Complexity-Aware Deep Symbolic Regression with Robust Risk-Seeking Policy Gradients

Zachary Bastiani [1,2] **Robert M. Kirby** [1,2] **Jacob Hochhalter**[3] **Shandian Zhe** [1]
[1] Kahlert School of Computing [2] Scientific Computing and Imaging Institute
[3] Department of Mechanical Engineering, University of Utah

Abstract

We propose a novel deep symbolic regression (DSR) approach to enhance the robustness and interpretability of data-driven mathematical expression discovery. Existing DSR methods are built on recurrent neural networks, solely guided by data fitness, and potentially meet tail barriers that can zero out the policy gradient, causing inefficient model updates. To address these issues, we design a decoder-only architecture that performs attention in the frequency domain and introduce a dual-indexed position encoding to conduct layer-wise generation. Second, we propose a Bayesian information criterion (BIC)-based reward function that can automatically adjust the trade-off between expression complexity and data fitness, without the need for explicit manual tuning. Third, we develop a ranking-based weighted policy update method that eliminates the tail barriers and enhances training effectiveness. Extensive benchmarks and systematic experiments demonstrate the advantages of our approach. We have released our implementation at `https://github.com/ZakBastiani/CADSR`.

1 INTRODUCTION

Symbolic regression (SR) (Schmidt and Lipson, 2009; Jobin et al., 2019; Rudin, 2019) is an important research direction for achieving interpretability in machine learning. Given a dataset that records the input and output of a complex system of interest, symbolic regression seeks to discover a simple, concise equation that reveals the system's underlying mechanism — thereby enhancing our understanding and ensuring the model's reliability.

Genetic programming (GP) (Koza, 1994; Randall et al., 2022; Burlacu et al., 2020) has long been the dominant approach for symbolic regression. However, GP is known to be computationally expensive and time-consuming due to its evolutionary nature. Deep Symbolic Regression (DSR) (Petersen et al., 2019) and its variants (Tenachi et al., 2023; Jiang et al., 2024) represent a recent breakthrough by training a recurrent neural network (RNN) to generate expressions from data efficiently. While DSR has demonstrated success across many SR benchmarks, the RNN-based architecture can struggle with capturing long-range dependencies and is prone to vanishing gradients (Hochreiter, 1998), especially in large expression trees. In addition, DSR methods typically rely on data-fitting rewards, which can lead to overly complex expressions and overfitting — particularly in noisy settings. Furthermore, due to the usage of the reward difference as the weights in the policy gradients, DSR takes the risk of meeting tail barriers, *i.e.,* zero policy gradients, which can result in inefficient model updates.

To address these issues, we introduce a robust, complexity-aware deep symbolic regression method (CADSR). It is worth noting that CADSR and the DSR framework aim to learn a data-specific expression generator from scratch, rather than utilize a pretrained supervised model mapping datasets directly to expressions (Valipour et al., 2021; Shojaee et al., 2023). Our major contributions are as follows.

Expression Model: We design a decoder-only architecture that performs attention in the frequency space. We apply the discrete cosine transform to the embeddings to obtain their frequency representations. High-frequency components — often attributed to noise, outliers, or excessive parameter perturbations — are removed, and attention is computed over the resulting low-frequency components. An inverse transform is

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

then applied to recover updated embeddings in the original space. Next, to enhance the model's ability to capture positional information within the expression tree, we introduce a dual-indexed positional encoding based on both the depth and horizontal locations of each token. We use a breadth-first search (BFS) strategy to generate the expression tree layer by layer, which offers both computational efficiency and implementation simplicity.

Reward Design: We propose a Bayesian Information Criterion (BIC)-based reward function that computes the model evidence to evaluate the trade-off between the expression complexity and data fit. This enables the learning process to explicitly optimize the trade-off between interpretability and accuracy, avoiding overly complex expressions that tend to overfit — particularly in the presence of noise. BIC is grounded in Bayesian model selection theory (Wasserman, 2000) and is closely related to the principle of Minimum Description Length (MDL) (Rissanen, 1978), making it a principled and robust criterion. Moreover, BIC introduces no additional hyperparameters, allowing for automatic trade-off adjustment to maximize model evidence, and eliminating the need for manual tuning.

Policy Gradient: We propose a novel risk-seeking policy gradient method, integrated with Group Relative Policy Optimization (GRPO) (Shao et al., 2024). Unlike DSR, which uses reward differences as gradient weights, our approach employs a ranking-based linear weighting scheme to perform step-wise reward mapping. This design not only preserves distinctions among the top-ranked candidates but also avoids any tail barriers or partial tail barriers in the gradient updates. As a result, our method can effectively utilize top-performing samples for model updates, minimizing inefficient or overly exploratory behavior. Furthermore, integration with GRPO enables repeated and reliable optimization steps for each batch of sampled expressions within a trust region, thereby enhancing training effectiveness while reducing sample complexity.

Experiments: We evaluated CADSR on the standard SR benchmark, a practical fracture mechanical application, and many ablation studies. In addition to DSR, we compared with seventeen other popular and/or state-of-the-art SR methods and several commonly used machine learning approaches. The performance of CADSR in both symbolic discovery and prediction accuracy is consistently among the best. In particular, the symbolic discovery rate of CADSR is the highest when data includes significant noises. In all the cases, CADSR generates the most interpretable expressions, while maintaining a high level of accuracy. CADSR outperforms the most comparable model, DSR, in all categories showing that it is a direct improvement. Extensive ablation studies further demonstrate the effectiveness of each component of our method.

2 BACKGROUND

Given a set of input and output examples collected from the target system, denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, symbolic regression aims to identify a concise expression that characterizes the input-output relationship, such as $y = \sin(2\pi x_1) + \cos(2\pi x_2)$. Deep symbolic regression (DSR) (Petersen et al., 2019) discovers equations via an RNN-based reinforcement learning approach (Sutton and Barto, 2018), which can be broken down into four parts: environment, actor, reward, and policy.

The environment is designed to be the creation of an expression tree that represents a specific equation. Expression trees are directed trees where each node holds a token from the available list of operations and variables, *e.g.*, $\{+, -, \times, x_1\}$. Expression trees are built by selecting nodes in a preorder traversal of the trees. These trees are many-to-one mappings to the mathematical expressions, which can increase the search space but prevent generating invalid expressions.

The actor is an RNN that predicts a categorical distribution of the available tokens for each node in the expression tree based on the hidden state of the RNN and the sibling and parent of the current node. Each token is randomly sampled from the categorical distribution. Additional rules are applied to the sampling process to prevent the selection of redundant operations or variables.

The reward function and policy drive the actor to explore and exploit the complex environment. In DSR, the reward function is a direct measurement of the data fitness of the generated expression,

$$R(\tau) = \frac{1}{1 + \text{NRMSE}}, \tag{1}$$

where τ denotes the expression, NRMSE represents the normalized root-mean-square error, and is defined as $\text{NRMSE} = \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^n (y_i - \tau(\mathbf{x}_i))^2}$ where σ_y is the standard deviation of the training output in the dataset. DSR applies a risk-seeking policy to update the actor model according to its best predictions. Specifically, at each step, DSR samples a batch of expressions, ranks their rewards, and selects the top $\alpha\%$ expressions to compute a policy gradient,

$$\begin{aligned}\nabla_\theta J_{\text{risk}}(\theta;\alpha) = \frac{1}{\alpha B/100}\sum\nolimits_{i=1}^{B}[R(\tau^{(i)}) - R_\alpha] \\ \cdot \mathbf{1}_{R(\tau^{(i)})\geq R_\alpha}\nabla_\theta \log(p(\tau^{(i)}|\theta)),\end{aligned} \tag{2}$$

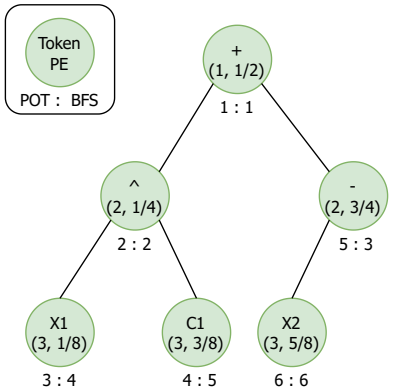


Figure 1: Expression tree for $y = x_1^{c_1} + \sin(x_2)$. POT and BFS denote the node order for preorder traversal and breadth-first-search, respectively.

where B is the batch size, $\mathbf{1}_{(\cdot)}$ is an indicator function, $\tau^{(i)}$ is the i-th expression in the batch, θ denotes the RNN parameters, $\log(p(\tau^{(i)}|\theta))$ is the probability of $\tau^{(i)}$ being sampled by the current RNN, R_α is the $1-\alpha/100$ quantile of the rewards in the batch. Accordingly, all the equations below top $\alpha\%$ will not influence the update of the actor. This policy enables the actor to generate lower-performing equations without negatively impacting its overall performance, as we only care about the top performed expressions. This allows for more unrestrained exploration and targeted exploitation of the top performers.

3 METHOD

3.1 Decoder-Only Expression Generator

We first design a decoder-only transformer architecture for expression generation. Unlike RNNs, which rely on a single hidden state to summarize information across all previous nodes, making them susceptible to vanishing gradients and poor at capturing long-range dependencies (Hochreiter, 1998), transformers compute dependencies explicitly between all nodes through the attention mechanism. This allows them to prevent gradient vanishing and effectively capture various short- and long-range dependencies.

Dual-Indexed Position Encoding. We use one-hot encoding to represent each token, such as operations, variables, and constants, and adopt an autoregressive approach to generate each node in the expression tree. Instead of assigning a single pre-order index to represent each node, we introduce a dual-indexed position encoding (DPE) to more effectively capture the tree's structural information. Specifically, given a particular node, we consider both its depth d, and horizontal location h, in the tree. To align the horizontal locations of the nodes across different layers, we propose the following design. Denote the horizontal location of the parent node by h_p, if the node is the left child, we assign its horizontal position as $h = h_p - h_p/2^d$, otherwise we assign $h = h_p + h_p/2^d$. In this way, the parent node will be in between its two children horizontally, which naturally reflects the tree structure. The horizontal position for the root node is set to $1/2$. See Fig. 1 for an illustration. We develop a recursive algorithm to efficiently calculate the horizontal positions of all the nodes, as listed in Appendix Algorithm 2. Given d and h, we construct a 2D-dimensional encoding,

$$\mathrm{DPE}(d,h)_{2i} = \sin\left(\frac{d}{10000^{(4i/D)}}\right), \quad (3)$$

$$\mathrm{DPE}(d,h)_{2i+1} = \cos\left(\frac{d}{10000^{(4i/D)}}\right), \quad (4)$$

$$\mathrm{DPE}(d,h)_{D+2j} = \sin\left(\frac{h}{10^{(4j/D)}}\right), \quad (5)$$

$$\mathrm{DPE}(d,h)_{D+2j+1} = \cos\left(\frac{h}{10^{(4j/D)}}\right) \quad (6)$$

where $0 \le i, j \le \lfloor D/2 \rfloor$. The embedding of each tree node is constructed as the positional encoding plus the one-hot encoding of the token.

Robust Attention in Frequency Space. To capture a diverse range of complex, short- and long-range dependencies among tree nodes, we construct multiple self-attention layers. Inspired by the recent DCT-Former (Scribano et al., 2023) developed for large language models, we propose performing attention in the frequency domain to improve both learning robustness and efficiency. Let the embeddings of the current tree nodes be denoted as $\mathbf{H} = [\mathbf{h}_0, \ldots, \mathbf{h}_{N-1}]^\top$, where each $\mathbf{h}_n \in \mathbb{R}^r$ $(0 \le n \le N-1)$. We treat these embeddings as samples of a function with r-dimensional outputs at N discrete input locations. We then apply a Discrete Cosine-Transform (DCT) to extract their frequency-domain representations $\widehat{\mathbf{H}} = [\widehat{\mathbf{h}}_0, \ldots, \widehat{\mathbf{h}}_{N-1}]^\top$, where

$$\widehat{\mathbf{H}} = \mathbf{C}\mathbf{H}, \quad C_{k,n} = \alpha_k \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right]. \quad (7)$$

Here, $0 \le k, n \le N-1$, and the normalization constant α_k is defined as $\alpha_k = \sqrt{1/N}$ for $k = 0$, and $\alpha_k = \sqrt{2/N}$ otherwise. The DCT decomposes the function into a weighted sum of cosine basis functions oscillating at different frequencies $k = 0, \ldots, N-1$. Each $\widehat{\mathbf{h}}_k \in \mathbb{R}^r$ represents the coefficient of the cosine basis at frequency k. One advantage of using the DCT is that the resulting frequency-domain representation remains entirely in the real-valued domain. In contrast, applying a Fourier transform would yield complex-valued representations, requiring special handling to avoid numerical issues. To improve robustness and efficiency during training, we apply a low-pass filter by removing the high-frequency components, which are often

associated with noise, outliers in data, or excessive perturbations to the model parameters. This filtering helps to denoise the representations and stabilize learning. Specifically, we retain only the first M frequency components, denoted as $\widehat{\mathbf{H}}_{1:M} = [\widehat{\mathbf{h}}_0, \ldots, \widehat{\mathbf{h}}_{M-1}]^\top$, where M is a tunable hyperparamter. We perform attention on $\widehat{\mathbf{H}}_{1:M}$:

$$\begin{aligned}\widehat{\mathbf{A}} &= \text{Attention}(\widehat{\mathbf{H}}_{1:M}; \mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= \text{softmax}\Big(\frac{(\widehat{\mathbf{H}}_{1:M}\mathbf{Q})(\widehat{\mathbf{H}}_{1:M}\mathbf{K})^\top}{\sqrt{r}}\Big)(\widehat{\mathbf{H}}_{1:M}\mathbf{V}). \qquad (8)\end{aligned}$$

We perform this self-attention in the frequency domain multiple times to fully capture the rich, complex dependencies of the frequency components. Then we transform the representation back to the original domain using inverse DCT:

$$\mathbf{H} \leftarrow \mathbf{C}^\top[\widehat{\mathbf{A}}; \mathbf{0}], \qquad (9)$$

where the zero-padding restores the original dimensionality by appending $N - M$ frequency components. To predict the token distribution for the next node, we apply a linear layer followed by a softmax function. The overall architecture of our model is illustrated in Appendix Fig. 6.

Expression Sampling. Starting from the root node, we generate the expression tree in a layer-wise, breadth-first-search (BFS) order, auto-regressively expanding the nodes layer by layer. This generation scheme aligns naturally with our dual-indexed positional encoding, enabling effective integration of the spatial relationships between the previously generated nodes and the current target node for token sampling. Moreover, since the tree grows in a fixed layer-by-layer fashion, the ordering of previously generated nodes remains unchanged throughout the process. As a result, we only need to perform vectorization once prior to sampling, making the approach both computationally efficient and straightforward to implement.

Given the predicted token distribution, we first mask out invalid tokens and then sample from the remaining candidates. Once the sampled tokens form a valid expression, the generation process terminates and returns the resulting expression. The complete sampling procedure is summarized in Algorithm 1 in the Appendix.

3.2 BIC Reward Function

Interpretability is a central motivation for symbolic regression. However, when using only data fit as the reward signal — such as in DSR — the model tends to produce overly complex expressions that overfit the data, especially in the presence of noise, which is common in real-world applications. This overfitting undermines the interpretability of the learned expressions. To address this issue, we adopt the Bayesian Information Criterion (BIC) (Schwarz, 1978) to define a new reward function. BIC is theoretically grounded in Bayesian model selection (Wasserman, 2000) and closely related to the Minimum Description Length (MDL) principle (Rissanen, 1978), effectively serving as its approximation.

As a principled and robust criterion, BIC computes the model evidence by integrating out the model parameters, which in essence evaluates the trade-off between the model complexity and data fit in terms of generalizability. In our design, we define the model complexity k as the sum of the number of nodes in the expression tree and the number of constant tokens. This inclusion serves two purposes: (1) Since the values of constant tokens are unknown a priori and must be estimated from data, they effectively act as model parameters, introducing additional degrees of freedom; (2) penalizing the number of constant tokens discourages their excessive use, which would otherwise increase the computational cost of optimizing their values. The BIC reward is defined as:

$$\text{BIC}(\tau) = k\log(S) - 2\log(p(\tau)), \qquad (10)$$

where $\log(p(\tau)) = \sum_{i=1}^{S} \log \mathcal{N}(y_i|\tau(\mathbf{x}_i), \sigma^2)$ is the Gaussian log-likelihood, σ^2 is the variance of the training outputs, and S is the number of training samples. To accommodate more complex data distributions, $p(\tau)$ can be replaced with a Student-t likelihood or a mixture of Gaussians. Notably, the BIC reward introduces no additional tuning hyperparameters. This allows for automatic balancing between model complexity and data fit, enabling the actor to directly optimize for the expression's generalizability without manual intervention.

3.3 Robust Risk-Seeking Policy

While our BIC reward (10) accounts for expression complexity, its value is unbounded and can grow arbitrarily large. As a result, directly applying it in a risk-seeking policy framework, as done in DSR, can introduce unpredictable high variance and lead to highly unstable learning. Even after normalization, a few outlier expressions can dominate the reward distribution, causing the majority of well-performing samples to contribute little or nothing to the policy update.

To mitigate this issue, a commonly used strategy, which is also adopted in DSR, is to introduce a continuous mapping that maps the reward value to a bounded domain. For example, DSR uses the mapping $f(z) = \frac{1}{1+z}$ where $z \in [0, \infty]$, which maps the unbounded NRMSE (in $[0, \infty]$) to the range $(0, 1]$; see (1). Another choice can be the sigmoid function, $s(z) = \frac{1}{1+e^{-z}}$ that

maps from $(-\infty, \infty)$ to $(0, 1)$. In this way, we can control the variance and improve learning stability. However, this strategy introduces another challenge. That is, the reinforcement learning can encounter a *tail barrier*, defined as follows.

Definition 3.1 (Tail barrier)**.** Let $\alpha \in [0, 1)$. A risk seeking policy meets a α-tail barrier if the top $\alpha\%$ rewards of the sampled actions (*e.g.*, expression trees) are all equal to R_α.

Lemma 3.2. *Given any continuous mapping f that can map unbounded reward function values to a bounded domain (e.g., $(-\infty, \infty) \to [0, 1]$), suppose the reward function is continuous, there always exists a set of distinct rewards values that numerically create a tail barrier in the risk-seeking policy.*

The proof is given in Appendix Section D.1. In practice, since we often use continuous reward functions (*e.g.*, NRMSE or Gaussian likelihood) and reward mappings, there is a risk of encountering the tail barrier. From (2), we can see that the tail barrier can incur *zero* policy gradient, since all the top $\alpha\%$ rewards are identical to R_α, leading to a zero weight for every gradient. As a consequence, the actor model would not have any effective updates according to the feedback from the selected expressions (top performers). In DSR, the RNN model will be updated only from an entropy bonus term (Petersen et al., 2019), and henceforth the learning starts to explore wildly.

To address this challenge, we introduce a ranking-based linear weighting scheme to construct a step-wise reward mapping,

$$f(z) = \lambda \cdot \mathrm{ReLU}\left(1 - \frac{|\{\tau^{(i)} : R(\tau^{(i)}) > R(z)\}|}{\alpha B/100}\right) \tag{11}$$

where z denotes an arbitrary expression sampled by the actor, $|\cdot|$ represents the size of the set, and $\lambda > 0$ is a scaling constant. This formulation ensures that expressions ranked within the top $\alpha\%$ receive positive mapped rewards, with higher-ranked expressions assigned larger values of $f(z)$, while all other expressions receive a reward of zero. Consequently, only top-performing expressions contribute to the gradient update, and lower-ranked ones are effectively ignored. The ranked-based weighting of our policy produces smoother updates, prevents pathological reward spikes, and improves training stability by removing tail barriers. Our risk-seeking policy gradient is therefore:

$$\nabla J_{\mathrm{risk}}(\theta; \alpha) \approx \frac{100}{\alpha B} \sum_{i=1}^{B} f(\tau^{(i)}) \nabla_\theta \log p(\tau^{(i)}|\theta). \tag{12}$$

Lemma 3.3. *By using the step function* (11) *for reward mapping, the policy gradient with our BIC reward, as shown in* (12)*, is unbiased and will not encounter any tail barrier.*

The proof is given in Appendix Section D.2.

Group Relative Policy Optimization. Inspired by the recent success in large language model reasoning, we adapt our policy to conceptually align with the Group Relative Policy Optimization (GRPO) framework (Shao et al., 2024) to further enhance the learning stability and efficiency. Equation 12 provides a group-relative advantage function which is integrated into a PPO- and TRPO-based policy (Schulman et al., 2017). Specifically, among the top $\alpha\%$ candidates, we only use the tokens from those expressions within a trust-region determined by the relative variation in trajectory probability as compared to the earlier model version:

$$\begin{aligned}\nabla \widehat{J}(\theta; \alpha) =& \tfrac{100}{\alpha B} \textstyle\sum_{i=1}^{B} \sum_{j=1}^{|\tau^{(i)}|} f(\tau^i) \cdot \\ & \nabla_\theta \min\Big[g(\tau^i, j), \mathrm{clip}\left(g(\tau^i, j), 1 \pm \epsilon\right)\Big], \\ g(\tau^i, j) =& \tfrac{p(\tau_j^{(i)}|\tau_{<j}^{(i)}, \theta)}{p(\tau_j^{(i)}|\tau_{<j}^{(i)}, \theta_{\mathrm{old}})},\end{aligned}$$

where θ_{old} denotes the model's parameters at the beginning of the current epoch, $\epsilon \in (0, 1)$ controls the size of the trust-region, and $|\tau^{(i)}|$ is the number of tokens in the expression. The trust region can prevent violent model updates caused by outlier samples, as the clipping operator sets $g(\tau^i, j) > 1 + \epsilon$ to a constant, thereby yielding a gradient of zero. To further improve the learning stability, we introduce a penalty term that penalizes model changes that are too aggressive. Our final policy update is:

$$\begin{aligned}\nabla J_{\mathrm{GRPO}}(\theta; \alpha) = \nabla \widehat{J}(\theta; \alpha) - \frac{100}{\alpha B} \sum\nolimits_{i=1}^{B} \sum\nolimits_{j=1}^{|\tau^{(i)}|} \\ \beta \cdot \nabla \mathrm{KL}[p(\tau_j^{(i)}|\tau_{<j}^{(i)}, \theta) \| p(\tau_j^{(i)}|\tau_{<j}^{(i)}, \theta_{\mathrm{ref}})],\end{aligned} \tag{13}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence, θ_{ref} denotes the parameters of the reference model, which is chosen from a previous epoch, and $\beta > 0$ controls the penalty strength. We use an experience replay buffer (Mnih et al., 2013) to merge the historically best performing $\alpha B/100$ expressions for each epoch. Our approach is summarized in Appendix Algorithm 4.

4 RELATED WORK

Since its invention, genetic programming (GP) has served as a dominant approach for symbolic regression (Koza, 1994). Recent representative works include Bingo (Randall et al., 2022), Operon (Burlacu et al., 2020), GP-GOMEA (Virgolin et al., 2021), and others. Bomarito et al. (2023) introduced a variant GP that leverages factional Bayes factor to indicate the most likely expression for noisy datasets.

The DSR framework (Petersen et al., 2019) uses reinforcement learning to train an RNN-based expres-

sion generator from a given measurement dataset. Recent extensions by Tenachi et al. (2023); Jiang et al. (2024) have incorporated domain knowledge into DSR by enforcing physics-unit constraints or generating rules within a vertical discovery space for vector symbolic regression. The DSR framework and our method can be regarded as *unsupervised* learning: only numerical measurements are provided, with no ground-truth expressions available. The goal, in fact, is to discover or generate those underlying expressions. DySymNet (Li et al., 2024) is another reinforcement learning method that trains an RNN controller to generate the structure of an EQL network (Kim et al., 2021). The generated EQL-network is fit to the training data with sparsity regularization applied to the network's weights. DySymNet's reward signal leverages the network's fit quality and complexity to select for compact, high-quality network structures. DySymNet hybridizes DSR-like search methods with EQL networks to improve accuracy and reduce complexity relative to EQL. DySymNet differs from CADSR through its application of reinforcement learning to improve EQL-network structure.

An alternative line of work aims to train a foundation model that can map numerical datasets directly to symbolic expressions (Biggio et al., 2021; Kamienny et al., 2022; Valipour et al., 2021; Vastl et al., 2022; Li et al., 2023). These approaches adopt a *supervised* learning paradigm, where the training data consist of input-output pairs: numerical measurements as inputs and their corresponding ground-truth symbolic expressions as outputs. The objective is to learn the underlying mapping between the two. These approaches typically rely on an encoder-decoder transformer architecture, where the encoder layers extract information from the numerical measurements, and the decoder layers integrate this information to generate expressions. While promising, such foundation models are expensive to train and highly sensitive to data preparation, which often involves fabricating large-scale synthetic datasets. More importantly, due to the absence of a data-specific search mechanism, these methods may struggle with generalization — particularly when applied to out-of-distribution data (Kamienny et al., 2023). To address this, recent research has explored coupling pretrained foundation models with data-specific search or planning mechanisms to improve expression discovery. Notable examples include TPSR (Shojaee et al., 2023), DGSR (Holt et al., 2023), and DGSR-MCTS (Kamienny et al., 2023). TPSR integrates a pretrained model within a Monte Carlo Tree Search (MCTS) (Browne et al., 2012) framework, using the model to guide token selection and search tree expansion via an upper confidence bound (UCB) heuristic. The MCTS reward function includes a complexity penalty, controlled by a tunable hyperparameter.

Other notable efforts include uDSR (Landajuela et al., 2022), an ensemble framework combining multiple symbolic regression methods such as GP and DSR, and model-free approaches (Sun et al., 2023; Xu et al., 2024) that rely solely on MCTS or hybrid strategies combining MCTS and GP for symbolic expression search.

5 EXPERIMENTS

For evaluation, we examined CADSR on the comprehensive and well-known SRBench dataset (La Cava et al., 2021), and then conducted an ablation study to assess the effectiveness of each individual component. Finally, we showcased the effectiveness of CADSR through an application to fracture mechanics.

5.1 Overall Performance

In SRBench, we first tested on the 133 problems with known solutions. We ran eight trials for each problem at the four *noise* levels: 0%, 0.1%, 1%, and 10%. We ran CADSR in a large computer cluster, for which we set a time limit of 4 hours for each trial. We deployed the trials on NVIDIA A40 GPUs. The maximum number of epochs is set to 600 without early termination. In each epoch, we sampled a batch of 1000 expressions to compute the policy gradient. We used ADAM optimization with an initial learning rate of 1E-4. The full list of hyperparameters of our method is provided in Appendix Table 2. We compared with 17 popular and/or state-of-the-art SR methods in terms of Symbolic Solution Rate (%), Accuracy Rate, and Simplified Complexity, which are standard metrics for SR evaluation. These SR baselines include DSR (Petersen et al., 2019), Bingo (Randall et al., 2022), GP-GOMEA (Virgolin et al., 2021), ITEA (de Franca and Aldeia, 2020), TPSR (Shao et al., 2024), BSR (Jin et al., 2020), AIFeynman (Udrescu and Tegmark, 2020), AFP_FE (Schmidt and Lipson, 2009), among others. The majority of these are genetic programming methods with a few notable exceptions: DSR is deep reinforcement learning, TPSR leverages a pretrained foundation model (Kamienny et al., 2022) to conduct Monte-Carlo tree search over expressions, BSR is an MCMC method with a prior placed on the tree structure, and AIFeynmen is a divide-and-conquer method that breaks the problem apart by hyper-planes and fits with polynomials. The results of the competing methods are retrieved from the public SRBench report (La Cava et al., 2021) and from published resources. For TPSR, we used the recommended setting of $\lambda = 0.1$, as suggested by the authors, which provides the best trade-off between expression accuracy and complexity. The DSR results reported in the SRBench report correspond to the variant with No Constant

Tokens (DSR-NCT), which may not fully capture the method's performance. To address this, we additionally evaluated an enhanced version of DSR by introducing constant tokens into the token library and optimizing them before each policy update, referred to as DSR-OCT (DSR with Optimized Constant Tokens). The results of all methods are presented in Fig. 2.

Overall, CADSR shows strong performance in symbolic discovery as measured by Symbolic Solute Rate. In particular, when data includes significant noise (10%), CADSR achieves the best solution rate, showing that our method is more *robust* to noise than all the competing methods. Meanwhile, the simplified complexity of our discovered expressions is among the lowest. This together shows that our method, with the BIC reward design, not only can find simpler and hence more interpretable expressions, but also is more resistant to data noise. The Accuracy Rate shows the ratio for which the method is able to discover an equation with $R^2 > 0.999$. CADSR has a competitive accuracy rate with AFP_FE, and Bingo, while outperforming DSR with and without constant optimization. The slightly better methods, such as TPSR and GP-GOMEA, however, generate lengthier and more complex expressions, which lack interpretability and are much far away from the ground truth expression. It is worth noting that CADSR outperforms DSR in both Symbolic Solution Rate (%) and Accuracy Rate, showing an improvement on both expression discovery and prediction accuracy. When data is noise-free, AIFeynman, uDSR, and Bingo shows better Symbolic Solution Rate than CADSR. This might be because: AIFeynman tends to use polynomials to construct the expressions, which match most of the ground-truth; Bingo as a genetic programming approach, uses evolution operators to sample new expressions, which might explore more broadly; uDSR is an ensemble approach using AIFeynman, genetic programming, and DSR and thus can achieve higher symbolic accuracy. However, in the presence of noise, the performance of these methods deteriorates largely and immediately falls behind CADSR, demonstrating CADSR's superior robustness to noisy data

Next, we tested with the 120 black-box problems in SRBench. These problems consists of real-world scientific and engineering problems spanning domains such as health informatics, technology, environmental science, and economics. Since the true solutions are *unknown*, we examined the Pareto front of all the methods in Model Size *vs.* R^2 Test Rank. As shown in Fig. 3, CADSR lies on the Pareto frontier, meaning CADSR is among the best in terms of the trade-off between the model size (expression complexity) and R^2 score (prediction accuracy). It is interesting to see that CADSR lies between DSR-NCT and DSR-OCT. On the contrary, methods like Operon and TPSR, typically generates way more complex expressions yet with smaller prediction error. It shows that our method can push the best trade-off toward more interpretability due to BIC penalizing complexity, which can be important in practice.

Runtime. In Appendix Figure 9, we provide a runtime comparison across all the methods. CADSR achieves an average runtime of approximately 4 hours over all problems, which is faster than DCT-OCT and slower than DCT-NCT. Note that DCT-NCT does not involve any constant optimization. On average, Bingo requires around 6 hours, while AIFeynman takes approximately 8 hours. Overall, these results confirm the training efficiency of CADSR.

5.2 Ablation Study

To evaluate the effectiveness of each component of CADSR, we ran ablations on the Feynman dataset from SRBench, which is composed of 119 problems. The numerical values of the metrics are given in Appendix C.

BIC Reward Function: To validate our BIC-based reward function, we first compared it against the standard NRMSE-based reward (see (1)) across varying noise levels. As shown in Figure 4, while the BIC reward performs slightly worse under zero noise, it substantially outperforms the NRMSE-based reward at all other noise levels. Notably, at a 10% noise level, the BIC reward improves upon the NRMSE-based reward by 111% in Symbolic Solution Rate.

In addition, we evaluated our BIC reward against two recently proposed alternatives: the SPL (Sun et al., 2023) and TPSR (Shojaee et al., 2023) reward functions, which both incorporate an explicit trade-off hyperparameter to balance expression complexity and data fit — denoted by η in SPL and λ in TPSR. Definitions of these rewards are provided in (14) and (15) in the Appendix. We performed the comparison at noise levels of 0% and 10%, testing a range of values for the trade-off hyperparameters. As shown in Figure 5a, our BIC reward — which requires no manually tuned trade-off parameter — consistently outperforms both alternatives across all settings. Specifically, the SPL reward achieves its highest Symbolic Solution Rate at $\eta = 0.99$ (for 0% noise) and $\eta = 0.8$ (for 10% noise), yet still falls short by 0.5% and 3%, respectively, compared to our BIC reward. The TPSR reward reaches its peak at $\lambda = 0.1$ for both noise levels, trailing our method by 5% and 20%. These results highlight the strength of our BIC-based approach, particularly in noisy settings, and confirm its effectiveness in balancing complexity and data fidelity without requiring parameter tuning

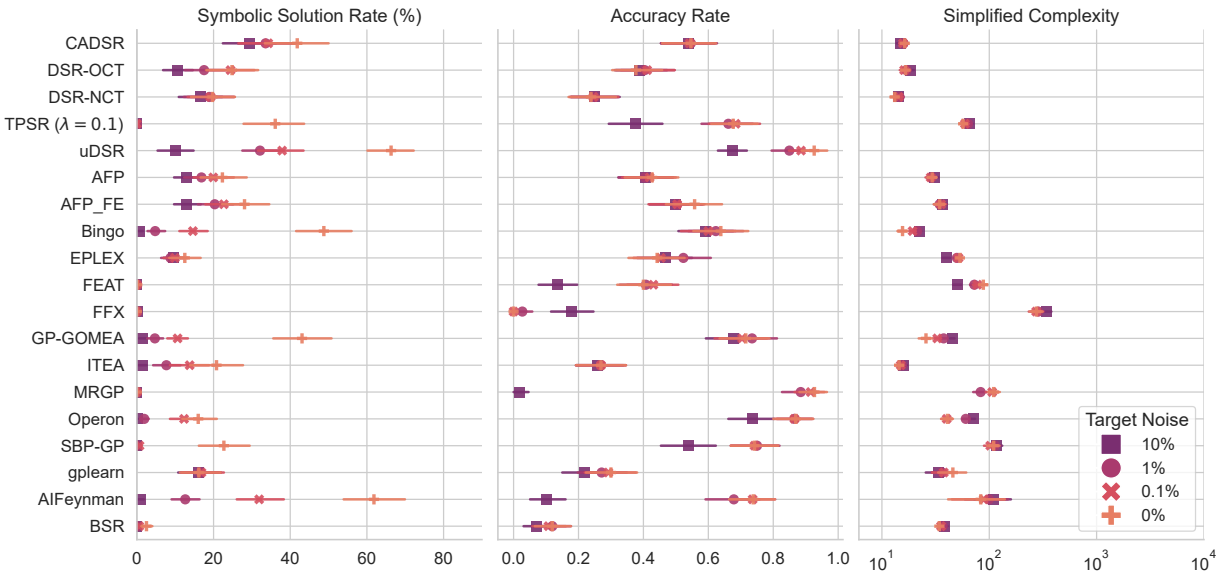


Figure 2: Symbolic regression performance on 133 SRBench problems with known solutions. Error bars denote a 95% confidence region. The numerical values are reported in Appendix Table 7.

Other Components: Next, we conducted an ablation study by individually removing other components from CADSR to evaluate their contributions to overall performance. Specifically, we replaced the DCT attention layer (see (7) and (8)) with a standard attention mechanism, substituted the dual-indexed position encoding (DPE) with a 1D positional encoding, and replaced GRPO with a standard risk-seeking policy gradient. The comparative results are presented in Figure 5b. As shown, removing each component generally led to a performance drop. The only exception was replacing DPE with 1D positional encoding, which resulted in a slight improvement of 0.5% in the noise-free setting; however, at the 10% noise level, performance declined by 3%. The DCT attention module not only improved the Symbolic Solution Rate compared to standard attention, but also reduced expression sampling time (see C.2 for runtime details). GRPO had a substantial impact: its removal decreased the symbolic discovery rate by 4.5%, and also slowed convergence — with the final expression discovered at an average of epoch 257, compared to epoch 168 with GRPO. Detailed convergence plots are provided in Appendix C.3.

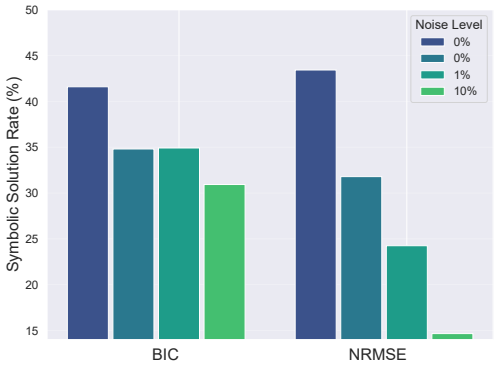


Figure 4: CADSR performance the standard NRMSE reward function.

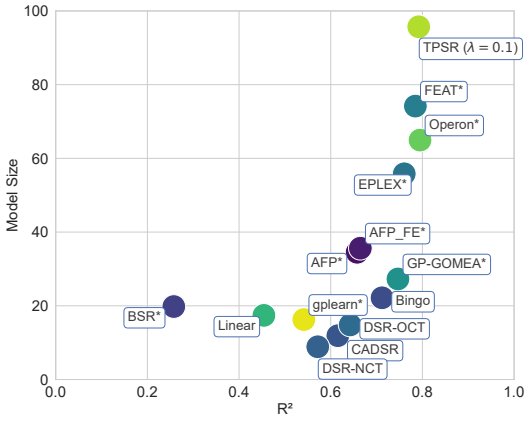
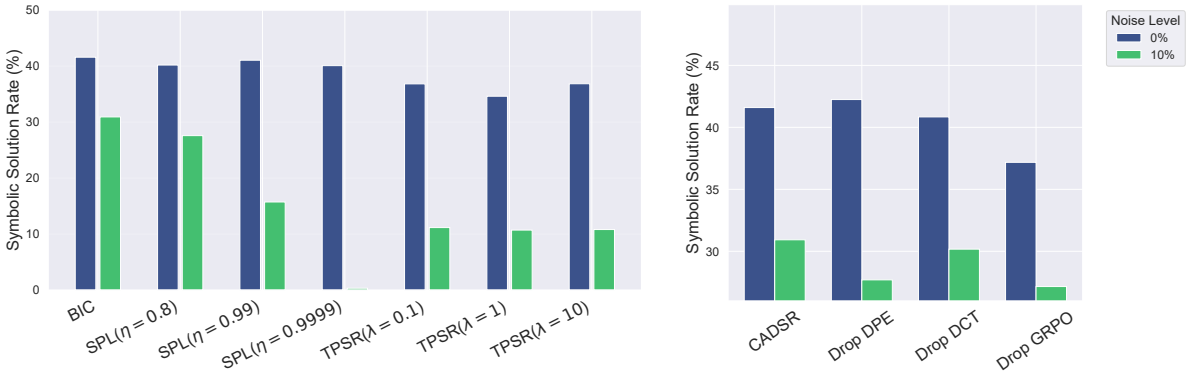


Figure 3: Pareto front of each method in 120 black-box problems of SRBench; the true solutions are unknown. Numerical metrics are reported in Appendix Table 9.

5.3 Fracture Mechanics

Finally, we applied CADSR to a critical problem in fracture mechanics — generating accurate and interpretable expressions for predicting crack initiation in microstructures (Spear et al., 2018). The dataset consists of six grain-level features used to predict the Fatigue Indicator Parameters (FIPs) (Hansen et al.,



(a) BIC *vs.* SPL *vs.* TPSR reward functions

(b) Individual component ablations.

Figure 5: Symbolic Solution Rate of CADSR with (a) alternative reward function and (b) individual component removes on Feynman dataset.

2024), which are known to be correlated with crack initiation. FIPs lack a closed-form solution and typically require computationally expensive simulations, so we aim to identify surrogate expressions that can quickly approximate FIPs (McDowell, 2010). We compare CADSR against DSR, GP-GOMEA, TPSR, and DySymNet, running eight trials per method with access to both CPU and GPU resources. The details about the dataset and the running of each method is provided in Appendix Section G.

The R^2 score and complexity of discovered expressions are reported in Table 1. In Appendix Table 12, we list the top expressions found by each method. As we can see, CADSR achieves the best trade-off between accuracy and interpretability. It consistently produces concise, interpretable expressions with strong predictive performance. In contrast, TPSR fails to generate accurate expressions, likely due to the numerical domain of the problem being outside its foundational model's training set. DSR achieves performance comparable to CADSR but yields slightly longer and less accurate expressions. GP-GOMEA provides marginal improvements in accuracy but at the cost of significantly higher complexity, producing expressions that were on average 50.75 tokens longer than CADSR and effectively uninterpretable. DySymNet produced significantly complex expression without improving the R^2 score. We suspect that DySymNet's performance is affected by its 8-hour runtime limit, as converging to its optimal expression may require additional computation. Overall, CADSR prioritizes simplicity and interpretability while maintaining accuracy, enabling engineers to better understand how cracks initiate.

Method	R^2	Complexity
CADSR	0.626	22.25
DSR	0.616	24.63
TPSR	0.046	40.75
GP-GOMEA	0.636	73.00
DySymNet	0.456	1697.88

Table 1: Symbolic regression performance for predicting crack initiation.

6 CONCLUSION

We have presented CADSR, a new symbolic regression approach based on reinforcement learning. On standard SR benchmark problems, CADSR shows promising performance. The ablation study confirms the effectiveness of each component of our method. The Fracture Mechanics problem enforces CADSR application for real world problems. Nonetheless, our current work has two limitations. First, the implementation is inefficient, especially for expression optimization, as each expression optimization is time intensive, causing a slow training process. A potential solution is to leverage a hybrid optimization approach, using a fast gradient-based optimizer (e.g., Adam, BFGS) initially, followed by refinement with LM. Thus, a hybrid approach could yield the same accuracy and stable constant optimization of LM while reducing total runtime. Second, we lack an early stopping mechanism to reduce the training cost further and prevent useless exploration. In the future, we plan to address these limitations.

7 Acknowledgements

ZB and JH acknowledge supports from Air Force Research Lab STTR Phase under project number 250504306. SZ acknolwedges support from NSF CAREER Award IIS-2046295, NSF OAC-2311685 (Elements: A Convergent Physics-based and Data-driven Computing Platform for Building Modeling), NSF DMS-2529112 (Collaborative Research: MATH-DT: Computationally efficient hypercomplex variable-based sensitivity methods for rapid Digital Twin model updating)

This work used Delta at the National Center for Supercomputing Applications through allocation CIS250317 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program Boerner et al. (2023), which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

References

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

Boerner, T. J., Deems, S., Furlani, T. R., Knuth, S. L., and Towns, J. (2023). ACCESS: Advancing innovation: NSF's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing (PEARC '23)*, page 4, Portland, OR, USA. ACM.

Bomarito, G., Leser, P., Strauss, N., Garbrecht, K., and Hochhalter, J. (2023). Automated learning of interpretable models with quantified uncertainty. *Computer Methods in Applied Mechanics and Engineering*, 403:115732.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Burlacu, B., Kronberger, G., and Kommenda, M. (2020). Operon C++: an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, pages 1562–1570, New York, NY, USA. Association for Computing Machinery.

de Franca, F. O. and Aldeia, G. S. I. (2020). Interaction-Transformation Evolutionary Algorithm for Symbolic Regression. *Evolutionary Computation*, pages 1–25.

Hansen, C. K., Whelan, G. F., and Hochhalter, J. D. (2024). Interpretable machine learning for microstructure-dependent models of fatigue indicator parameters. *International Journal of Fatigue*, 178:108019.

Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116. Publisher: World Scientific Publishing Co.

Holt, S., Qian, Z., and van der Schaar, M. (2023). Deep generative symbolic regression. In *The Eleventh International Conference on Learning Representations*.

Jiang, N., Nasim, M., and Xue, Y. (2024). Vertical symbolic regression via deep policy gradient. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5891–5899.

Jin, Y., Fu, W., Kang, J., Guo, J., and Guo, J. (2020). Bayesian symbolic regression.

Jobin, A., Ienca, M., and Vayena, E. (2019). The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399. Publisher: Nature Publishing Group.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. (2022). End-to-end Symbolic Regression with Transformers.

Kamienny, P.-A., Lample, G., Lamprier, S., and Virgolin, M. (2023). Deep generative symbolic regression with monte-carlo-tree-search. In *International Conference on Machine Learning*, pages 15655–15668. PMLR.

Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Ceperic, V., and Soljacic, M. (2021). 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.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

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

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. (2021). Contemporary Symbolic Regression Methods and their Relative Performance. arXiv:2107.14351 [cs].

Landajuela, M., Lee, C., Yang, J., Glatt, R., Santiago, C. P., Aravena, I., Mundhenk, T. N., Mulcahy, G., and Petersen, B. K. (2022). A unified framework for

deep symbolic regression. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168.

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

Li, W., Li, W., Yu, L., Wu, M., Sun, L., Liu, J., Li, Y., Wei, S., Deng, Y., and Hao, M. (2024). A neural-guided dynamic symbolic network for exploring mathematical expressions from data.

McDowell, D. L. (2010). A perspective on trends in multiscale plasticity. *International Journal of Plasticity*, 26(9):1280–1309. Special Issue In Honor of David L. McDowell.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. (2019). Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients.

Randall, D. L., Townsend, T. S., Hochhalter, J. D., and Bomarito, G. F. (2022). Bingo: a customizable framework for symbolic regression with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, pages 2282–2288, New York, NY, USA. Association for Computing Machinery.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. arXiv:1811.10154 [cs, stat].

Schmidt, M. and Lipson, H. (2009). Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85. Publisher: American Association for the Advancement of Science.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Schwarz, G. (1978). Estimating the dimension of a model. *The annals of statistics*, pages 461–464.

Scribano, C., Franchini, G., Prato, M., and Bertogna, M. (2023). Dct-former: Efficient self-attention with discrete cosine transform. *Journal of Scientific Computing*, 94(3).

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. (2024). Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

Shojaee, P., Meidani, K., Farimani, A. B., and Reddy, C. K. (2023). Transformer-based Planning for Symbolic Regression.

Spear, A. D., Kalidindi, S. R., Meredig, B., Kontsos, A., and le Graverend, J.-B. (2018). Data-driven materials investigations: The next frontier in understanding and predicting fatigue behavior. *JOM*, 70(7):1143–1146.

Sun, F., Liu, Y., Wang, J.-X., and Sun, H. (2023). Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *The Eleventh International Conference on Learning Representations*.

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

Tamar, A., Glassner, Y., and Mannor, S. (2014). Policy Gradients Beyond Expectations: Conditional Value-at-Risk.

Tenachi, W., Ibata, R., and Diakogiannis, F. I. (2023). Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *The Astrophysical Journal*, 959(2):99.

Udrescu, S.-M. and Tegmark, M. (2020). AI Feynman: a Physics-Inspired Method for Symbolic Regression. arXiv:1905.11481 [hep-th, physics:physics].

Valipour, M., You, B., Panju, M., and Ghodsi, A. (2021). SymbolicGPT: A Generative Transformer Model for Symbolic Regression. arXiv:2106.14131 [cs].

Vastl, M., Kulhánek, J., Kubalík, J., Derner, E., and Babuvska, R. (2022). SymFormer: End-to-end symbolic regression using transformer-based architecture. arXiv:2205.15764 [cs].

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. N. (2021). Improving Model-based Genetic Programming for Symbolic Regression of Small Expressions. *Evolutionary Computation*, 29(2):211–237. arXiv:1904.02050 [cs].

Wasserman, L. (2000). Bayesian model selection and model averaging. *Journal of mathematical psychology*, 44(1):92–107.

Xu, Y., Liu, Y., and Sun, H. (2024). Reinforcement symbolic regression machine. In *The Twelfth International Conference on Learning Representations*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendix

A Algorithms

Below, we show the algorithms for expression sampling and positional encoding generation. Note that for expression sampling, we over-sample expressions so that we can return a high number of unique ones. If not enough unique expressions exist, then we begin to allow duplicates to fill out our batch size requirements.

Algorithm 1 Expression Tree Sampling

input Number of expressions to sample B; oversampling scalar $\gamma > 1$, maximum tree-node number ν
output A set of expressions $\mathcal{T}$

$\mathcal{T} \leftarrow$ ExpressionTrees(γB) {Creates γB empty expression trees}
while $i < \nu$ **do**
  $V_{\mathcal{T}} \leftarrow$ Inputs($\mathcal{T}$) {Fetching the input embeddings of all the expression trees}
  $S \leftarrow p(V_{\mathcal{T}}|\theta)$ {Predicting categorical distributions from the transformer}
  $S \leftarrow R(S)$ {Applying rules to each distribution}
  $K \leftarrow P(\cdot|S)$ {Sampling from the categorical distribution to obtain tokens}
  $\mathcal{T}_i \leftarrow K$ {Adding the new tokens into the expression trees}
end while
$\mathcal{T} =$ Unique($\mathcal{T}, B$) {Take the first B Unique expression trees}
return $\mathcal{T}$

Algorithm 2 Dual Indexed Position Encoding (DPE) Generation

input An expression tree τ

τ.root_node.depth = 1
τ.root_node.horizontal = 1/2
PositionEncodingInformation(τ.root_node)

Algorithm 3 PositionEncodingInformation

input Current node

if node has left **then**
  node.left.depth = node.depth + 1
  node.left.horizontal = node.horizontal - $1/(2^{\text{node.left.depth}})$
  PositionEncodingInformation(node.left)
end if
if node has right **then**
  node.right.depth = node.depth + 1
  node.right.horizontal = node.horizontal + $1/(2^{\text{node.right.depth}})$
  PositionEncodingInformation(node.right)
end if

B Model Details

Table 2 and Fig. 6 show the comprehensive hyperparameter settings and the architecture of the transformer used in CADSR. Note that we use the Levenberg–Marquardt algorithm (Levenberg, 1944) to optimize the constant tokens for each discovered equation and used Adam (Kingma and Ba, 2017) to optimize the model.

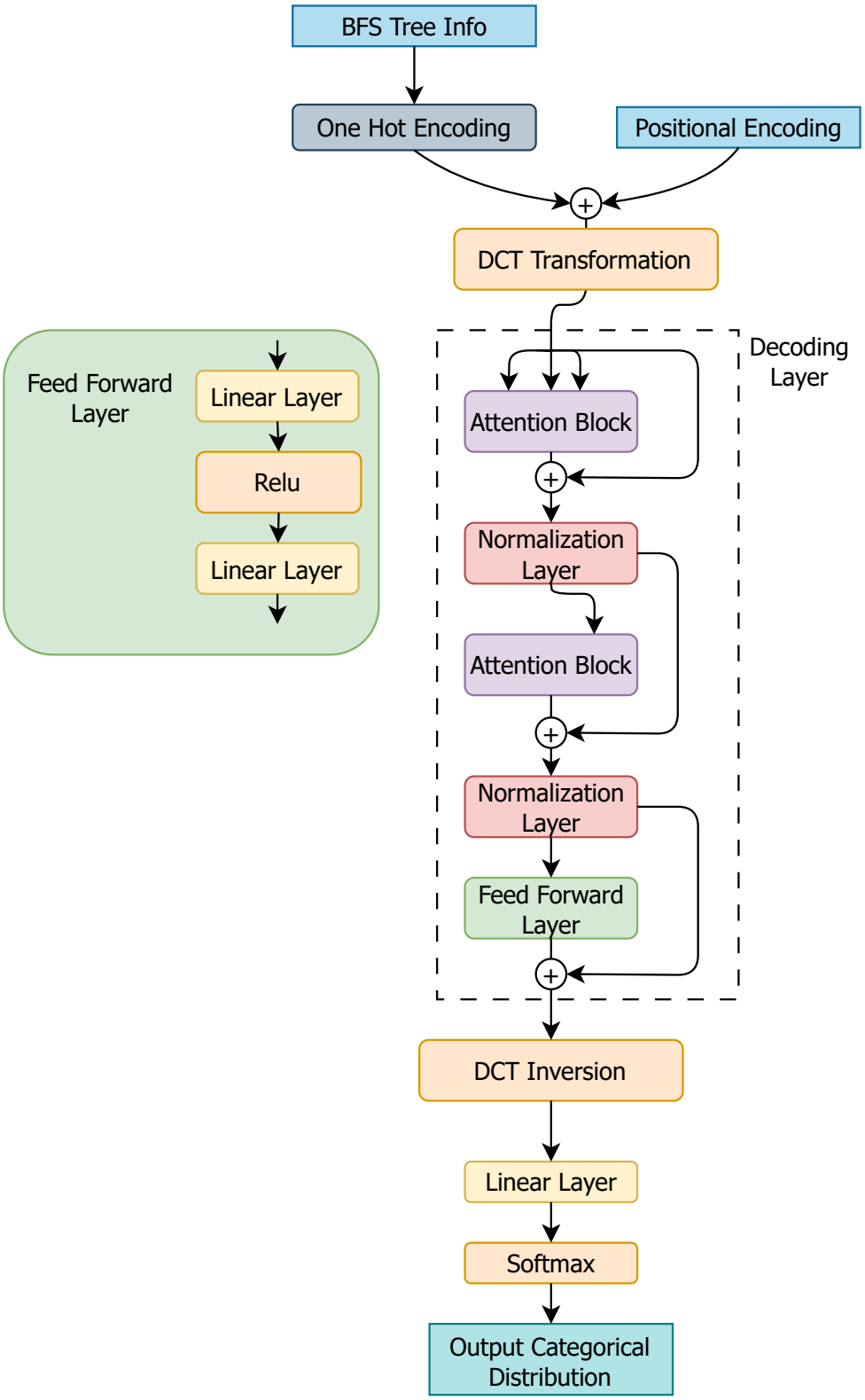


Figure 6: The architecture of the transformer actor in CADSR.

Algorithm 4 Complexity-Aware Deep Symbolic Regression (CADSR)

input Learning rate l; risk factor α; batch size B; coefficients $\lambda > 0$; steps per epoch C; epochs per reference G; number of epochs T
output The best equation τ^*

Initialize transformer with parameters θ
while $i < T$ **do**
  if $i \bmod(G) = 0$ **then**
    $\theta_{\text{ref}} \leftarrow \theta$ {Set the reference weights to the current weights}
  end if
  $\theta_{\text{old}} \leftarrow \theta$ {Set the old weights to the current weights}
  $\mathcal{T} \leftarrow \{\text{OptimizeConstants}(\tau^{(i)}) : \tau^{(i)} \sim p(\cdot|\theta)\}_{i=1}^{B}$ {Sample B expressions from the transformer actor and optimize the values of constant tokens}
  $\mathcal{R} \leftarrow \{\text{BIC}(\tau^{(i)})\}_{i=1}^{B}$ {Calculate the reward for each expression using (10)}
  $\mathcal{R}_\alpha \leftarrow (1 - \alpha/100)$-quantile of $\mathcal{R}$
  $\mathcal{T} \leftarrow \{\tau^{(i)} : \mathcal{R}(\tau^{(i)} \geq \mathcal{R}_\alpha\}$ {Pick top α% expressions}
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_{\text{historical}}$ {Merge with the historical top performing expressions}
  for $j = 1$ to C **do**
    $\theta \leftarrow \theta + l \cdot (\nabla_\theta J_{\text{GRPO}} + \lambda_{\mathcal{H}} \cdot \text{Entropy-Bonus})$ {Compute the policy gradient using (13); the entropy bonus term comes from the original DSR.}
    if $\max \mathcal{R} > \mathcal{R}(\tau^*)$ **then** $\tau^* \leftarrow \tau^{(\text{argmax } \mathcal{R} \in \mathcal{T})}$ {Update the best equation}
  end for
  $\mathcal{T}_{\text{historical}} \leftarrow$ Top α% of the expressions in $\mathcal{T}$ {Update the experience replay buffer}
  $i \leftarrow i + 1$
end while
return τ^*

Table 2: Hyperparameter settings of CADSR.

Hyperparameter	SRBench	Fracture Mechanics
Variables	{1, c (Constant Token), x_i}	{1, c (Constant Token), x_i}
Unary Functions	{sin, cos, log, $\sqrt{(\cdot)}$, exp}	{sin, cos, log, $\sqrt{(\cdot)}$, exp, sqrt, tan, square}
Binary Functions	{+, -, *, /, ^}	{+, -, *, /, ^}
Batch Size	1000	2000
Risk Seeking Percent (α)	5%	5%
Learning Rate	1E-4	1E-4
Max Depth	32	32
Oversampling	2	3
Number of Epochs	600	500
Policy	BIC	BIC
λ	0.2	0.2
Entropy Coefficient $\lambda_{\mathcal{H}}$	0.005	0.0005
Encoder Number	0	0
Decoder Number	1	2
Number of Heads	1	3
Feed Forward Layers Size	2048	2048
β	0.01	0.1
ϵ	0.2	0.2
Embedding Dim	10	15
DCT Clip Dim	8	12
C	5	5
G	5	5
CPU Count	1	10

B.1 Sensitivity Analysis

We conducted a compact sensitivity analysis of the key parameters introduced by CADSR: DCT frequency cutoff M, trust region ϵ, and KL Divergence regularizer β. We ran three trials for each setting on the Strogatz dataset

Table 3: Hyperparameter settings for DSR

Hyperparameter	*DSR*
Batch Size	1000
Learning Rate	0.0005
Entropy coefficient	0.005
Risk Factor Percent	5%
RNN Type	LSTM
Layer Number	1

Table 4: Sensitivity analysis for new parameters introduced by CADSR

	DCT Cutoff (ω)				**Trust Region (δ)**				**KL Regularizer (λ)**			
	4	6	**8**	10	0.01	**0.2**	0.5	1.0	0.001	0.01	**0.1**	1.0
Acc. (%)	78.6	78.6	**78.6**	71.4	75.0	**78.6**	64.3	71.4	71.4	71.4	**78.6**	71.4
Sol. Rate (%)	71.4	61.9	**71.4**	50.0	63.1	**71.4**	59.5	61.9	64.3	69.0	**71.4**	61.9
Complexity	14.79	13.29	**13.36**	14.00	14.57	**13.36**	14.00	14.21	13.86	13.36	**13.36**	12.64

with 0% noise. Table 4 presents the results of this analysis with bolded columns referencing the hyperparameter setting used for the SRBench experiments.

For the DCT frequency cutoff, we found that removing high-frequency components is beneficial. Not clipping any frequencies led to a 21.4% drop in solution rate. Interestingly, the model appears to depend strongly on the low-frequency domain: discarding the highest 60% of frequencies ($M = 4$) produced only a modest performance change and increased the expression complexity by just 1.43 tokens compared with our default ($M = 8$), which suggests that DCT truncation acts primarily as a model regularizing mechanism.

The trust-region sweep indicates that the method relies on a sufficiently tight region to prevent unstable or erroneous updates. Increasing ϵ results in notable performance degradation across all metrics, while overly small ϵ values also reduce performance — likely because an excessively restrictive trust region slows or hinders policy improvement.

The KL penalty coefficient β plays an influential role as Table 4 highlights the degradation in performance when altering β. Smaller values of β decreased both accuracy and solution rate, presumably due to insufficient regularization and overly aggressive updates. Conversely, larger values of β led to reductions in accuracy and symbolic recovery, consistent with over-regularization.

C Additional Ablation Analysis

Table 5: Performance of CADSR for each ablation on the Feynman dataset

	Symbolic Solution Rate (%)	
Algorithm	0.0%	10%
CADSR	41.59	**30.93**
Drop DPE	42.24	27.69
Drop BIC	**43.43**	14.65
Drop DCT	40.84	30.17
Drop GPRO	37.18	27.16
SPL ($\eta = 0.8$)	40.19	27.59
SPL ($\eta = 0.99$)	41.06	15.73
SPL ($\eta = 0.9999$)	40.09	0.32
TPSR ($\lambda = 0.1$)	36.83	11.16
TPSR ($\lambda = 1$)	34.62	10.71
TPSR ($\lambda = 10$)	36.85	10.81

C.1 Reward Functions

SPL and TPSR introduced reward functions with regularization terms and tunable hyperparameters. SPL's reward function is specified in (14), and TPSR's reward function is (15). SPL introduced a scalar term η^n, where n denotes the number of multiplication operators in the expression, and η is a hyperparameter. η controls the strength of the regularizer where $\eta = 1$ means no regularization and $\eta < 1$ strengthens the regularazation. The authors of the SPL work found that tuning η significantly impacts the performance, and they suggest a starting value of $\eta = 0.99$. TPSR introduced an additive term $\lambda \exp(\frac{-l(\tau)}{L})$, where L is the max number of tokens, $l(\tau)$ is the length of the expression, and λ is a hyperparameter. λ controls the regularization, and having a large λ will increase the regularization.

$$R(\tau) = \frac{\eta^n}{1 + \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \tau(\mathbf{x}_i))^2}} \tag{14}$$

$$R(\tau) = \frac{1}{1 + \mathrm{NMSE}(y, \tau(\mathbf{x}))} + \lambda \exp(\frac{-l(\tau)}{L}) \tag{15}$$

C.2 DCT Attention

The DCT converts the input into the signal space before using the attention mechanism, which maintains the transformer's time complexity of $\mathcal{O}(n^2 d)$, where n is the sequence length and d is the embedding dimension. We reduce the embedding space by clipping the highest frequency components. For SRBench, we clipped the 2 highest frequency signals from a 10-dimensional embedding space. Therefore, we should attain around a 20% decrease in sampling time of the standard transformer. Since the runtime for the whole SRBench is dominated by the optimization of the constant token(s), to exclusively evaluate the sampling efficiency, we tested with the `Nguyen-4` problem, which does not include the constant token in the token library. We ran our method with DCT attention layers and with ordinary attention layers, each for 50 trials with 100 epochs per trial on an RTX 3080 GPU. We recorded the time each architecture took to predict tokens. The empirical results, as shown in Table 6, match the theoretical runtime reduction, resulting in a 21% decrease in sample time using the DCT attention layers.

Table 6: Sampling Time on `Nguyen-4` with 50 trials.

Architecture	Total Sample Time (s)	Sample Time per Expression (ms)
DCT attention	1072.8	3.576
Standard attention	1354.2	4.514

C.3 GRPO

We observed that GRPO can increase performance and convergence rate compared to the standard risk-seeking gradient. Figure 7 includes 20 random problems from Feynman dataset in SRBench. We can see in all the cases, GRPO enables the model to converge faster than the standard risk-seeking policy gradient. In the vast majority of the case, GRPO leads to a higher reward. On average, GRPO improves the symbolic recovery rate across the entire dataset.

C.4 Dual-Position Encoding Info

In the DPE ablations, we observed that replacing DPE with a standard 1D position encoding results in a marginal decrease in symbolic recovery peformance at 0.0% noise, while improving symbolic recovery at 10% noise. We conject that this effect is realated to the improvement in exploration that the DPE introduces. Empirically, we found that using DPE consistently increases the diversity of expressions generated by the model compared with standard 1D positional encoding. We re-evaluated four randomly picked SRBench problems at the 10% noise level with each positional encoding and found that DPE discovered an average of 473 novel expressions per epoch, whereas standard positional encoding discovered 284.

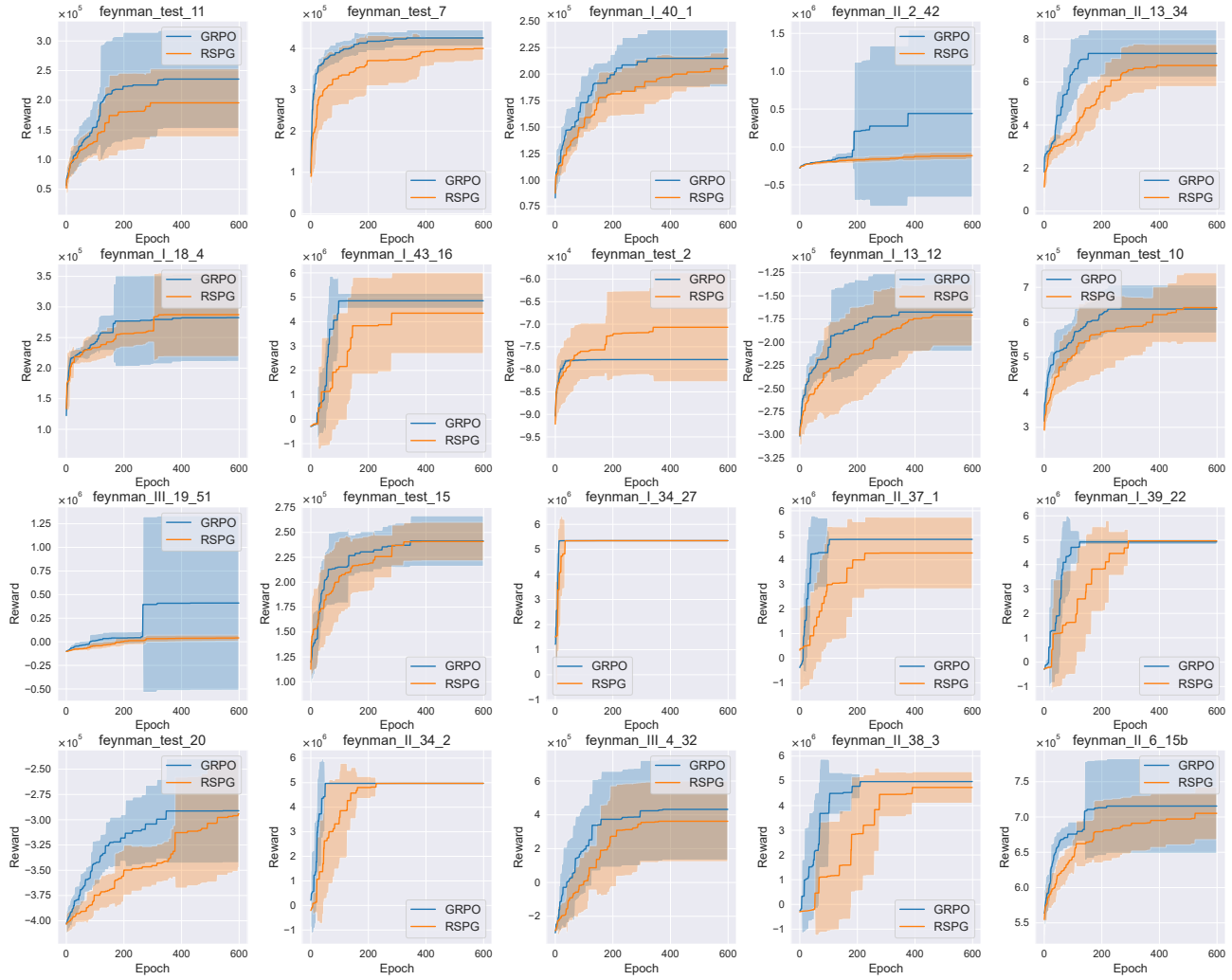


Figure 7: Twenty examples of learning curves (the number of epochs *vs.* reward) for running CADSR with GRPO and with standard risk seeking policy gradient (RSPG). The problems were randomly selected from the Feynman dataset. Errors are one standard deviation.

This increased exploration may function as a form of regularization: by reducing over-exploitation, it helps prevent the model from collapsing into poor modes — such as overly complex expressions that overfit noise — thereby improving robustness in noisy settings. Conversely, in the noise-free setting, this additional exploration may slightly hinder performance because the model may spend more effort exploring alternatives rather than converging quickly to the (easily discoverable) correct expression.

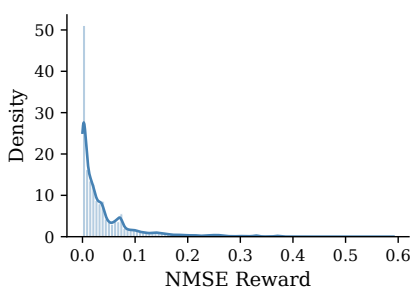
We suspect that DPE provides richer positional information — capturing both the depth and the horizontal position of tokens within the expression tree — which can lead to more expressive latent representations that enhance exploration. This, in turn, likely enables the policy to generate a broader set of candidate expressions. Evaluation of these statements poses challenges, as we can only observe correlations in the results, causing these statements to remain conjectures.

C.5 Tail Barrier

Empirically, we observed "tail barrier" effect occur fairly often as the reward distribution is often highly skewed, wherein the top k expressions within the top-α% receive a substantially larger reward than the rest. These top-k expressions dominate the policy update due to the difference in reward, resulting in a "tail barrier" effect. Importantly, these top-k expressions are not always representative or a derivative of the underlying expressions and can lead to a local optimum.

To observe the phenomenon, we evaluated five randomly selected problems from the Feynman dataset with five trials per problem. For each problem, we ran CADSR with the standard NMSE reward and recorded the reward for each expression in the top-α% at every training step. Across all epochs, we observed that the bottom 10% of all expression rewards was below 10^{-3}, whereas the top 10% of expressions were above 0.12 - a two orders of magnitude difference. Figure 8a highlights how significantly the reward is right-skewed.

We observed that 0.7% of all epochs had a full-tail barrier, resulting in all rewards for that epoch being zero. Additionally, we found that k expressions dominated an epoch if the top-k expressions contained over 80% of the total reward for the epoch. Table 8b shows that there is a significant probability that multiple epochs are dominated by a small set of expressions, leading to the effects of a tail barrier being a probable occurrence during training.



(a) NMSE reward distribution during training

Top-k	Percent Dominated
0	0.74%
1	1.51%
3	2.39%
5	3.16%
10	6.29 %

(b) Percent of epochs dominated by k expressions

Figure 8: Overall caption for both panels

D Theoretical Results

D.1 Proof of Lemma 3.2

Proof. Pick up any reward value z_0. Due to the continuity of the mapping f, for arbitrary $\epsilon > 0$, there exists $\delta > 0$ such that for all $z \neq z_0$, if $|z - z_0| < \delta$, then $|f(z) - f(z_0)| < \epsilon$. Let us take $\epsilon = \frac{1}{2}s$, where s is the machine precision (*e.g.*, 2^{-32}). Since the reward function is continuous, we can find a set of distinct reward values $z_1, \ldots, z_M$ from $B(z_0, \delta(\epsilon)) = \{z \in \text{dom } f, |z - z_0| < \delta(\epsilon)\}$. Let us look at the mapped rewards, $f(z_1), \ldots, f(z_M)$. For any $1 \leq i, j \leq M$, we have

$$|f(z_i) - f(z_j)| = |f(z_i) - f(z_0) + f(z_0) - f(z_j)| \leq |f(z_i) - f(z_0)| + |f(z_0) - f(z_j)| < \frac{s}{2} + \frac{s}{2} = s.$$

Therefore, there are no numerical difference among these mapped rewards, and they can create a tail barrier. □

D.2 Proof of Lemma 3.3

Proof. For any set of BIC reward values, $\mathcal{S} = \{R(\tau^{(1)}), \ldots, R(\tau^{(B)})\}$, we denote the mapped reward values by $\widehat{\mathcal{S}} = \{\widehat{R}_1, \ldots, \widehat{R}_B\}$, where each $\widehat{R}_j = f(R(\tau^{(j)}))$ $(1 \leq j \leq B)$. We know that each

$$\widehat{R}_j = \lambda \cdot \text{ReLU}\left(1 - \frac{\{\tau^{(i)} : R(\tau^{(i)}) > R(\tau^{(j)}), 1 \leq i \leq B\}|}{B\alpha/100}\right) \tag{16}$$

where R_α is the $1 - \frac{\alpha}{100}$ quantile of the rewards in $\mathcal{S}$. Let B_α denotes the number of top $\alpha\%$ expressions. We have $B\alpha/100 = B_\alpha$, and

$$\widehat{R}_j = \lambda \cdot \text{ReLU}\left(1 - \frac{\{\tau^{(i)} : R(\tau^{(i)}) > R(\tau^{(j)}), 1 \leq i \leq B\}|}{B_\alpha}\right).$$

Since for every top $\alpha\%$ expression $\tau^{(j)}$, the numerator $|\{\tau^{(i)} : R(\tau^{(i)}) > R(\tau^{(j)}), 1 \leq i \leq B\}|$ is an integer and is guaranteed to be less than B_α, $\widehat{R}_j$ is always positive. Accordingly, every expression in the top $\alpha\%$ will not have their gradient zeroed out (see (12)), and we will never meet a tail or partial tail barrier.

To show the unbiasedness, we need to replace R_α by the $1 - \alpha/100$ quantile of the distribution of the BIC reward R. Denote the mapped reward by $\widehat{R}$. Since $\widehat{R}$ is bounded, namely, $\widehat{R} \in [0, 1]$, we can follow exactly the same steps in the proof of the original risk-adverse gradient paper (Tamar et al., 2014) to show the unbiasedness of (12). □

E SRBench

E.1 Symbolic Problems

We provide Table 7 containing the numerical values for all of the algorithms for all of the algorithms we compare within Figure 2. Bolded values are the best for the given column. CADSR has achieved the best performance for 0.1%, 1%, and 10% noise, generating the second most simple models on average for these noise levels. MRGP has the best performance in terms of accuracy but generates overly complex expressions with high accuracy. Note that uDSR is omitted because its exact numerical data is not available. SRBench uses SymPy[1] to evaluate symbolic accuracy. However, SymPy is known for generating false negatives due to incomplete simplifications. For our table, we also counted any expression as being symbolically correct when the R^2 test score is exactly 1.0, as it is improbable that an expression that is not symbolically equivalent will perfectly predict at $\sim 20,000$ testing points. Additionally, we compare with DySymNet in Table 8 for SRBench at 0.0% noise based on the ability categories reported in DySymNet's original paper.

E.2 Black Box Problems

Table 9 shows the numerical performance metrics and runtimes of each method tested on the black box SRBench dataset. While CADSR does not give the best metric in any category, it does appear on the Pareto front in Figure 3. Notably, SBP-GP, TPSR, and Operon are the best-performing algorithms in terms of R^2 test score on the black box problems and have a higher level of complexity than CADSR. SBP-GP and Operon are GP-based methods. TPSR is a transformer that uses MCTS on a pretrained transfomer to predict lengthy and accurate expressions.

F Run Times

Figure 9 shows the recorded runtime of each method in solving the symbolic problems from SRBench. However, this is a rough comparison, since CADSR does not use any early stopping criterion and the hardware resources — though comparable — are not exactly the same across all the methods for this benchmark.

[1] `https://www.sympy.org/en/index.html`

Table 7: Symbolic regression performance in SRBench with known solutions.

	Symbolic Solution Rate (%)				Accuracy Rate (%)				Simplified Complexity			
Algorithm	0.0%	0.1%	1%	10%	0.0%	0.1%	1%	10%	0.0%	0.1%	1%	10%
AFP	22.31	19.92	16.85	12.85	43	42	40	41	29.35	28.43	29.01	30.82
AFP_FE	28.08	22.69	20.31	12.85	56	50	50	50	34.58	35.66	34.45	36.77
AIFeynman	**61.84**	31.89	12.61	0.86	74	74	68	10	83.29	88.66	99.27	110.54
BSR	2.50	0.61	0.08	0.00	12	11	12	7	34.29	35.51	36.80	38.38
Bingo	48.77	14.62	4.77	0.77	64	60	62	59	15.56	19.29	21.32	22.54
CADSR	41.83	**34.27**	**33.57**	**29.25**	55	54	55	54	16.30	15.69	15.89	15.03
DSR-OCT	24.81	24.42	17.53	10.48	38	41	40	39	16.57	16.10	16.96	18.45
DSR-NCT	19.71	19.23	18.92	16.61	24	25	25	25	**13.14**	**14.36**	**14.61**	**14.40**
EPLEX	12.50	9.92	8.77	9.54	44	45	52	47	53.24	51.74	49.91	40.04
FEAT	0.10	0.00	0.00	0.00	40	43	41	14	88.01	77.32	72.61	50.40
FFX	0.00	0.00	0.00	0.08	0	0	3	18	274.88	273.29	286.03	341.38
GP-GOMEA	43.08	10.62	4.69	1.46	71	70	73	**68**	25.73	32.75	37.59	45.41
ITEA	20.77	13.77	7.69	1.46	27	27	27	26	14.46	14.96	15.35	16.00
MRGP	0.00	0.00	0.00	0.00	**93**	**92**	**89**	2	109.95	106.50	83.06	0.00
Operon	16.00	12.31	1.92	0.08	87	86	86	73	40.80	40.13	60.40	70.78
SBP-GP	22.69	0.69	0.00	0.00	74	74	75	54	109.94	102.09	112.93	116.30
TPSR	36.09	0.00	0.00	0.00	68	68	66	38	57.11	59.32	63.42	65.83
gplearn	16.15	16.86	16.59	16.00	30	29	27	22	45.80	37.76	36.42	33.84

Table 8: R^2 scores on SRBench with 0.0% noise.

Dataset	CADSR	DySymNet	GP-Gomea	Operon	TPSR	AIFeynman
Feynman	0.9363	0.9931	**0.9969**	0.9908	0.9925	0.9045
Strogatz	0.9689	0.9968	0.9975	**0.9994**	0.9648	0.5900

Table 9: Symbolic regression performance and runtime on black box problems in SRBench.

Algorithm	**R^2 Test**	**Model Size**	**Training Time (s)**
AFP	0.657613	34.5	6002.55
AFP_FE	0.664599	35.6	6153.50
AIFeynman	-3.745132	2500	82069.31
AdaBoost	0.704752	10000	65.29
BSR	0.257598	19.8	20426.86
Bingo	0.711951	22.2	53537.42
CADSR	0.615459	11.9	5643.43
DSR-NCT	0.571669	**8.89**	35096.16
DSR-OCT	0.642417	14.8	4267.68
EPLEX	0.760414	55.8	15673.69
FEAT	0.784662	74.2	6723.91
FFX	-0.667716	1570	243.74
GP-GOMEA	0.746634	27.3	9063.54
ITEA	0.640731	112	12337.48
KernelRidge	0.615147	1820	38.88
LGBM	0.637670	5500	28.53
Linear	0.454174	17.4	**0.24**
MLP	0.531249	3880	30.47
MRGP	0.417864	12100	190697.78
Operon	0.794831	65.0	2979.43
RandomForest	0.698541	1.54e+06	120.21
SBP-GP	**0.798932**	639	166745.46
TPSR	0.792001	95.7	504.38
XGB	0.775793	16400	240.58
gplearn	0.541264	16.3	24157.99

F.1 Expressions Evaluated

Evaluated expressions are an additional metric to compare symbolic regression methods. Table 10 compares the average number of expressions evaluated by CADSR and DSR across 5 trials of the Stragatz dataset. In these trials CADSR had a 4-hour runtime limit, and DSR had an 8-hour runtime limit, and CADSR had an evaluation budget of 600k while DSR had an evaluation budget of 500k. CADSR achieved 40% reduction in evaluations

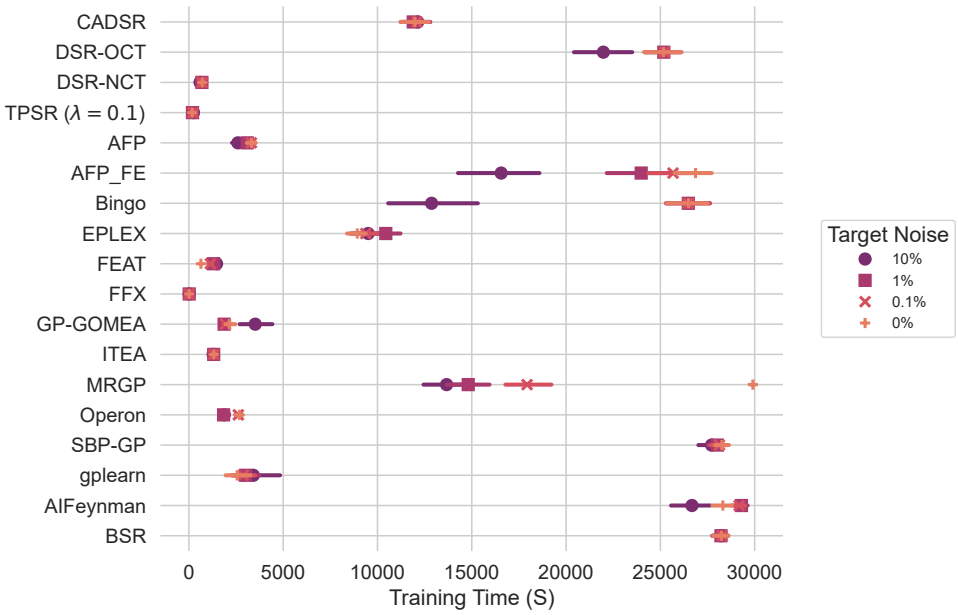


Figure 9: Runtime in solving SRBench problems with solutions known. Error bars denote a 95% confidence interval.

compared to DSR on noisy datasets while maintaining higher accuracy and a higher symbolic recovery rate. These results suggest that CADSR is more sample-efficient in identifying notable expressions. In the majority of the experiments, DSR exhausts its evaluation budget, while CADSR terminates earlier due to runtime limitations. Notably, CADSR had a higher evaluation budget of 600k, but rarely surpassed 450k evaluations during these trials.

Table 10: Evaluated expressions for DSR and CADSR on the Strogatz dataset

Noise Level	DSR	CADSR	Reduction (%)
0.0%	338k	331k	2.1
0.1%	500k	287k	42.6
1.0%	500k	272k	45.6
10.0%	500k	299k	40.2

G Fracture Mechanics

The dataset contains the following 6 features: Max Schmid of Grain, Average Schmid Factor of Grain, Number of High Grain Schmids, Principal Components Grain, Schmid Variance in Grain, sin NST. Complexity of each expression was calcualate by counting the number of nodes in the expression tree using SymPy.

Table 11 comparing average accuracy, runtime, and complexity of the four methods. Each method was given access to up 10 CPU cores and a GPU. TPSR was ran with the default parameter settings for the model on a H200 GPU, as the model required over 20 GB of VRAM. DSR and GP-GOMEA used the same parameters used for the SRBench benchmark. DySymNet was given an 8 hour runtime with the same GPU and CPU resources as CADSR. Treating this as a real-world problem, we ran CADSR without the hardware limitations used for SRBench, enabling potential improvements in runtime and performance. A list of hyperparameters for CADSR can be found in Table 2. Table 12 reports the best overall expression found by each method. CADSR achieves similar accuracy as GP-GOMEA, while providing the most human interpretable expression. DySymNet is excluded from Table 12 due to its high complexity making it unsuitable for print.

To ensure a fair comparison, we tested DSR with a batch size of 2000, as to match the batch size of CADSR. We labeled this experiment DSR (2000) in Table 11. The larger batch size caused all eight DSR trials to converge early, leading to identical expression discovered in across all trials. While this version of DSR discovers an

expression that is less complex than CADSR's expression, it significantly impacts R^2 score.

Table 11: Symbolic regression performance for predicting crack initiation.

Method	Train R^2	Test R^2	Runtime (s)	Complexity
CADSR	0.636	0.626	11583.643	22.250
DSR	0.622	0.616	7479.159	24.625
DSR (2000)	0.590	0.587	1871.127	21.000
TPSR	0.043	0.046	667.433	40.750
GP-GOMEA	0.645	0.636	7479.687	73.000
DySymNet	0.584	0.456	23766.691	1696.875

Table 12: Top expressions found by each method for predicting crack initiation.

Method	R^2	Complexity	Simplified Expression
CADSR	0.663	21	$c_0 + c_1 x_0 + \frac{c_2}{x_3} + \frac{c_3 x_4 + c_4}{x_1}$
DSR	0.634	26	$-\frac{x_2\left(\exp(x_5 \log(\cos(x_1)) \cos(x_2)) - c_0 \sin(c_1 x_1 - x_5 + c_2)\right)}{x_4}$
DSR (2000)	0.616	21	$-\frac{\sin\left(x_1(x_1+x_4)\exp(c_0 x_1 + c_1 x_2) + c_2\right)}{x_4}$
TPSR	0.372	45	$c_0 - c_1\left(\left(c_2 - c_3 x_0\right) \cdot \left(c_4 \lvert c_5 x_1 - c_6 x_3 + c_7 \rvert - c_8 - \frac{c_9}{c_{10}\sin(-c_{11}x_2 + c_{12}x_4 + c_{13}x_5 + c_{14}) - c_{15}}\right)\right)^{0.5}$
GP-GOMEA	0.668	87	$c_0 + c_1\left(-\sin(c_2 x_2) + c_3 \frac{(x_1+x_4)}{x_3}\right) \cdot \left(c_4 x_2 + c_5 x_3 - x_4 + c_6 x_5 - \cos(x_0 + x_1 - x_3 x_5)\right) \cdot \frac{\cos(x_0 - x_4)}{\sqrt{-x_0 + x_4}} + \frac{c_7 \cos(x_0/x_2)\cos\left(\frac{-x_1 - c_8}{\sqrt{x_1}}\right)}{x_4} + \frac{c_9 \cos(c_{10}/\sqrt{x_3})}{x_3}$