

SRBench (La Cava et al., 2021) has reported the performance of 14 SR methods and 7 machine learning methods on *Feynman*, *Black-box*, and *Strogatz* datasets from PMLB (Olson et al., 2017). We briefly introduce these methods in Appendix D. In addition, we compare the performance of our method against four other mainstream SR algorithms currently:

- **A Unified Framework for Deep Symbolic Regression (uDSR).** A modular, unified framework for symbolic regression that integrates five disparate SR solution strategies (Landajuela et al., 2022).
- **Symbolic Regression via Neural-Guided Genetic Programming Population Seeding (NGGP).** A symbolic regression method combining GP and RL (Mundhenk et al., 2021), the current state-of-the-art algorithm, superseding DSR (Petersen et al., 2020).
- **Neural Symbolic Regression that Scales (NeSymReS).** A transformer-based symbolic regression method based on large-scale supervised training (Biggio et al., 2021).
- **Learning Equations for Extrapolation and Control (EQL).** A method of identifying an expression on a preset fixed symbolic network (Sahoo et al., 2018).

All details for baselines are reported in Appendix C.

4.3. Benchmark problem sets

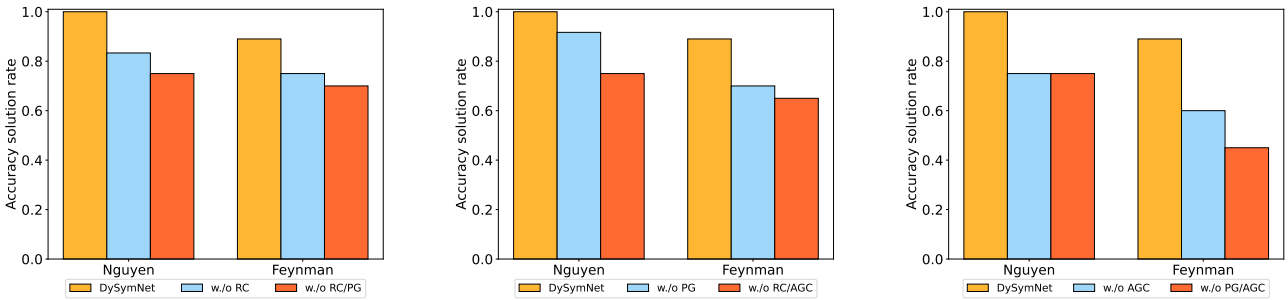
To facilitate quantitative benchmarking of our and other SR algorithms, we conducted evaluations and comparisons on almost all publicly available benchmark datasets in the SR field. We categorized these datasets into two groups, consisting of eight benchmark datasets.

Standard benchmarks $(d \leq 2)$ They are widely used by current SR methods, including the Nguyen benchmark (Uy et al., 2011), Nguyen* (Nguyen with constants) (Petersen et al., 2020), Constant, Keijzer (Keijzer, 2003), Livermore (Mundhenk et al., 2021), R rationals (Krawiec & Pawlak, 2013), Jin (Jin et al., 2019) and Koza (Koza, 1994). The complete benchmark functions are given in Appendix E. Nguyen was the main benchmark used in (Petersen et al., 2020).

SRBench $(d \geq 2)$ SRBench (La Cava et al., 2021) is a living SR benchmark that includes datasets with more variables $(d \geq 2)$. It includes three benchmark datasets: *Feynman*, *Black-box*, and *Strogatz*. The *Feynman* dataset comprises a total of 119 equations sourced from *Feynman Lectures on Physics database* series (Udrescu & Tegmark, 2020). The *Strogatz* dataset contains 14 SR problems sourced from the *ODE-Strogatz database* (La Cava et al., 2016). The *Black-box* dataset comprises 122 problems without ground-truth expression. The input points $(\boldsymbol{x}, y)$ from these three problem sets are provided in Penn Machine Learning Benchmark

Table 1. Average R^2 of DYSYMNET compared with current strong baselines on the standard benchmarks ($d \leq 2$) and SRBench ($d \geq 2$) (La Cava et al., 2021) across 20 independent training runs, where d represents the dimensionality of the problem. 95% confidence intervals are obtained from the standard error between mean R^2 on each problem set.

Data Group	Benchmark	DYSYMNET	uDSR	NGGP	NeSymReS	EQL
		$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$
Standard ($d \leq 2$)	Nguyen	**0.9999**$_{\pm 0.0001}$	0.9989$_{\pm 0.0000}$	0.9848$_{\pm 0.0000}$	0.9221$_{\pm 0.0000}$	0.9758$_{\pm 0.0032}$
	Nguyen*	**0.9999**$_{\pm 0.0000}$	**0.9999**$_{\pm 0.0000}$	**0.9999**$_{\pm 0.0001}$	0.3523$_{\pm 0.0000}$	0.7796$_{\pm 0.0019}$
	Constant	**0.9992**$_{\pm 0.0000}$	0.9989$_{\pm 0.0002}$	0.9989$_{\pm 0.0002}$	0.7670$_{\pm 0.0000}$	0.9817$_{\pm 0.0088}$
	Keijzer	**0.9986**$_{\pm 0.0002}$	0.9968$_{\pm 0.0000}$	0.9912$_{\pm 0.0005}$	0.9358$_{\pm 0.0000}$	0.9446$_{\pm 0.0014}$
	Livermore	**0.9896**$_{\pm 0.0001}$	0.9670$_{\pm 0.0003}$	0.9665$_{\pm 0.0005}$	0.9195$_{\pm 0.0000}$	0.7496$_{\pm 0.0002}$
	R	0.9895$_{\pm 0.0012}$	**0.9999**$_{\pm 0.0000}$	**0.9999**$_{\pm 0.0000}$	0.9368$_{\pm 0.0000}$	0.8588$_{\pm 0.0061}$
	Jin	**1.0000**$_{\pm 0.0000}$	0.9864$_{\pm 0.0002}$	0.9942$_{\pm 0.0001}$	0.9987$_{\pm 0.0000}$	0.9921$_{\pm 0.0002}$
	Koza	**1.0000**$_{\pm 0.0000}$	0.9965$_{\pm 0.0000}$	0.9999$_{\pm 0.0000}$	0.9999$_{\pm 0.0000}$	0.7571$_{\pm 0.0022}$
Data Group	Benchmark	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$	$R^2 \uparrow$
SRBench ($d \geq 2$)	Feynman	**0.9931**$_{\pm 0.0015}$	0.9806$_{\pm 0.0003}$	0.9190$_{\pm 0.0004}$	0.9234$_{\pm 0.0000}$	0.5641$_{\pm 0.0028}$
	Strogatz	**0.9968**$_{\pm 0.0031}$	0.9455$_{\pm 0.0003}$	0.9534$_{\pm 0.0003}$	0.8816$_{\pm 0.0000}$	0.6511$_{\pm 0.0012}$
	Black-box	**0.8908**$_{\pm 0.0028}$	0.6697$_{\pm 0.0010}$	0.6086$_{\pm 0.0021}$	0.4226$_{\pm 0.0000}$	0.4528$_{\pm 0.0107}$



(a) without RC and RC+PG (b) without PG and RC+AGC (c) without AGC and PG+AGC

Figure 3. Ablation study of DYSYMNET on Nguyen and Feynman benchmarks. The three subfigures show the performance comparison of Dysymnet without different components, including Refine Constant (**RC**), Policy Gradient (**PG**), and Adaptive Gradient Clipping (**AGC**).

(PMLB) (Olson et al., 2017) and have been examined in SRBench for the SR task.

5. Results on Benchmark Problem Sets

(RQ1) Fitting accuracy on benchmarks Table 1 reports the performance comparison results of our method with the baselines across two group benchmarks. Notably, the problem sets in the standard benchmarks have at most two variables ($d \leq 2$) and ground-truth expressions that are relatively simple in form. Our method can quickly find the best expression without requiring many rounds of controller iteration. Thus, our method has a clear advantage in inference efficiency compared to GP-based methods, uDSR, and NGGP, which require iterating through thousands of expressions to find the best one. The results indicate that DYSYMNET outperforms the current strong SR baselines in terms of the R^2 while maintaining a lower mean squared error (MSE). Additionally, DYSYMNET successfully recovered all expressions in the Jin dataset ($d = 2$) and Koza dataset. This highlights the importance of reducing the search space to find optimal expressions.

(RQ2) Ablation studies We performed a series of ablation studies to quantify the effect of each of the components. In figure 3, we show the accuracy solution rate for DYSYMNET on the Nguyen and Feynman benchmarks for each ablation. The results demonstrate the effectiveness of these components in improving performance.

(RQ3) Performance under noisy data Since real data are almost always afflicted with measurement errors or other forms of noise, We investigated the robustness of DYSYMNET 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. In Figure 4, we varied the proportionality constant from 0 (noiseless) to 10^{-1} and evaluated each algorithm across Standard benchmarks. *Accuracy solution rate* is defined as the proportion for achieving $R^2 > 0.99$ on the benchmark. The experiments show that our method has the best robustness to noise, and there is no overfitting to noise when a small amount of noise is added, which is significantly better than the NeSymReS based on large-scale supervised training.

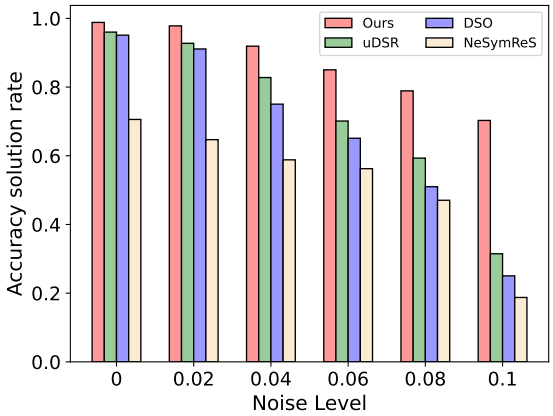


Figure 4. Acuracy solution rate of four approaches on the Standard benchmarks against increasing the noise level.

6. (RQ4) Physical Law Discovery: Free Falling Balls with Air Resistance

It is well known that Galileo discovered the law of free fall of an object by means of two small balls of different masses, which can be formalised as $H(t) = h_0 + v_0 t - \frac{1}{2} g t^2$, where h_0 denotes initial height, v_0 the initial velocity, and g the gravitational acceleration. and g the gravitational acceleration. However, the above laws of physics apply only to the ideal state and cease to apply when air resistance is encountered. Many efforts have been made to uncover the effect of air resistance and to derive mathematical models to describe free-falling objects with air resistance (Clancy, 1975; Lindemuth, 1971; Greenwood et al., 1986).

We performed our experiments on data from 11 different balls, where each ball was experimented with individually. The measurement dataset of a free-falling ball is divided into a training set (records from the first 2 seconds) and a test set (records after 2 seconds). Three physical models derived from mathematical principles were selected as baseline models for this experiment[1,2,3], and the unknown constant coefficient values were estimated using Powell's conjugate direction method. Other methods like uDSR, NGGP, EQL, NeSymRes, and AIFeynman (Udrescu & Tegmark, 2020) are not included as these methods tend to have large generalisation errors due to only a limited number of 20-30 points per training set. The expressions of three baseline physical models are presented at Table 2.

Table 2. Expressions of baseline models, where c_i is unknown constants.

Physics Model	Derived model expression
Model-A	$H(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$
Model-B	$H(t) = c_0 + c_1 t + c_2 e^{c_3 t}$
Model-C	$H(t) = c_0 + c_1 \log(\cosh(c_2 t))$

The physical laws discovered by DYSYMNET from training data are evaluated to the test data and compared with the ground truth. Since the R^2 obtained by Model-B on the test set are all negative, we use Mean square error (MSE) as the evaluation metric. Table 3 reported the MSE of DYSYMNET and other three physical models. The DYSYMNET-discovered physical laws are shown in Appendix Table 6, and their predictions in the last 2 seconds of dropping, compared to the ground truth trajectory, are shown in Appendix Figure 8. It can be concluded that compared to physical laws from human, the physical laws distilled by DYSYMNET leads to a better approximation of the free-falling objects with air resistance.

Table 3. MSE of the test set. The DYSYMNET reaches the best prediction results in most (10 out of 11) cases.

Type	DYSYMNET	Model-A	Model-B	Model-C
baseball	**0.0071**	2.776	94.63	2.623
blue basketball	**0.3713**	0.5022	79.23	2.446
green basketball	0.2395	**0.1029**	85.44	1.242
volleyball	**0.2838**	0.5773	80.98	0.6238
bowling ball	**0.0020**	0.3282	87.06	2.471
golf ball	**0.0380**	0.2117	86.12	1.005
tennis ball	**0.0052**	0.2523	72.28	0.06219
whiffle ball 1	**0.0055**	1.883	65.40	0.2410
whiffle ball 2	**0.4923**	0.6223	58.49	0.9178
yellow whiffle ball	**0.4558**	17.33	44.99	2.544
orange whiffle ball	**0.0090**	0.3827	36.75	3.317
Average	**0.1736**	2.2698	71.94	1.5902

[1]Model-A: https://faraday.physics.utoronto.ca/IYearLab/Intros/FreeFall/FreeFall.html
[2]Model-B: https://physics.csuchico.edu/kagan/204A/lecturenotes/Section15.pdf
[3]Model-C: https://en.wikipedia.org/wiki/Free_fall

7. Conclusion

In this work, we propose DYSYMNET, a neural-guided dynamic symbolic network for SR. We provide a new search paradigm for SR that automatically identify the best expression from different symbolic network guided by RL. Through extensive numerical experiments, we demonstrate the effectiveness of DYSYMNET in solving high-dimensional problems. We also show the value of the application of DYSYMNET to real-world problems, where the physical laws explored by DYSYMNET provide a better approximation of the falling ball compared to three physical models in the falling ball experiment with air resistance. Future research may focus on enhancing the pruning techniques of symbolic networks to improve the comprehensive performance and inference efficiency of the model.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 92370117), and CAS Project for Young Scientists in Basic Research (No. YSBR-090).

Impact Statement

The research presented in this paper is aimed at advancing the field of Machine Learning. After careful consideration, we believe that although any technological advancement has the potential to impact society at large, our work does not entail ethical issues or foresee any potential harm to society. Therefore, we do not specifically highlight any societal consequences here. Should our research be applied in real-world scenarios in the future, we will ensure that it is conducted within a framework of strict ethical standards and social responsibility.

References

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

Arnaldo, I., O'Reilly, U.-M., and Veeramachaneni, K. Building predictive models via feature synthesis. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pp. 983–990, 2015.

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

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

Clancy, L. J. *Aerodynamics*. John Wiley & Sons, 1975.

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

de Silva, B. M., Higdon, D. M., Brunton, S. L., and Kutz, J. N. Discovery of physics from data: Universal laws and discrepancies. *Frontiers in artificial intelligence*, 3:25, 2020.

Dubčáková, R. Eureqa: software review, 2011.

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

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

Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

Greenwood, M. S., Hanna, C., and Milton, R. J. W. Air resistance acting on a sphere: Numerical analysis, strobe photographs, and videotapes. *The Physics Teacher*, 24 (3):153–159, 1986.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Holt, S., Qian, Z., and van der Schaar, M. Deep generative symbolic regression. In *The Eleventh International Conference on Learning Representations*, 2022.

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

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

Keijzer, M. Improving symbolic regression with interval arithmetic and linear scaling. In *Genetic Programming: 6th European Conference, EuroGP 2003 Essex, UK, April 14–16, 2003 Proceedings*, pp. 70–82. Springer, 2003.

Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., and Soljačić, M. Integration of neural network-based symbolic regression in deep learning for

scientific discovery. *IEEE transactions on neural networks and learning systems*, 32(9):4166–4177, 2020.

Kommenda, M., Burlacu, B., Kronberger, G., and Affenzeller, M. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21(3):471–501, 2020.

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

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

La Cava, W., Danai, K., and Spector, L. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence*, 55: 292–306, 2016.

La Cava, W., Singh, T. R., Taggart, J., Suri, S., and Moore, J. H. Learning concise representations for regression by evolving networks of trees. *arXiv preprint arXiv:1807.00981*, 2018.

La Cava, W., Helmuth, T., Spector, L., and Moore, J. H. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation*, 27(3):377–402, 2019.

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

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

Li, W., Li, W., Sun, L., Wu, M., Yu, L., Liu, J., Li, Y., and Tian, S. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Li, Y., Li, W., Yu, L., Wu, M., Liu, J., Li, W., Hao, M., Wei, S., and Deng, Y. Discovering mathematical formulas from data via gpt-guided monte carlo tree search. *arXiv preprint arXiv:2401.14424*, 2024.

Lindemuth, J. The effect of air resistance on falling balls. *American Journal of Physics*, 39(7):757–759, 1971.

Liu, J., Li, W., Yu, L., Wu, M., Sun, L., Li, W., and Li, Y. SNR: Symbolic network-based rectifiable learning framework for symbolic regression. *Neural Networks*, 165:1021–1034, 2023.

Martius, G. and Lampert, C. H. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

McConaghy, T. Ffx: Fast, scalable, deterministic symbolic regression technology. *Genetic Programming Theory and Practice IX*, pp. 235–260, 2011.

Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C. P., Petersen, B. K., et al. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. *Advances in Neural Information Processing Systems*, 34:24912–24923, 2021.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10:1–13, 2017.

Pawlak, T. P., Wieloch, B., and Krawiec, K. Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(3):326–340, 2014.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2020.

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018.

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

Schmidt, M. D. and Lipson, H. Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749, 2008.

Schmidt, M. D. and Lipson, H. Age-fitness pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 543–544, 2010.

Staelens, N., Deschrijver, D., Vladislavleva, E., Vermeulen, B., Dhaene, T., and Demeester, P. 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.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

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

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

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Virgolin, M., Alderliesten, T., and Bosman, P. A. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pp. 1084–1092, 2019.

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation*, 29(2):211–237, 2021.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Wu, M., Li, W., Yu, L., Sun, L., Liu, J., and Li, W. Discovering mathematical expressions through DeepSymNet: A classification-based symbolic regression framework. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Wu, W., Fan, Q., Zurada, J. M., Wang, J., Yang, D., and Liu, Y. Batch gradient method with smoothing l1/2 regularization for training of feedforward neural networks. *Neural Networks*, 50:72–78, 2014.

Xu, Y., Liu, Y., and Sun, H. Rsrm: Reinforcement symbolic regression machine. *arXiv preprint arXiv:2305.14656*, 2023.

Xu, Z., Zhang, H., Wang, Y., Chang, X., and Liang, Y. L 1/2 regularization. *Science China Information Sciences*, 53:1159–1169, 2010.

Zong-Ben, X., Hai-Liang, G., Yao, W., and ZHANG, H. Representative of l1/2 regularization among lq (0¡ q $\leq$ 1) regularizations: an experimental study based on phase diagram. *Acta Automatica Sinica*, 38(7):1225–1228, 2012.

Appendix for "A Neural-Guided Dynamic Symbolic Network for Exploring Mathematical Expressions from Data"

Contents of Appendix:

Method details:

- Appendix A: Pseudocode for DySymNet

Experimental details:

- Appendix B: Computing Infrastructure and Hyperparameter Settings
- Appendix C: Baselines Algorithms Details
- Appendix D: Short Descriptions of SRBench Baselines
- Appendix E: Standard Benchmark Expressions

Results:

- Appendix F: Extrapolation Experiment
- Appendix G: Convergence Analysis
- Appendix H: Free falling Balls with Air Resistance

A. Pseudocode for DySymNet

In this section, we provide the pseudocode for DySymNet. The overall algorithm is detailed in Algorithm 1. Moreover, we provide pseudocode for the function `SampleSymbolicNetwork` (line 4 of Algorithm 1) and `TrainSymbolicNetwork` (line 5 of Algorithm 1) in Algorithm 2 and Algorithm 3, respectively. Specifically, Algorithm 1 describes the overall framework of DySymNet. We use RNN as the controller to sample a batch of symbolic network structures and instantiate them. By training the symbolic network, we can obtain compact expressions. Then, we refine the constants using BFGS and calculate corresponding rewards. When we obtain a batch of rewards, we calculate the ϵ-quantile of the rewards. We use the top $(1-\epsilon)$ rewards to calculate policy gradients and corresponding entropies to update the RNN. This iterative process continues until we reach the threshold we set.

Algorithm 1 A neural-guided dynamic symbolic network for symbolic regression

input learning rate α_c of controller; entropy coefficient $\lambda_{\mathcal{H}}$; risk factor ϵ; batch size N; sample epochs M; reward function R; early stop threshold R_t; SR problem $\mathcal{P}$ consisting of tabular data $\mathcal{D}(X, y)$

output Best fitting expression $\tilde{f}^*$

1: Initialize controller RNN with parameters θ_c, defining distribution over symbolic network $p(\mathcal{S}_{net}|\theta_c)$
2: **for** $i \leftarrow 1$ **to** M **do**
3: **for** $j \leftarrow 1$ **to** N **do**
4: $\mathcal{S}_{net} \leftarrow$ SampleSymbolicNetwork(θ_c) ▷ Sample a symbolic network via controller
5: $\tilde{f}_j \leftarrow$ TrainSymbolicNetwork$(\mathcal{S}_{net}, \mathcal{D})$ ▷ Train the symbolic network
6: $\tilde{f}_j^*, \text{MSE}_j \leftarrow$ BFGS$(\tilde{f}_j, \mathcal{D})$ ▷ Refine constants using BFGS
7: $R(\tilde{f}_j^*) \leftarrow 1/(1+\text{MSE}_j)$ ▷ Compute reward through MSE
8: **if** $R(\tilde{f}_j^*) > R_t$ **then**
9: **return** $\tilde{f}_j^*$ ▷ Return the best expression if reach the threshold
10: $\mathcal{R} \leftarrow \{R(\tilde{f}_j^*)\}_{j=1}^N$ ▷ Compute batch rewards
11: $R_\epsilon \leftarrow (1-\epsilon)$-quantile of $\mathcal{R}$ ▷ Compute reward threshold
12: $\mathcal{F} \leftarrow \{\tilde{f}_j^* : R(\tilde{f}_j^*) \geq R_\epsilon\}$ ▷ Select subset of expressions above threshold
13: $\mathcal{R} \leftarrow \{R(\tilde{f}_j^*) : R(\tilde{f}_j^*) \geq R_\epsilon\}$ ▷ Select corresponding subset of rewards
14: $\hat{g}_1 \leftarrow$ ReduceMean$((\mathcal{R} - R_\varepsilon)\nabla_{\theta_c} \log p(\mathcal{F}|\theta_c))$ ▷ Compute risk-seeking policy gradient
15: $\hat{g}_2 \leftarrow$ ReduceMean$(\lambda_{\mathcal{H}} \nabla_{\theta_c} \mathcal{H}(\mathcal{F}|\theta_c))$ ▷ Compute risk-seeking policy gradient
16: $\theta_c \leftarrow \theta_c + \alpha(\hat{g}_1 + \hat{g}_2)$ ▷ Apply gradients

Algorithm 2 describes the specific process of using RNN to sample a symbolic network structure, which is corresponding to *Generative process* of Section 3.2. In a particular sampling process, we first sample from the number library of layers to determine the current number of layers in the symbolic network. Then, we iterate through each layer to decide the number of operators and operator categories for each layer. Once a complete symbolic network structure has been sampled, we instantiate it. The various libraries of symbolic network structure are reported in Table 4.

Algorithm 3 describes the specific process of training a symbolic network, which is corresponding to Section 3.1. We divide the training process into two stages. In the first stage, we use MSE loss to supervise and converge the weights of the symbolic network to an appropriate interval. In the second stage, we add $L_{0.5}^*$ regularization term to sparse the symbolic network and finally perform pruning to obtain a compact expression. We use the adaptive gradient clipping technology described in Section 3.1 in both stages, which makes the training process of the symbolic network more stable.

B. Computing Infrastructure and Hyperparameter Settings

In this section, we describe additional experimental details, including the experimental environment, hyperparameter settings of our approach and other baselines.

Computing infrastructure Experiments in this work were executed on an Intel Xeon Platinum 8255C CPU @ 2.50GHz, 32GB RAM equipped with NVIDIA Tesla V100 GPUs 32 GB.

Hyperparameter settings The hyperparameters of our method mainly include three parts, which are the hyperparameters of controller RNN, the hyperparameters of symbolic network training, and the hyperparameters of symbolic network

Algorithm 2 Sample a symbolic network autoregressively via controller RNN

input RNN with parameters θ_c; number library for symbolic network layers $\mathcal{L}_L$; number library for operators in each layer $\mathcal{L}_n$; operators library $\mathcal{L}_o$
output Symbolic network $\mathcal{S}_{net}$

$c_0 \leftarrow \vec{0}$ ▷ Initialize RNN cell state to zero
$x \leftarrow \text{rand}()$ ▷ Initialize input randomly
$(\psi_L^{(1)}, c_1) \leftarrow \text{RNN_Cell}(x, c_0; \theta_c)$ ▷ Emit probabilities over $\mathcal{L}_L$; update state
$L \leftarrow \text{Categorical}(\psi_L^{(1)})$ ▷ Sample a value from $\mathcal{L}_L$ as the number of layers
$x \leftarrow \psi_L^{(1)}$ ▷ Take the probabilities over $\mathcal{L}_L$ as input
for $i \leftarrow 1$ **to** L **do**
  $(\psi_n^{(i)}, c_i) \leftarrow \text{RNN_Cell}(x, c_{i-1}; \theta_c)$ ▷ Emit probabilities over $\mathcal{L}_n$; update state
  $n^{(i)} = \text{Categorical}(\psi_n^{(i)})$ ▷ Sample a value from $\mathcal{L}_n$ as the number of operators for layer L^i
  $x \leftarrow \psi_n^{(i)}$ ▷ Take the probabilities over $\mathcal{L}_n$ as input
  $(\psi_o^{(i)}, c_i) \leftarrow \text{RNN_Cell}(x, c_{i-1}; \theta_c)$ ▷ Emit probabilities over $\mathcal{L}_o$; update state
  $o^{(i)} \leftarrow \text{Categorical}(\psi_o^{(i)})$ ▷ Sample n^i operators for layer L^i
  $x \leftarrow \psi_o^{(i)}$ ▷ Take the probabilities over $\mathcal{L}_o$ as input
$\mathcal{S}_{net} \leftarrow \text{Instantiation}(L, n, o)$ ▷ Instantiate a corresponding symbolic network
return $\mathcal{S}_{net}$

Algorithm 3 Training process of symbolic network

input learning rate α_s, regulation weight λ; training epochs n_1; training epochs n_2; window size w; weight clipping factor c; clipping threshold γ; pruning threshold β
output expression $\tilde{f}$

Initialize symbolic network with parameters $\mathbf{\Theta}$
queue $\leftarrow []$ ▷ Initialize gradient norm queue
for $i \leftarrow 1$ **to** n_1 **do**
  $J(\mathbf{\Theta}) \leftarrow \text{MSE}(y, \tilde{y})$ ▷ Compute MSE loss
  queue.append($\|\mathbf{\Theta}\|_2$) ▷ Append current gradient norm to queue
  if len(queue) $> w$ **then**
    queue.popleft()
  $\gamma \leftarrow \frac{c}{w} \sum_{i=1}^{w} \sum_{\ell=1}^{L} \|\mathbf{\Theta}^{(\ell)}\|_2$ ▷ Compute gradient clipping threshold
  $g \leftarrow \text{clip_grad_norm}(\mathbf{\Theta}, \gamma)$ ▷ Clip gradient
  $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha_s \nabla J(\mathbf{\Theta})$ ▷ Apply gradient
for $i \leftarrow 1$ **to** n_2 **do**
  $J(\mathbf{\Theta}) \leftarrow \text{MSE}(y, \tilde{y}) + \lambda L_{0.5}^*(\mathbf{\Theta})$ ▷ Compute MSE loss and regularizaiton loss
  queue.append($\|\mathbf{\Theta}\|_2$) ▷ Append current gradient norm to queue
  if len(queue) $> w$ **then**
    queue.popleft()
  $\gamma \leftarrow \frac{c}{w} \sum_{i=1}^{w} \sum_{\ell=1}^{L} \|\mathbf{\Theta}^{(\ell)}\|_2$ ▷ Compute gradient clipping threshold
  $g \leftarrow \text{clip_grad_norm}(\mathbf{\Theta}, \gamma)$ ▷ Clip gradient
  $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha_s \nabla J(\Theta)$ ▷ Apply gradient
$\tilde{f} \leftarrow \text{Pruning}(\mathbf{\Theta}, \beta)$ ▷ Pruning and using Sympy to convert to corresponding expression
return $\tilde{f}$

structure. For the RNN, the space of hyperparameters considered was learning rate $\in \{0.0003, 0.0006, 0.001\}$, and entropy weight $\lambda_{\mathcal{H}} \in \{0.001, 0.005, 0.01\}$. For the symbolic network training, the space of hyperparameters considered was learning rate $\in \{0.01, 0.05, 0.1\}$, weight pruning threshold $\beta \in \{0.001, 0.01, 0.05\}$, adaptive gradient clipping $\in \{\text{TRUE}, \text{FALSE}\}$, window size $w \in \{10, 50, 100\}$, and add bias $\in \{\text{TRUE}, \text{FALSE}\}$. Adding bias can further improve the performance of the algorithm, but it also increases the complexity of the final expression. We tuned hyperparameters by performing grid search on benchmarks Nguyen-5 and Nguyen-10. We selected the hyperparameters combination with the highest average R^2. The best found hyperparameters are listed in Table 4 and used for all experiments and all benchmark expressions. For the symbolic network structure, there are three categories of hyperparameters that are the number of layers, the number of operators for each layer, and the type of each operator. Notably, these parameters differ from the model training parameters. Each structural parameter is a set that RNN can choose from during sampling, and there is no optimal value for them, as they only determine the upper limit of the symbolic network's representational capacity, similar to the maximum sequence length defined in NGGP (Mundhenk et al., 2021). Thus, the selection of these parameters can be adjusted based on specific scenarios to achieve the best results. For fair comparison, we used the same structural hyperparameter in all experiments, where the operator library was consistent with that of uDSR and NGGP.

Table 4. Hyperparameters for DYSYMNET.

Hyperparameter	**Symbol**	**Value**
RNN Parameters		
Learning rate	α_c	0.0006
Entropy weight	$\lambda_{\mathcal{H}}$	0.005
RNN cell size	–	32
RNN cell layers	–	1
Risk factor	ϵ	0.5
Symbolic network training parameters		
Learning rate	α_s	0.1
Regularization weight	λ	0.005
Weight pruning threshold	β	0.01
Training epochs (stage 1)	n_1	10000
Training epochs (stage 2)	n_2	10000
Adaptive gradient clipping	–	TRUE
Window size	w	50
Add bias	–	FALSE
Symbolic network structure parameters		
Operators library	–	$\{+, -, \times, \sin, \cos, \tan, \exp, \log, \cosh, \bullet^2\}$
Number library of layers	–	$\{1, 2, 3, 4, 5\}$
Number library of operators for each layer	–	$\{1, 2, 3, 4, 5, 6\}$

C. Baselines Algorithms Details

A Unified Framework for Deep Symbolic Regression (uDSR) Landajuela et al. (2022) proposed a modular, unified framework for SR that integrates five disparate SR solution strategies. In this work, we use the open-source code and implementation provided by the authors. To ensure fair comparison, we use the same symbol library as our algorithm and add the CONST token to give it the ability to generate constants. In addition, we used the hyperparameter combination for training reported in the literature.

Deep Symbolic Optimization (NGGP) NGGP (Mundhenk et al., 2021) is a hybrid method that combines RL and GP. They use a policy network to seed the starting population of a GP algorithm. In this work, we use the open-source code and implementation provided by the authors[4]. To ensure fair comparison, we use the same symbol library as our algorithm and

[4] https://github.com/brendenpetersen/deep-symbolicregression

add the CONST token to give it the ability to generate constants. In addition, we used the hyperparameter combination for training reported in the literature.

Neural Symbolic Regression that Scales (NeSymReS) NeSymReS (Biggio et al., 2021) is a transformer-based model for SR. It is trained on synthetic data with a scale of 10M and 100M. In this work, we use the open-source code and implementation provided by the authors[5], following their proposal, and using their released 100M model. To ensure fair comparison, we use the settings reported in the literature, with a beam size of 32 and 4 restarts of BFGS per expression. Due to the limitation of training data, NeSymReS only supports to inference the problem with up to three variables. Following (Landajuela et al., 2022), we utilize the sklearn library (Pedregosa et al., 2011) to select the top-k $(k = 3)$ most relevant features when inferring expressions with more than three variables on SRBench.

Learning Equations for Extrapolation and Control (EQL) EQL (Sahoo et al., 2018) is a fully-connected network where elementary functions are used as activation functions. Whenever reasoning about a new problem, the user needs to first determine the structure of the network. In this work, we use the open-source code and implementation provided by the authors[6].

D. Short Descriptions of SRBench Baselines

Herein, we present concise descriptions of the 14 SR baseline methodologies employed by SRBench, as depicted in Figure 1. These 14 methods include both GP-based and deep learning-based approaches and do not encompass seven other well-known machine learning methods.

- **Bayesian symbolic regression (BSR):** Jin et al. (2019) present a method that incorporates prior knowledge and generates expression trees from a posterior distribution by utilizing a highly efficient Markov Chain Monte Carlo algorithm. This approach is impervious to hyperparameter settings and yields more succinct solutions as compared to solely GP-based methods.

- **AIFeynman:** AIFeynman (Udrescu & Tegmark, 2020) exploits knowledge of physics and the given training data, by identifying simplifying properties (e.g., multiplicative separability) of the underlying functional form. They decompose a larger problem into several smaller sub-problems and resolve each sub-problem through brute-force search.

- **Age-fitness Pareto optimization (AFP):** Schmidt & Lipson (2010) propose a genetic programming (GP) approach that mitigates premature convergence by incorporating a multidimensional optimization objective that assesses solutions based on both their fitness and age.

- **AFP with co-evolved fitness estimates (AFP-FE):** Schmidt & Lipson (2009) expands on (Schmidt & Lipson, 2008) by adding a new fitness estimation approach.

- **ϵ-lexicase selection (EPLEX):** La Cava et al. (2019) present an approach that enhances the parent selection procedure in GP by rewarding expressions that perform well on more challenging aspects of the problem rather than assessing performance on data samples in aggregate or average.

- **Feature engineering automation tool (FEAT):** La Cava et al. (2018) present a strategy for discovering simple solutions that generalize well by storing solutions with accuracy-complexity trade-offs to increase generalization and interpretation.

- **Fast function extraction (FFX):** McConaghy (2011) propose a non-evolutionary SR technique based on pathwise learning (Friedman et al., 2010) that produces a set of solutions that trade-off error versus complexity while being orders of magnitude faster than GP and giving deterministic convergence.

- **GP (gplearn):** The open source library `gplearn` (`https://github.com/trevorstephens/gplearn`) is used in Koza-style GP algorithms. This implementation is very similar to the GP component in NGGP and uDSR.

[5] `https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales`
[6] `https://github.com/martius-lab/EQL`

- **GP version of the gene-pool optimal mixing evolutionary algorithm (GP-GOMEA):** Virgolin et al. (2021) propose combining GP with linkage learning, a method that develops a model of interdependencies to predict which patterns will propagate and proposes simple and interpretable solutions.
- **Deep symbolic regression (DSR):** Petersen et al. (2020) leverage a recurrent neural network to sample the pre-order traversal of the expression guiede by the proposed risk-seeking gradient.
- **Interaction-transformation evolutionary algorithm (ITEA):** de Franca & Aldeia (2021) present a mutation-based evolutionary algorithm based on six mutation heuristics that allows for the learning of high-performing solutions as well as the extraction of the significance of each original feature of a data set as an analytical function.
- **Multiple regression genetic programming (MRGP):** As a cost-neutral modification to basic GP, Arnaldo et al. (2014) describe an approach that decouples and linearly combines sub-expressions via multiple regression on the target variable.
- **SR with Non-Linear least squares (Operon):** Kommenda et al. (2020) improve generalization by including nonlinear least squares optimization into GP as a local search mechanism for offspring selection.
- **Semantic backpropagation genetic programming (SBP-GP):** Virgolin et al. (2019) improve the random desired operator algorithm (Pawlak et al., 2014), a semantic backpropagation-based GP approach, by introducing linear scaling concepts, making the process significantly more effective despite being computationally more expensive.

E. Standard Benchmark Expressions

This section describes the exact expressions in the standard benchmarks ($d \leq 2$) used to compare our method with the current mainstream baselines. In table 5, we show the name of the benchmark, corresponding expressions, and dataset information. For fair comparison, the dataset used in all methods was generated under the same seed.

Table 5: Standard benchmark symbolic regression problem specifications. Input variables are denoted by x and/or y. $U(a, b, c)$ denotes c random points uniformly sampled between a and b for each input variable; training and test datasets use different random seeds.

Name	Expression	Dataset
Nguyen-1	$x^3 + x^2 + x$	$U(-1, 1, 256)$
Nguyen-2	$x^4 + x^3 + x^2 + x$	$U(-1, 1, 256)$
Nguyen-3	$x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 256)$
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U(-1, 1, 256)$
Nguyen-5	$\sin(x^2)\cos(x) - 1$	$U(-3, 3, 256)$
Nguyen-6	$\sin(x) + \sin(x + x^2)$	$U(-3, 3, 256)$
Nguyen-7	$\log(x + 1) + \log(x^2 + 1)$	$U(0, 2, 256)$
Nguyen-8	$\sqrt{x}$	$U(0, 4, 256)$
Nguyen-9	$\sin(x) + \sin(y^2)$	$U(0, 1, 256)$
Nguyen-10	$2\sin(x)\cos(y)$	$U(0, 1, 256)$
Nguyen-11	x^y	$U(0, 1, 256)$
Nguyen-12	$x^4 - x^3 + \frac{1}{2}y^2 - y$	$U(0, 1, 256)$
Nguyen-1^c	$3.39x^3 + 2.12x^2 + 1.78x$	$U(-1, 1, 256)$
Nguyen-5^c	$\sin(x^2)\cos(x) - 0.75$	$U(-1, 1, 256)$
Nguyen-7^c	$\log(x + 1.4) + \log(x^2 + 1.3)$	$U(0, 2, 256)$
Nguyen-8^c	$\sqrt{1.23x}$	$U(0, 4, 256)$
Nguyen-10^c	$\sin(1.5x)\cos(0.5y)$	$U(0, 1, 256)$
Constant-1	$3.39x^3 + 2.12x^2 + 1.78x$	$U(-1, 1, 256)$
Constant-2	$\sin(x^2)\cos(x) - 0.75$	$U(-1, 1, 256)$
Constant-3	$\sin(1.5x)\cos(0.5y)$	$U(0, 1, 256)$

Constant-4	$2.7x^y$	$U\,(0, 1, 256)$
Constant-5	$\sqrt{1.23x}$	$U\,(0, 4, 256)$
Constant-6	$x^{0.426}$	$U\,(0, 2, 256)$
Constant-7	$2\sin(1.3x)\cos(y)$	$U\,(-1, 1, 256)$
Constant-8	$\log(x + 1.4) + \log(x^2 + 1.3)$	$U\,(0, 2, 256)$
Keijzer-3	$0.3 \cdot x \cdot \sin(2 \cdot \pi \cdot x)$	$U\,(-1, 1, 256)$
Keijzer-4	$x^3 \exp(-x) \cos(x) \sin(x)(\sin x^2 \cos x - 1)$	$U\,(-1, 1, 256)$
Keijzer-6	$\frac{x \cdot (x+1)}{2}$	$U\,(-1, 1, 256)$
Keijzer-7	$\ln x$	$U\,(1, 2, 256)$
Keijzer-8	$\sqrt{x}$	$U\,(0, 1, 256)$
Keijzer-9	$\log(x + \sqrt{(x^2 + 1)})$	$U\,(-1, 1, 256)$
Keijzer-10	x^y	$U\,(0, 1, 256)$
Keijzer-11	$xy + \sin((x - 1)(y - 1))$	$U\,(-1, 1, 256)$
Keijzer-12	$x^4 - x^3 + \frac{y^2}{2} - y$	$U\,(-1, 1, 256)$
Keijzer-13	$6\sin(x)\cos(y)$	$U\,(-1, 1, 256)$
Keijzer-14	$\frac{8}{2+x^2+y^2}$	$U\,(-1, 1, 256)$
Keijzer-15	$\frac{x^3}{5} + \frac{y^3}{2} - y - x$	$U\,(-1, 1, 256)$
Livermore-1	$\frac{1}{3} + x + \sin(x^2)$	$U\,(-5, 5, 256)$
Livermore-2	$\sin(x^2)\cos(x) - 2$	$U\,(-1, 1, 256)$
Livermore-3	$\sin(x^3)\cos(x^2) - 1$	$U\,(-1, 1, 256)$
Livermore-4	$\log(x + 1) + \log(x^2 + 1) + \log(x)$	$U\,(0, 2, 256)$
Livermore-5	$x^4 - x^3 + x^2 - y$	$U\,(-1, 1, 256)$
Livermore-6	$4x^4 + 3x^3 + 2x^2 + x$	$U\,(-1, 1, 256)$
Livermore-7	$\sinh(x)$	$U\,(-1, 1, 256)$
Livermore-8	$\cosh(x)$	$U\,(-1, 1, 256)$
Livermore-9	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U\,(-1, 1, 256)$
Livermore-10	$6\sin(x)\cos(y)$	$U\,(-1, 1, 256)$
Livermore-11	$\frac{x^2 x^2}{x+y}$	$U\,(-1, 1, 256)$
Livermore-12	$\frac{x^5}{y^3}$	$U\,(-1, 1, 256)$
Livermore-13	$x^{\frac{1}{3}}$	$U\,(0, 1, 256)$
Livermore-14	$x^3 + x^2 + x + \sin(x) + \sin(x^2)$	$U\,(-1, 1, 256)$
Livermore-15	$x^{\frac{1}{5}}$	$U\,(0, 1, 256)$
Livermore-16	$x^{\frac{2}{5}}$	$U\,(0, 1, 256)$
Livermore-17	$4\sin(x)\cos(y)$	$U\,(-1, 1, 256)$
Livermore-18	$\sin(x^2)\cos(x) - 5$	$U\,(-1, 1, 256)$
Livermore-19	$x^5 + x^4 + x^2 + x$	$U\,(-1, 1, 256)$
Livermore-20	$\exp(-x^2)$	$U\,(-1, 1, 256)$
Livermore-21	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	$U\,(-1, 1, 256)$
Livermore-22	$\exp(-0.5x^2)$	$U\,(-1, 1, 256)$
R-1	$\frac{(x+1)^3}{x^2-x+1}$	$U\,(-1, 1, 256)$
R-2	$\frac{x^5-3x^3+1}{x^2+1}$	$U\,(-1, 1, 256)$
R-3	$\frac{x^6+x^5}{x^4+x^3+x^2+x+1}$	$U\,(-1, 1, 256)$
Jin-1	$2.5x^4 - 1.3x^3 + 0.5y^2 - 1.7y$	$U\,(-1, 1, 256)$
Jin-2	$8.0x^2 + 8.0y^3 - 15.0$	$U\,(-1, 1, 256)$
Jin-3	$0.2x^3 + 0.5y^3 - 1.2y - 0.5x$	$U\,(-1.5, 1.5, 256)$
Jin-4	$1.5\exp(x) + 5.0\cos(y)$	$U\,(-1.5, 1.5, 256)$
Jin-5	$6.0\sin(x)\cos(y)$	$U\,(-1, 1, 256)$
Jin-6	$1.35xy + 5.5\sin((x - 1.0)(y - 1.0))$	$U\,(-1, 1, 256)$
Koza-2	$x^5 - 2x^3 + x$	$U\,(-1, 1, 256)$

Koza-3	$x^6 - 2x^4 + x^2$	$U(-1, 1, 256)$

F. Extrapolation Experiment

Figure 5 provides a qualitative comparison between the performance of DySymNet, NeSymReS (pre-training SR method), MLP (black-box model), and SVR with regards to the ground-truth expression $2sin(1.3x)cos(x)$. The training dataset, represented by the shaded region, comprises 256 data points in the range of [-2, 2], and the evaluation is conducted on the out-of-domain region between [-5, 5]. Although all three models fit the training data very well, our proposed DySymNet outperforms the NeSymReS baseline in fitting the underlying true function, as shown by the out-of-domain performance. Additionally, the result demonstrates a strong advantage of the symbolic approach: once it has found the correct expression, it can predict the whole sequence, whereas the precision of the black-box model deteriorates as it extrapolates further.

G. Convergence Analysis

In Figure 6, the average reward computed by the batch expressions, which are identified from DySymNet, gradually increases with each training step of the policy network RNN, indicating that the direction of the policy gradient is towards maximizing the expected reward value, and the RNN gradually converges to an optimal value.

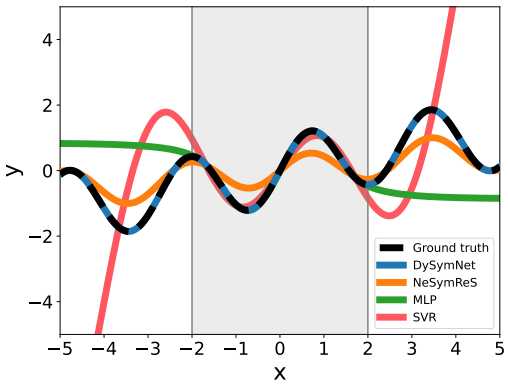


Figure 5. Extrapolation performance comparison with symbolic regression models and black-box models.

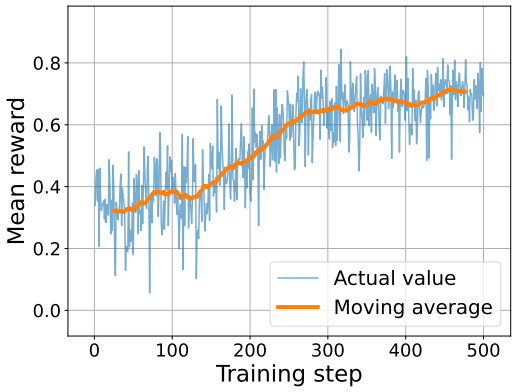


Figure 6. Training curves of RNN for the mean reward of full batch expressions identified from DySymNet.

H. Free Falling Balls with Air Resistance

This section shows more details on exploring the physical laws of free falling balls with air resistance. The experimental ball-drop datasets (de Silva et al., 2020) consist of the records of 11 different types of balls, as shown in Figure 7. These balls were dropped from a bridge and collected at a sampling rate of 30 Hz. Since air resistance affects each ball differently, which in turn leads to different physical laws, we explored the physical laws for each type of ball separately. We split the dataset of each ball into a training set and a test set, where the training set is the recordings from the first 2 seconds and the test set is the recordings from the remaining drop time.

We explore the physical laws on the training data for each type of ball and then compute the error on the test set. Table 6 reports the physical laws discovered by DySymNet and other three baseline physical models. Figure 7 illustrates the fitting trajectories of these discovered physical laws.

Figure 7. The experimental balls dropped from the bridge (de Silva et al., 2020). From left to right: golf ball, tennis ball, whiffle ball 1, whiffle ball 2, baseball, yellow whiffle ball, orange whiffle ball, green basketball, and blue basketball. Volleyball is not shown here.

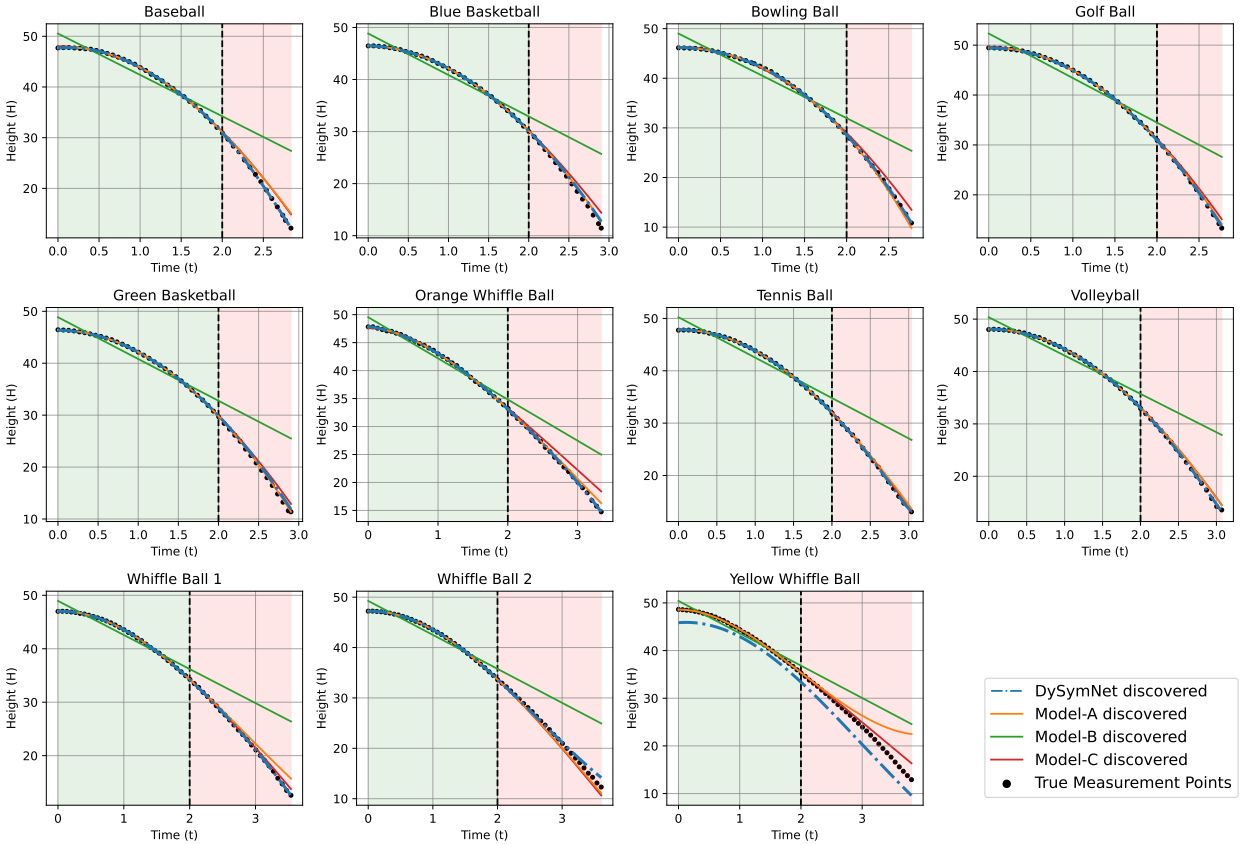


Figure 8. The fitting trajectories of the laws discovered by DySymNet and three baseline physical models on the dataset of 11 balls. Green area represents the extent of the training set (records from the first 2 seconds) and red area represents the extent of the test set (records from the remaining drop time).

Table 6. Discovered physical laws from the motions of free-falling balls by DYSYMNET and other three baseline physical models. Note that these formulas are the raw expressions generated by DYSYMNET, and further simplification helps to improve the parsimony of the expressions.

Ball type	Method	Disvovered expression
Baseball	DYSYMNET	$H(t) = log(cosh(-t * (4.731t - 0.108\sin(4t)) + t + 48.380))$
	Model-A	$H(t) = 47.682 + 1.456t - 5.629t^2 + 0.376t^3$
	Model-B	$H(t) = 45.089 - 8.156t + 5.448\exp(0t)$
	Model-C	$H(t) = 48.051 - 183.467\log(\cosh(0.217t))$
Blue Basketball	DYSYMNET	$H(t) = (0.588 - 0.172t) * (22.209t + 79.077)$
	Model-A	$H(t) = 46.513 - 0.493t - 3.912t^2 + 0.03t^3$
	Model-B	$H(t) = 43.522 - 7.963t + 5.306\exp(0t)$
	Model-C	$H(t) = 46.402 - 84.791\log(\cosh(0.319t))$
Bowling Ball	DYSYMNET	$H(t) = -1.310t(3t + \log(\cosh((\log(\cosh(t))))))$ $+ \log(\cosh(\log(\cosh(\cos(2.698t))))) + 46.111$
	Model-A	$H(t) = 46.139 - 0.091t - 3.504t^2 - 0.431t^3$
	Model-B	$H(t) = 43.336 - 8.525t + 5.676\exp(0t)$
	Model-C	$H(t) = 46.342 - 247.571\log(\cosh(0.189t))$
Golf Ball	DYSYMNET	$H(t) = (0.592 - 0.182t) * (26.163t + 83.442)$
	Model-A	$H(t) = 49.413 + 0.532t - 5.061t^2 + 0.102t^3$
	Model-B	$H(t) = 46.356 - 8.918t + 5.964\exp(0t)$
	Model-C	$H(t) = 49.585 - 178.47\log(\cosh(0.23t))$
Green Basketball	DYSYMNET	$H(t) = (0.492 - 0.146t) * (27.582t + 94.230)$
	Model-A	$H(t) = 46.438 - 0.34t - 3.882t^2 - 0.055t^3$
	Model-B	$H(t) = 43.512 - 8.043t + 5.346\exp(0t)$
	Model-C	$H(t) = 46.391 - 124.424\log(\cosh(0.263t))$
Orange Whiffle Ball	DYSYMNET	$H(t) = -t(t(0.368 - 0.191t^2) + 2t) - t - \exp(t) + 48.929$
	Model-A	$H(t) = 47.836 - 1.397t - 3.822t^2 + 0.422t^3$
	Model-B	$H(t) = 44.389 - 7.358t + 5.152\exp(0t)$
	Model-C	$H(t) = 47.577 - 12.711\log(\cosh(0.895t))$
Tennis Ball	DYSYMNET	$H(t) = \log(\cosh(3.616t^2 - 33.533)) + \cos(t) + 14.035$
	Model-A	$H(t) = 47.738 + 0.658t - 4.901t^2 + 0.325t^3$
	Model-B	$H(t) = 45.016 - 7.717t + 5.212\exp(0t)$
	Model-C	$H(t) = 47.874 - 114.19\log(\cosh(0.269t))$
Volleyball	DYSYMNET	$H(t) = t(t(-0.590 * t - 4.393) - \cos(t)) + \exp(t) + 47.089$
	Model-A	$H(t) = 48.046 + 0.362t - 4.352t^2 + 0.218t^3$
	Model-B	$H(t) = 45.32 - 7.317t + 5.037\exp(0t)$
	Model-C	$H(t) = 48.124 - 107.816\log(\cosh(0.27t))$
Whiffle Ball 1	DYSYMNET	$H(t) = \log(\cosh((-1.831t - 6.155) * \log(\cosh(t))$ $+ \log(\cosh(\log(\cosh(\log(\cosh(t))))))) + 47.719))$
	Model-A	$H(t) = 46.969 + 0.574t - 4.505t^2 + 0.522t^3$
	Model-B	$H(t) = 44.259 - 6.373t + 4.689\exp(0t)$
	Model-C	$H(t) = 47.062 - 34.083\log(\cosh(0.462t))$
Whiffle Ball 2	DYSYMNET	$H(t) = 21.017\sin(0.607t + 7.846) + 26.213$
	Model-A	$H(t) = 47.215 + 0.296t - 4.379t^2 + 0.421t^3$
	Model-B	$H(t) = 44.443 - 6.744t + 4.813\exp(0t)$
	Model-C	$H(t) = 47.255 - 38.29\log(\cosh(0.447t))$
Yellow Whiffle Ball	DYSYMNET	$H(t) = t + 33.507(0.337\cos(0.365t) + 1)^3 - 34.227$ $+48.6092\log(\cosh(t))/(\log(\cosh(t)) + 3.065)$
	Model-A	$H(t) = 48.613 - 0.047t - 4.936t^2 + 0.826t^3$
	Model-B	$H(t) = 45.443 - 6.789t + 4.973\exp(0t)$
	Model-C	$H(t) = 48.594 - 12.49\log(\cosh(0.86t))$