EditFlowSR: Revisable Expression Generation for Symbolic Regression

Yuhong Xu*
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
csyuhongxu@mail.scut.edu.cn

Ziyi Yang*
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
csziyi.yang@mail.scut.edu.cn

Jinghui Zhong†
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
jinghuizhong@scut.edu.cn

Abstract

Symbolic regression (SR) aims to discover mathematical expressions from data. Existing methods based on evolutionary search or autoregressive decoding leave structural refinement to indirect fitness selection or regeneration. We propose EditFlowSR, a pretraining framework that reframes symbolic regression as constructing expressions through iterative edits to an evolving symbolic sequence. We use dual-trajectory supervision to train this editing process jointly for equation discovery and structural refinement. One trajectory recovers expressions from random sequences conditioned on observed data, while the other transforms structurally redundant expressions into equivalent compact forms. Both trajectories use a shared editor and are optimized under a unified objective. Across the standard SRBench, EditFlowSR reaches the first Pareto front in predictive accuracy, expression complexity, and inference time. These results establish edit-based generation as a principled and effective paradigm for end-to-end symbolic regression. Code is available at https://github.com/SCUT-EIPI/EditFlowSR.

1 Introduction

Symbolic regression aims to discover mathematical expressions that explain observed data. Given N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, it seeks an expression f such that $y_i \approx f(\mathbf{x}_i)$ for all i captures the functional relationship, while balancing predictive accuracy and expression simplicity. Unlike traditional regression methods that assume a predefined functional form, symbolic regression searches over both expression structures and their parameters to identify the equation that best fits the observed data [Cava et al., 2021, Dong and Zhong, 2025]. The resulting equations provide compact descriptions of complex systems and insights into the underlying physical processes, making symbolic regression valuable for scientific discovery and engineering modeling [Strogatz, 2024, Udrescu and Tegmark, 2020]. Finding such an equation is challenging because it requires navigating the vast combinatorial space of possible expressions, where the suitability of each operator or subexpression depends on its role within the entire expression.

*Equal contribution.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Existing methods generate expressions with fixed rules. Many of them search for equations with heuristic methods such as genetic programming, which encodes expressions as trees and iteratively evolves a population of candidate expressions through mutation and crossover [Virgolin et al., 2021]. Other approaches rely on Monte Carlo tree search [Sun et al., 2023], Bayesian inference [Jin et al., 2019], or reinforcement-learning policies [Petersen et al., 2021, Xu et al., 2024], while recent methods use neural networks to generate candidates or guide the exploration [Han et al., 2025]. These methods select expression structures only indirectly through fitness evaluation and rely on variation operators to modify them. Another line of methods instead trains models on large synthetic datasets to map observed data to expressions through various pretraining strategies. These models produce an expression as a token sequence, one token at a time in a fixed serialization order [Biggio et al., 2021, Kamienny et al., 2022, Holt et al., 2023, Li et al., 2023, Meidani et al., 2024, Popov et al., 2023]. Under this autoregressive decoding paradigm, each newly generated token becomes a condition for all subsequent ones. As a result, even when external search is incorporated into decoding [Shojaee et al., 2023], an earlier part of the expression cannot be revised in place. Structural repair is thus performed through pruning and regeneration, and the model never learns to suppress redundancy as part of generation itself. Recent advances have also leveraged the scientific knowledge embedded in large language models (LLMs) to guide equation search and refinement, or to produce equations directly as programs [Shojaee et al., 2025, Li and Pan, 2026, Xia et al., 2026]. Among these methods, structural changes take the form of random variation or complete regeneration, and none of them learns the location and content of the modifications an expression requires, leaving generation decoupled from structural repair.

A natural approach to expression generation is therefore to insert the required structure and correct the mischosen components as the expression takes shape. In an expression, each operator or subexpression is linked through its children and ancestors to a broader part, so changing one component requires considering the expression as a whole and may require corresponding adjustments to other parts. This observation motivates generating expressions through edits to an evolving symbolic sequence, with each edit guided by the observed data and the overall structure of the current expression.

To this end, we propose EditFlowSR, a neural SR framework that reformulates expression generation as an iterative editing process. We use Edit Flows [Havasi et al., 2025] to implement this process, which naturally accommodates variable-length sequences through a continuous-time Markov chain. Conditioned on the observed data and the current sequence, EditFlowSR learns to apply edits that transform random symbolic sequences toward target expressions. We further exploit this editing framework to provide dual-trajectory supervision. The discovery trajectory trains the model to recover equations from random symbolic sequences conditioned on observed data, while the repair trajectory trains it to map algebraically redundant expressions to their concise equivalents. At inference time, EditFlowSR starts from a randomly initialized sequence and progressively refines it through iterative editing, simultaneously constructing valid structure and removing redundancy without any external pruning or post-processing. Our contributions are as follows:

- We propose EditFlowSR, a neural symbolic regression framework that reformulates expression generation as iterative editing of an evolving expression, with each edit conditioned on the observed data and the current expression.
- We introduce dual-trajectory supervision that jointly trains the model to discover equations from observed data and simplify algebraically redundant expressions, providing an explicit training signal for generating concise expressions.
- Experiments show that EditFlowSR reliably corrects flawed structures and achieves the first Pareto front in predictive accuracy, expression simplicity, and inference time.

2 Related Work

Existing symbolic regression methods can be broadly grouped into search-based and generation-based approaches. Search-based methods explore the space of expression structures by iteratively generating and evaluating candidate equations. Genetic programming represents expressions as trees and evolves a population of candidates through mutation and crossover, with representative methods including GPlearn [Stephens, 2015], Operon [Burlacu et al., 2020], and GP-GOMEA [Virgolin et al., 2021]. Other approaches navigate the space with Monte Carlo tree search [Sun et al., 2023], Bayesian inference [Jin et al., 2019], or reinforcement-learning policies [Petersen et al., 2021, Xu et al., 2024].

Neural models can also supply learned representations [Zhang et al., 2026, Meidani et al., 2024, Li et al., 2026] and pretrained priors [Yu et al., 2025] that guide the search. Large language models have likewise been incorporated into the search process, leveraging their embedded scientific knowledge to propose equation hypotheses [Shojaee et al., 2025, Li and Pan, 2026, Xia et al., 2026]. In these methods, structural changes are determined by search procedures and their associated transformation operators [Imai Aldeia et al., 2024, de França and Kronberger, 2023, 2025].

Generation-based methods train neural models on large synthetic datasets to learn a conditional mapping from observed data to mathematical expressions. NeSymReS [Biggio et al., 2021] and E2E [Kamienny et al., 2022] generate expression sequences with autoregressive decoders. Variants of these models further explore richer data representations [Holt et al., 2023], alternative supervision signals [Li et al., 2023], and different generative architectures [Popov et al., 2023, Mežnar et al., 2023]. These methods shift much of the cost of equation discovery to pretraining and provide fast inference. Other extensions either incorporate external search into decoding, as in TPSR [Shojaee et al., 2023], or embed symbolic equivalence via equality graphs [Jiang et al., 2026]. Nevertheless, revision in these methods relies either on regeneration under the autoregressive framework or on external pruning. In both cases, the model itself is never trained to prefer concise structures, leaving generation decoupled from redundancy reduction. In this work, we propose EditFlowSR, which formulates expression generation as an editing process that repeatedly revises an evolving expression based on its overall structure.

3 Background

Continuous-time Markov chains. A continuous-time Markov chain (CTMC) over a discrete space $\mathcal{X}$ is a Markov process that generates trajectories $(X_t)_{t\in[0,1]}$ and is characterized by a rate u_t denoting the infinitesimal transition probabilities between states,

$$\mathbb{P}(X_{t+h} = X' \mid X_t = X) = \delta_X(X') + h\,u_t(X' \mid X) + o(h), \tag{1}$$

where $\delta_X(X')$ is the Kronecker delta and $\lim_{h\to 0} o(h)/h = 0$. The rates satisfy $u_t(X' \mid X) \geq 0$ for $X' \neq X$ and $\sum_{X'} u_t(X' \mid X) = 0$. A rate u_t generates a probability path p_t if the time marginals of the associated CTMC satisfy $X_t \sim p_t$. The probability evolution is governed by the Kolmogorov forward equation. Further background on CTMCs is provided in Appendix A.1.

Discrete flow matching. Discrete flow matching (DFM) [Gat et al., 2024] learns a CTMC-based generative model that transports a source distribution p to a target distribution q. DFM prescribes a coupling $\pi(X_0, X_1)$ with marginals p and q, and a conditional rate $u_t(X' \mid X_t, X_0, X_1)$ that generates a conditional probability path $p_t(X \mid X_0, X_1)$ between the endpoints. The conditional path satisfies $p_0(X \mid X_0, X_1) = \delta_{X_0}(X)$ and $p_1(X \mid X_0, X_1) = \delta_{X_1}(X)$. Averaging these paths over π gives the marginal path $p_t(X)$ connecting p and q. The marginal rate $u_t(X' \mid X_t) = \mathbb{E}_{p_t(X_0, X_1 \mid X_t)}[u_t(X' \mid X_t, X_0, X_1)]$ generates the marginal probability path p_t. Prior works train a model to approximate this marginal rate using objectives captured by the family of Bregman divergences. For sequences of fixed length M over a vocabulary $\mathcal{T}$, $X \in \mathcal{T}^M$, a common choice is the token-wise mixture path

$$p_t(X \mid X_0, X_1) := \prod_{i=1}^{M} \left[(1-\kappa_t)\delta_{X_0^i}(X^i) + \kappa_t \delta_{X_1^i}(X^i) \right], \tag{2}$$

where κ_t is a differentiable, nondecreasing schedule with $\kappa_0 = 0$ and $\kappa_1 = 1$. Given the endpoints, each position independently retains its source token with probability $1 - \kappa_t$ or takes its target token with probability κ_t. For $t < 1$, each token that differs from its target switches to that target at rate $\dot{\kappa}_t/(1-\kappa_t)$. Each transition changes one position while leaving all other positions unchanged, preserving sequence length. Details are given in Appendix A.2.

Edit flows. Edit Flows [Havasi et al., 2025] introduces a CTMC-based generative model using edit operations to enable variable length generation. The state space is $\mathcal{X} := \bigcup_{k=0}^{L} \mathcal{T}^k$, where L is the maximum sequence length. The rate $u_t(X' \mid X)$ is non-zero for $X' \neq X$ only when the sequences differ by one insertion, deletion, or substitution. For $X = (X^1, \ldots, X^{|X|})$, these operations are

$$\begin{aligned}
\mathrm{ins}(X, i, a) &:= (X^1, \ldots, X^i, a, X^{i+1}, \ldots, X^{|X|}), \\
\mathrm{del}(X, i) &:= (X^1, \ldots, X^{i-1}, X^{i+1}, \ldots, X^{|X|}), \\
\mathrm{sub}(X, i, a) &:= (X^1, \ldots, X^{i-1}, a, X^{i+1}, \ldots, X^{|X|}),
\end{aligned}$$

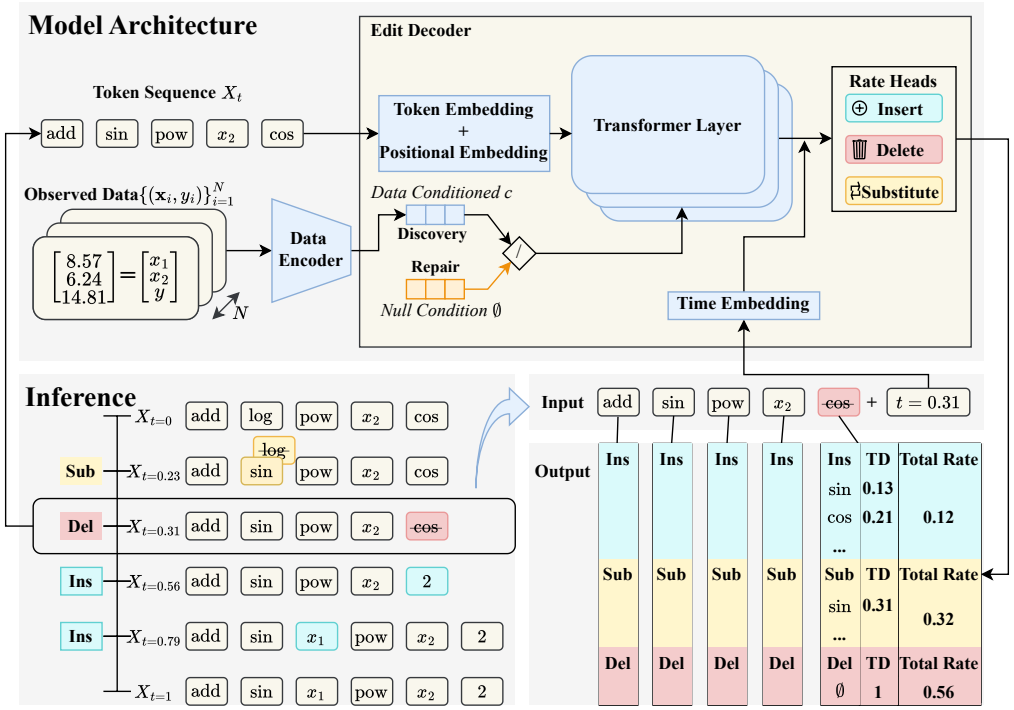


Figure 1: **Overview of EditFlowSR. The upper part presents the model architecture.** Data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ provide the data condition c for the discovery trajectory, while the repair trajectory uses the null condition $\emptyset$. The two trajectories share the editing backbone. **The lower-left part illustrates the inference process.** Starting from a random sequence X_0, EditFlowSR edits it into the final expression X_1. **The lower-right part shows the model inputs and outputs.** Given X_t and time t, the model predicts the total rates of insertion, deletion, and substitution, together with token distributions (TD) for insertion and substitution.

Here, $a \in \mathcal{T}$ is a token and i is a sequence position. Insertion places a after position i, where $i = 0$ denotes insertion before the first token. Deletion and substitution act at position i. These operations define transitions with rates $u_t(\text{ins}(X_t, i, a) \mid X_t)$, $u_t(\text{del}(X_t, i) \mid X_t)$, and $u_t(\text{sub}(X_t, i, a) \mid X_t)$. To obtain tractable conditional supervision, Edit Flows introduces an auxiliary alignment process. Let $\varepsilon \notin \mathcal{T}$ be a blank token, $\mathcal{Z} := (\mathcal{T} \cup \{\varepsilon\})^M$ the aligned sequence space of fixed length M, and $f_{\text{rm-}\varepsilon}$ the map that removes blanks. A chosen alignment of X_0 and X_1 produces aligned endpoints $z_0, z_1 \in \mathcal{Z}$ of the same length with $f_{\text{rm-}\varepsilon}(z_0) = X_0$ and $f_{\text{rm-}\varepsilon}(z_1) = X_1$. Applying Equation (2) to these endpoints gives $z_t \sim p_t(\cdot \mid z_0, z_1)$ and the sequence state $X_t = f_{\text{rm-}\varepsilon}(z_t)$. At each position where $z_t^i \neq z_1^i$, replacing z_t^i with z_1^i specifies a supervised edit. The changes $\varepsilon \to a$, $a \to \varepsilon$, and $a \to b$ correspond to insertion, deletion, and substitution, respectively. Training with these conditional rates under a Bregman objective yields the same gradient in expectation as the corresponding objective on the marginal rates in $\mathcal{X}$, with further details in Appendix A.3. **For example,** consider $X_0 = (\texttt{add}, x_1, x_2)$ and $X_1 = (\texttt{mul}, \texttt{sin}, x_1, x_2)$ with a given minimum edit alignment

$$z_0 = (\texttt{add}, \varepsilon, x_1, x_2), \qquad z_1 = (\texttt{mul}, \texttt{sin}, x_1, x_2).$$

Consider the intermediate state $z_t = (\texttt{mul}, \varepsilon, x_1, x_2)$, which gives $X_t = f_{\text{rm-}\varepsilon}(z_t) = (\texttt{mul}, x_1, x_2)$. At the second aligned position, z_t holds ε while z_1 holds $\texttt{sin}$, so the supervised edit is to insert $\texttt{sin}$ after the first token of X_t.

4 Method

We frame equation discovery as iterative expression editing and realize it with the Edit Flows framework introduced in Section 3. The following subsections describe the data preparation and

representation schemes in Section 4.1, the model architecture in Section 4.2, the dual-trajectory supervision in Section 4.3, and the inference procedure in Section 4.4.

4.1 Data Preparation and Representation

This section describes the synthesis of pretraining pairs consisting of an equation f and its samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $y_i = f(\mathbf{x}_i)$, and the representations of these samples and expressions.

Synthetic data generation. Following [Kamienny et al., 2022], we first sample the input dimension D uniformly from $\{1, \ldots, 10\}$, and then generate a random expression tree for each equation $f : \mathbb{R}^D \to \mathbb{R}$. The binary operators are sampled from $\{+, -, \times\}$, and the unary operators from $\{|\cdot|, (\cdot)^{-1}, \sqrt{\cdot}, \log(\cdot), \exp(\cdot), \sin(\cdot), \cos(\cdot), \tan(\cdot), \arctan(\cdot), (\cdot)^2, (\cdot)^3\}$. The initially sampled tree uses all D input variables and contains between $D-1$ and $D+3$ binary operators. The number of unary operators is sampled independently of D, providing separate control over expression complexity. The sample count N is drawn uniformly from $\{5D, \ldots, 200\}$ given D. Each equation is paired with samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathbb{R}^D \times \mathbb{R}$, where $y_i = f(\mathbf{x}_i)$. The inputs $\mathbf{x}_i$ are sampled from randomly rotated and standardized mixtures of Gaussian or uniform components. The equation–sample pairs are generated online during training.

Numerical and symbolic representation. Following [Kamienny et al., 2022], each number is tokenized with a base-10 floating-point representation, where the value is rounded to four significant digits and decomposed into sign, mantissa, and exponent tokens, e.g., 0.7895 becomes $[+,\ 7895,\ E{-}4]$, with the mantissa ranging from 0 to 9999 and the exponent from $E{-}100$ to $E100$. Each sample $(\mathbf{x}_i, y_i)$ is encoded as a token group and mapped by a learnable point embedder to a point embedding. Symbolic expressions are represented as binary trees and converted to token sequences via pre-order traversal. Operators, variables, and integers are mapped to dedicated tokens, and constants are tokenized in the same way as the numerical values.

4.2 Model Architecture

As illustrated in Figure 1, EditFlowSR encodes the data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ into the data condition c, then feeds c, the current token sequence X_t, and time t to the editing decoder, which predicts total rates and token distributions at each position of X_t.

Data encoder. The point embeddings from Section 4.1 are processed by an 8-layer Transformer encoder. The Transformer encoder produces contextualized point embeddings by modeling interactions among the samples and uses no positional embeddings, so that the encoding is invariant to the order of the samples. The contextualized point embeddings constitute the data condition c and are provided to the editing decoder through cross-attention, allowing every editing decision to access the individual samples.

Editing decoder. Given the current token sequence X_t, the editing decoder maps each token to a token embedding and adds a learnable positional embedding to form the input representations. The editing decoder contains 8 Transformer layers that transform the input representations into hidden states. Each Transformer layer applies non-causal self-attention across all positions, allowing each editing decision to account for the global expression structure. It then applies cross-attention to the contextualized point embeddings in the data condition c and a feed-forward subnetwork, with a residual connection and layer normalization after each sublayer. The time t is encoded independently by a sinusoidal time embedding followed by an MLP. At each position, the hidden state and the time representation jointly determine the outputs of three two-layer prediction heads. The rate head jointly predicts the total rates of insertion, deletion, and substitution. The insertion and substitution heads each predict a token distribution over the equation vocabulary. These outputs are formally defined and combined in Section 4.3 into the transition rates defined in Section 3. Detailed configurations of the data encoder and editing decoder are provided in Appendix B.1.

4.3 Dual-Trajectory Supervision

This subsection specifies the endpoints and conditioning of the two trajectories, combines the total rates and token distributions predicted in Section 4.2 into transition rates for the three edit operations $u_t(\mathrm{ins}(X_t, i, a) \mid X_t)$, $u_t(\mathrm{del}(X_t, i) \mid X_t)$, and $u_t(\mathrm{sub}(X_t, i, a) \mid X_t)$ defined in Section 3, and presents the training objective for dual-trajectory supervision.

Training trajectories. EditFlowSR trains the editing model on two trajectories with a shared editing backbone but different supervision sources and conditioning inputs. The discovery trajectory recovers the equation from a sequence of sampled tokens, and the repair trajectory transforms redundant expression token sequences into equivalent simplified forms. All target expressions are sampled with the random generator of Section 4.1. The discovery trajectory draws a sequence of random tokens as the source X_0 and the expression token sequence of the equation underlying the data as the target X_1, conditioned on the data condition c. The repair trajectory uses an algebraically redundant expression as the source X_0 and a concise expression as the target X_1, with X_0 constructed by expanding X_1 using the rules in Appendix B.2. The model learns this transformation under the null condition $\emptyset$, independently of the data. For both trajectories, the alignment between X_0 and X_1 is computed with the edit-distance dynamic program of Appendix B.3, which fixes the training path between the endpoints. Table 1 summarizes the source, the target, and the conditioning of the two trajectories.

Table 1: The two training trajectories of EditFlowSR.

Trajectory	Source X_0	Target X_1	Condition
Discovery	Random tokens	Target expression	Data condition c
Repair	Redundant expansion of X_1	Target expression	Null condition $\emptyset$

Rate decomposition. Following the edit operations of Section 3 and the model outputs of Section 4.2, each transition rate u_t is decomposed into a total rate $\lambda^e_{t,i}$ and a token distribution $Q^e_{t,i}$,

$$u_t(e \mid X) := \lambda^e_{t,i}(X)\, Q^e_{t,i}(a \mid X), \tag{3}$$

where $e \in \{\text{ins}, \text{del}, \text{sub}\}$ denotes the operation type, and the total rate $\lambda^e_{t,i}$ is the rate of operation e at position i marginalized over the inserted or substituted token. For insertion and substitution, $Q^e_{t,i}(a \mid X)$ is the conditional probability of selecting token a when operation e is applied at position i. Since deletion involves no token choice, $Q^{\text{del}}_{t,i} \equiv 1$ and $u_t(\text{del}(X,i) \mid X) = \lambda^{\text{del}}_{t,i}(X)$.

Training objective. Given X_0 and X_1, we obtain the aligned endpoints z_0 and z_1 and, following Section 3, sample t to obtain the interpolated aligned state z_t and the sequence state X_t. The loss for a single trajectory is

$$\begin{aligned}\mathcal{L}_{\text{traj}}(\theta, c) := \mathbb{E}_{t, z_0, z_1, z_t \sim p_t(\cdot \mid z_0, z_1)} \Big[& \sum_{X \neq X_t} u^\theta_t(X \mid X_t, c) \\ & - \sum_{i=1}^{M} \mathbf{1}\{z_1^i \neq z_t^i\} \frac{\dot\kappa_t}{1-\kappa_t} \log u^\theta_t\big(X(z_t, i, z_1^i) \mid X_t, c\big) \Big],\end{aligned} \tag{4}$$

where $X(z_t, i, a) := f_{\text{rm-}\varepsilon}(z_t^1, \ldots, z_t^{i-1}, a, z_t^{i+1}, \ldots, z_t^M)$ denotes the sequence in the original sequence space obtained by replacing the i-th aligned token z_t^i with a and removing ε, which corresponds to exactly one edit of the three types. Under the decomposition of Equation (3), the first term reduces to $\sum_{X \neq X_t} u^\theta_t(X \mid X_t, c) = \sum_i \big(\lambda^{\text{ins}}_{t,i}(X_t) + \lambda^{\text{del}}_{t,i}(X_t) + \lambda^{\text{sub}}_{t,i}(X_t)\big)$, since the $Q^e_{t,i}$ sum to one. Minimizing the first term pushes all outgoing rates down, while the second term is a weighted cross-entropy over the edits that bring X_t closer to X_1. The weight $\dot\kappa_t/(1-\kappa_t)$ is the conditional rate of the path in Equation (2) applied to the aligned endpoints, as defined in Equation (13). The time t is sampled uniformly from $(0,1)$. The final objective mixes the two trajectories with the trajectory probability ρ:

$$\mathcal{L}(\theta) := (1-\rho)\, \mathbb{E}_{\text{discover}}\big[\mathcal{L}_{\text{traj}}(\theta, c)\big] + \rho\, \mathbb{E}_{\text{repair}}\big[\mathcal{L}_{\text{traj}}(\theta, \emptyset)\big]. \tag{5}$$

Interpolation schedule. Following prior work on discrete flow matching [Gat et al., 2024, Havasi et al., 2025], EditFlowSR uses a cubic interpolation schedule. In this setting, κ_t is the probability that a position takes its target token, so the endpoints $\kappa_0 = 0$ and $\kappa_1 = 1$ follow from the path definition, while the derivative $\dot\kappa_t$ controls how quickly the mixture weight assigned to the target token increases. To encourage the training path to move away from the random initial sequence quickly and approach the target more gradually, we set the endpoint slopes to $\dot\kappa_0 = 2$ and $\dot\kappa_1 = 0.5$. Cubic Hermite interpolation gives the polynomial of lowest degree that can satisfy the four endpoint constraints $\kappa_0 = 0$, $\kappa_1 = 1$, $\dot\kappa_0 = 2$, and $\dot\kappa_1 = 0.5$ independently, yielding

$$\kappa_t = \tfrac{1}{2}t^3 - \tfrac{3}{2}t^2 + 2t, \tag{6}$$

whose derivative $\dot\kappa_t = 0.5 + 1.5(1-t)^2$ decreases from 2 to 0.5 on $t \in [0,1]$, so that κ_t is strictly increasing, lies in $[0,1]$, and keeps $1 - \kappa_t > 0$ for $t < 1$. See the training details in Appendix B.4.

4.4 Inference

Given samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the data encoder produces the data condition c, and inference uses the editing decoder with the data condition. The initial state X_0 is a sequence of random tokens, as in the discovery trajectory. EditFlowSR transports X_0 to $t = 1$ by simulating the CTMC defined by the learned rates in n_{steps} sampling steps with step size $\Delta t = 1/n_{\text{steps}}$. At each step, the editing decoder predicts the total rates and token distributions of Equation (3) for the current state X_t. Insertion writes a token between existing ones and does not modify any existing token, so it does not compete with the other operations and is sampled first and independently, with probability $\Delta t\, \lambda_{t,i}^{\text{ins}}$ and the inserted token drawn from $Q_{t,i}^{\text{ins}}$. Deletion and substitution act on an occupied position and cannot occur simultaneously, so they are sampled as a single event with probability $\Delta t\, (\lambda_{t,i}^{\text{del}} + \lambda_{t,i}^{\text{sub}})$; when the event occurs, a deletion is performed with probability $\lambda_{t,i}^{\text{del}} / (\lambda_{t,i}^{\text{del}} + \lambda_{t,i}^{\text{sub}})$, and otherwise a substitution applies a new token drawn from the substitution token distribution $Q_{t,i}^{\text{sub}}(\cdot \mid X_t)$. After the edit operations have been sampled at all positions, substitutions are first applied. An insertion and a deletion that coincide at the same position are merged into a single substitution that writes the inserted token. The remaining deletions and insertions are applied last. The constants of the final expression are refined with BFGS [Petersen et al., 2021, Biggio et al., 2021, Kamienny et al., 2022]. The inference procedure and configuration are provided in Algorithm 2 in Appendix B.5.

5 Experiments

5.1 Experimental Setup

Following [Li et al., 2026], we evaluate on SRBench [Cava et al., 2021], which comprises the Feynman, Strogatz, and Black-box datasets. The Feynman dataset contains 119 physics equations from the *Feynman Lectures on Physics* [Udrescu and Tegmark, 2020], with input dimension up to $D = 10$. The Strogatz dataset contains 14 regression tasks derived from nonlinear dynamical systems [Strogatz, 2024], each with input dimension $D = 2$, 400 samples, and a known target expression. The Black-box dataset consists of 89 regression tasks drawn from the Penn Machine Learning Benchmark [Olson et al., 2017], with input dimension $D \leq 10$ and unknown target functions. For each task, 75% of the data points are used as fit data for expression generation and constant optimization, while the remaining 25% are used to compute test R^2. We compare EditFlowSR with 22 representative baselines covering search-based and generation-based symbolic regression, including LLM-guided search methods, together with sparse regression and conventional non-symbolic baselines, as detailed in Appendix C.1. We evaluate predictive accuracy with the test coefficient of determination and equation complexity with the number of tokens in the expression,

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \qquad \text{Complexity}(f) = |T[f]|,$$

where $\hat{y}_i$ denotes the value predicted by the expression f, $\bar{y}$ denotes the mean of the true values, and $|T[f]|$ denotes the number of tokens in the preorder serialization of the simplified expression f. Computational efficiency is measured by the wall-clock inference time in seconds, counted until a method outputs its final symbolic expression or, for non-symbolic references, a final fitted predictor. All generated expressions are simplified with SymPy before computing R^2 and complexity. Additional analyses are in Appendix D.

5.2 Comparison with Other Methods

Pareto performance analysis. As shown in Figure 2 on the Black-box dataset, which is particularly challenging for pretraining methods [Kamienny et al., 2022] due to the potential absence of simple symbolic expressions to model this dataset, EditFlowSR lies on the first Pareto front in both comparisons. In this setting, EditFlowSR achieves leading predictive accuracy, which is a notable advantage for a generation-based method over existing approaches. As task difficulty increases, the complexity of the generated expressions also remains low, mitigating the overfitting issue. Moreover, as a generation-based method, EditFlowSR achieves shorter runtime and avoids the combinatorial explosion of the search space, which makes search-based and hybrid methods computationally expensive on more complex tasks. See Appendix C.3 for the full results.

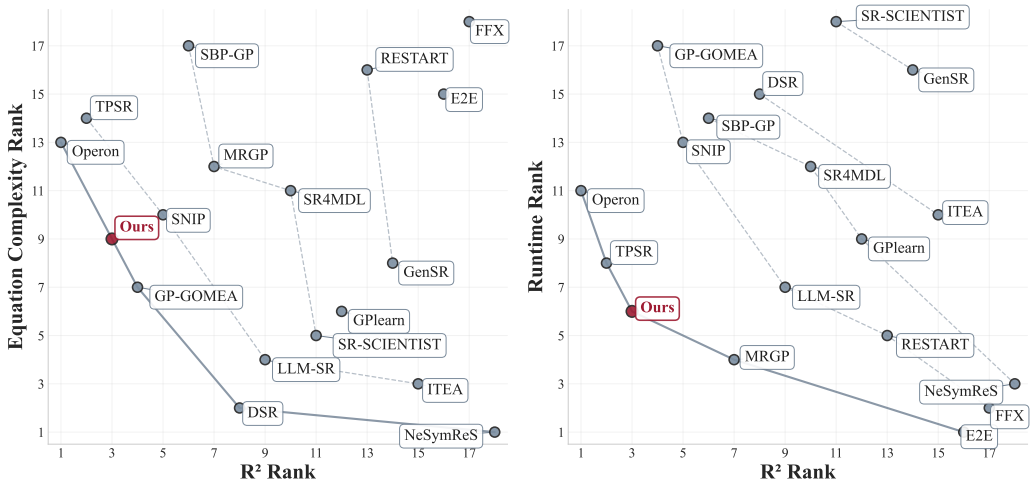


Figure 2: Pareto-front results on the Black-box dataset. The x-axis shows the mean test R^2 rank, while the y-axes show the ranks of equation complexity (**left**) and runtime (**right**). Lower ranks indicate better performance. Solid and dashed lines denote the first and subsequent fronts.

Performance under increasing noise. Figure 3 compares EditFlowSR with methods that report equation complexity on the Strogatz dataset across four noise levels, $\sigma \in \{0, 0.001, 0.01, 0.1\}$ (conventional non-symbolic baselines output a fitted predictor instead of a symbolic expression and thus have no equation complexity; see Appendix C.2 for details on noise injection). EditFlowSR shows a clear advantage in noisy settings, as the test R^2 of several baselines drops as noise increases, whereas EditFlowSR remains stable. EditFlowSR's favorable complexity and stable runtime highlight the benefit of edit-based generation in controlling expression complexity under noisy conditions, avoiding error accumulation in autoregressive decoding and fitting noise when candidates are selected or guided by fitting performance. See Appendix C.3 for the full results.

5.3 Ablation Study

Repair-trajectory ablation. We compare three variants to isolate the contribution of the repair trajectory. EditFlowSR (No Repair) sets the probability of using the repair trajectory to $\rho = 0$. EditFlowSR (Non-equivalent Shortening) retains the repair-trajectory construction scheme but uses trajectories from longer expressions to shorter expressions whose endpoints are not required to be equivalent, assessing whether encouraging shorter expressions is effective without the equivalence relations used by the full repair trajectory.

As shown in Table 2, the full EditFlowSR model simultaneously improves predictive accuracy and reduces expression complexity across all settings. The repair trajectory incorporates equivalence-preserving shortening knowledge into the shared backbone, so the equation generator captures the fundamental relations in the data with concise expressions that have few degrees of freedom to fit noise, yielding better test performance and robustness.

Redundancy localization. This experiment tests whether EditFlowSR assigns higher deletion scores to truly redundant tokens than to structurally similar valid tokens. For each concise base expression, a positive probe injects one redundant fragment, while a matched negative probe keeps the same local syntax without true redundancy. Metric definitions are given in Appendix C.4. Table 3 shows that the full model locates redundant positions most accurately. Figure 4 further shows that its deletion score remains high under numerically equivalent small perturbations, yet reliably decreases as the perturbation magnitude η grows into a potentially meaningful structure. This confirms that the model correctly identifies true redundancy instead of blindly sustaining high deletion scores.

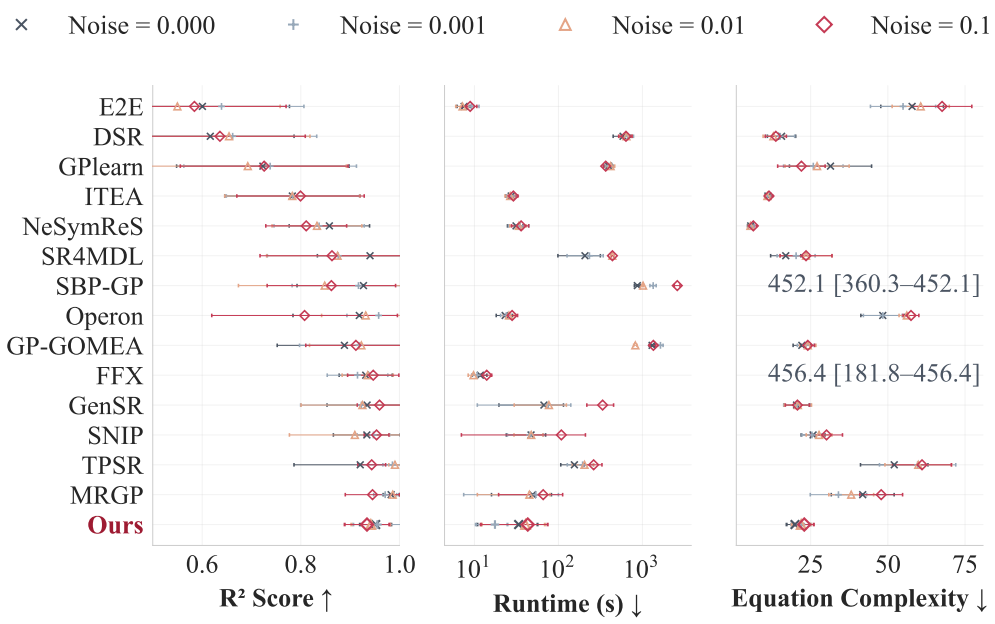


Figure 3: Performance on the Strogatz dataset under four noise levels. The test R^2 (**left**), runtime (**center**), and equation complexity (**right**) panels report the corresponding means, with horizontal error bars representing the 95% confidence interval. For methods whose complexity lies far outside the plotted range, the complexity panel reports the mean at $\sigma = 0.1$ and the range across noise levels as text.

Table 2: Ablation results for the EditFlowSR (No Repair) and EditFlowSR (Non-equivalent Shortening) variants on the Black-box dataset under four noise levels, abbreviated as No Repair and Non-equiv. in the table. Parenthesized values are differences to the full EditFlowSR model.

σ	EditFlowSR	No Repair	Non-equiv.
Test R^2			
0	**0.707**	0.683 (−0.024)	0.660 (−0.047)
0.001	**0.717**	0.701 (−0.016)	0.664 (−0.054)
0.01	**0.736**	0.717 (−0.019)	0.642 (−0.094)
0.1	**0.673**	0.645 (−0.029)	0.613 (−0.060)
Complexity			
0	**38.4**	44.9 (+6.5)	47.3 (+8.9)
0.001	**36.8**	47.1 (+10.3)	49.1 (+12.3)
0.01	**37.7**	42.3 (+4.6)	49.6 (+11.9)
0.1	**36.8**	47.1 (+10.3)	49.0 (+12.2)

6 Conclusion

This paper presented EditFlowSR, which formulates symbolic regression as iterative expression editing conditioned on observed data. Its dual-trajectory training couples equation discovery with structural repair that preserves equivalence through a shared editor and a unified objective. On standard SRBench benchmarks, EditFlowSR reaches the first Pareto front across predictive accuracy, inference time, and expression complexity, while maintaining stable complexity and competitive accuracy under noisy targets. These results support edit-based modeling as an effective approach to direct symbolic expression generation with pretrained models.

Table 3: Token-level redundancy localization on all probes, with tokens ranked by the deletion score. Gap Ratio divides the mean deletion score on redundant tokens by that on matched control tokens, so values above one indicate that redundant tokens are scored higher. Variant names are abbreviated as in Table 2.

Model	Macro-AUPRC ↑	Micro-AUPRC ↑	Gap Ratio ↑	Top-1 ↑
EditFlowSR	0.915	0.740	1.259	1.000
EditFlowSR (No Repair)	0.885	0.690	1.055	0.999
EditFlowSR (Non-equiv.)	0.801	0.666	0.985	0.977

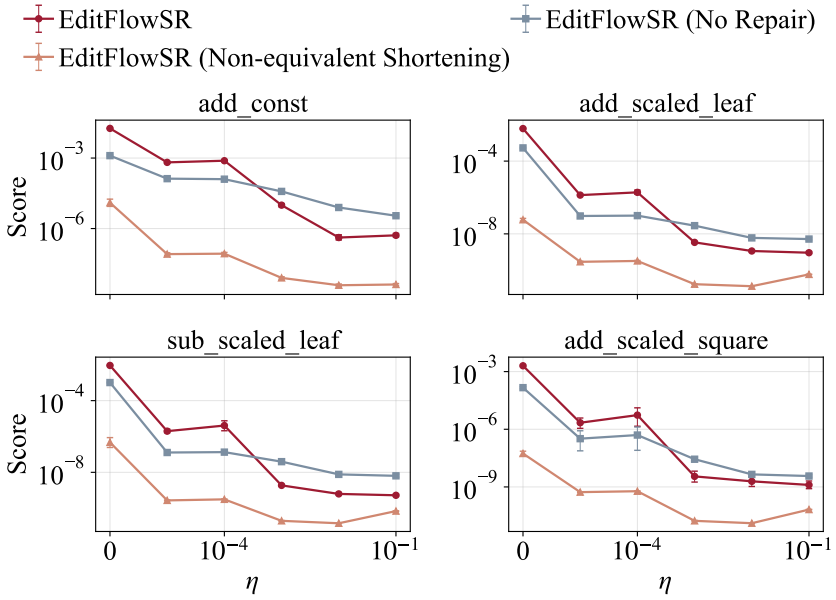


Figure 4: Calibration of deletion scores under approximate perturbations across four perturbation families. Curves plot the mean deletion score assigned to the injected coefficient token as the perturbation magnitude η increases; shaded bands show confidence intervals.

Limitation. EditFlowSR is trained for end-to-end generation, and adapting its editing dynamics to guide optimization may require task-specific training. Future work will explore broader pretraining distributions and search or verification objectives to improve reliability and extrapolation.

Broader Impacts. Symbolic regression can help researchers inspect candidate equations instead of trusting black-box predictors. EditFlowSR improves transparency by exposing the full edit path from initialization to output. Users can inspect intermediate expressions and the edits applied at each step, which provides a concrete basis for examining how a candidate equation was formed and whether its successive changes align with domain knowledge. This visibility can support closer human scrutiny during scientific discovery.

Acknowledgments and Disclosure of Funding

We thank anonymous reviewers for helpful discussions and feedback on an earlier draft of this paper. This work was supported in part by the National Natural Science Foundation of China under Grant 62476096, in part by the Key Core and Pivotal Technology Research Project of the Hengqin Guangdong-Macao In-Depth Cooperation Zone under Grant 2430004045040.

References

Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pages 879–886, 2014.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International conference on machine learning*, pages 936–945. Pmlr, 2021.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon c++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 genetic and evolutionary computation conference companion*, pages 1562–1570, 2020.

William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *NeurIPS Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c0c7c76d30bd3dcaefc96f40275bdc0a-Abstract-round1.html.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Fabrício Olivetti de França and Guilherme Seidyo Imai Aldeia. Interaction–transformation evolutionary algorithm for symbolic regression. *Evolutionary computation*, 29(3):367–390, 2021.

Fabrício Olivetti de França and Gabriel Kronberger. Reducing overparameterization of symbolic regression models with equality saturation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1064–1072, 2023.

Fabrício Olivetti de França and Gabriel Kronberger. Improving genetic programming for symbolic regression with equality graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 989–998, 2025.

Junlan Dong and Jinghui Zhong. Recent advances in symbolic regression. *ACM Computing Surveys*, 57(11): 1–37, 2025.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37:133345–133385, 2024.

Xiaoxu Han, Jinghui Zhong, Zhitong Ma, Xin Mu, and Nikola Gligorovski. Transformer-assisted genetic programming for symbolic regression [research frontier]. *IEEE Computational Intelligence Magazine*, 20(2): 58–79, 2025.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit flows: Variable length discrete flow matching with sequence-level edit operations. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=FXWwYz1p8a.

Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=o7koEEMA1bR.

Guilherme Seidyo Imai Aldeia, Fabrício Olivetti De França, and William G La Cava. Inexact simplification of symbolic regression expressions with locality-sensitive hashing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 896–904, 2024.

Nan Jiang, Ziyi Wang, and Yexiang Xue. EGG-SR: Embedding symbolic equivalence into symbolic regression via equality graph. In *International Conference on Learning Representations*, volume 2026, pages 13650–13689, 2026.

Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Qian Li, Yuxiao Hu, Juncheng Liu, and Yuntian Chen. GenSR: Symbolic regression based on equation generative space. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=8emIjwUQZg.

Wenqiang Li, Weijun Li, Linjun Sun, Min Wu, Lina Yu, Jingyi Liu, Yanjie Li, and Songsong Tian. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ULzyv9M1j5.

Yunlun Li and Sinno Jialin Pan. Robust equation structure learning with adaptive refinement. In *International Conference on Learning Representations*, volume 2026, pages 70029–70053, 2026.

Trent McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pages 235–260. Springer, 2011.

Kazem Meidani, Parshin Shojaee, Chandan K. Reddy, and Amir Barati Farimani. Snip: Bridging mathematical symbolic and numeric realms with unified pre-training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KZSEgJGPxu.

Sebastian Mežnar, Sašo Džeroski, and Ljupčo Todorovski. Efficient generator of mathematical expressions for symbolic regression. *Machine Learning*, 112(11):4563–4596, 2023.

James R Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):36, 2017.

Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=m5Qsh0kBQG.

Sergei Popov, Mikhail Lazarev, Vladislav Belavin, Denis Derkach, and Andrey Ustyuzhanin. Symbolic expression generation via variational auto-encoder. *PeerJ Computer Science*, 9:e1241, 2023.

Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. 1998.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. Transformer-based planning for symbolic regression. *Advances in Neural Information Processing Systems*, 36:45907–45919, 2023.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, volume 2025, pages 16054–16085, 2025.

Trevor Stephens. gplearn: Genetic programming in python. https://github.com/trevorstephens/gplearn, 2015. Software package, accessed 2026-05-03.

Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2024.

Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ZTK3SefE8_Z.

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

Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.

Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter AN Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation*, 29(2):211–237, 2021.

Shijie Xia, Yuhan Sun, and Pengfei Liu. SR-Scientist: Scientific equation discovery with agentic AI. In *International Conference on Learning Representations*, volume 2026, pages 75787–75811, 2026.

Yilong Xu, Yang Liu, and Hao Sun. Reinforcement symbolic regression machine. In *International Conference on Learning Representations*, volume 2024, pages 54416–54440, 2024.

Zihan Yu, Jingtao Ding, Yong Li, and Depeng Jin. Symbolic regression via MDLformer-guided search: from minimizing prediction error to minimizing description length. In *International Conference on Learning Representations*, volume 2025, pages 65306–65333, 2025.

Zihan Yu, Guanren Wang, Jingtao Ding, Huandong Wang, and Yong Li. Beyond accuracy and complexity: The effective information criterion for structurally stable symbolic regression. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=9mFx2t9RGj.

Hengzhe Zhang, Qi Chen, Bing Xue, Wolfgang Banzhaf, and Mengjie Zhang. Contrastive symbolic regression: Aligned representations, adaptive prediction, and diverse ensembles. In *International Conference on Machine Learning*, pages 154542–154567. PMLR, 2026.

Appendix

Table of Contents

1 Introduction **1**

2 Related Work **2**

3 Background **3**

4 Method **4**

4.1 Data Preparation and Representation 5

4.2 Model Architecture 5

4.3 Dual-Trajectory Supervision 5

4.4 Inference 7

5 Experiments **7**

5.1 Experimental Setup 7

5.2 Comparison with Other Methods 7

5.3 Ablation Study 8

6 Conclusion **9**

Appendix **14**

A Technical Background of Edit Flows **16**

A.1 Continuous-time Markov chains 16

A.2 Discrete flow matching 16

A.3 Training Edit Flows 17

B EditFlowSR Implementation Details **18**

B.1 Model Configuration 18

B.1.1 Point embedding 18

B.1.2 Time embedding 19

B.1.3 Data encoder and editing decoder 19

B.1.4 Prediction heads 19

B.2 Repair-Trajectory Expansion Rules 19

B.3 Edit-distance Alignment 19

B.4 Pretraining Procedure 20

B.5 Inference Procedure 20

C Experimental Details **22**

C.1 Baselines 22

C.2 Noise Injection 22

C.3 Complete Numerical Results 22

C.4 Redundancy Localization and Approximate Calibration Protocol 27

D Additional Analyses **28**

D.1 Expression Nesting Depth 28

D.2 Structural Stability Analysis 28

D.3 Sensitivity of Inference Hyperparameters 28

D.4 Qualitative Edit-Path Example 29

A Technical Background of Edit Flows

This appendix provides technical background for the Edit Flows construction used in this work. Section A.1 reviews continuous-time Markov chains and Section A.2 reviews discrete flow matching. Section A.3 follows the training construction of [Havasi et al., 2025] and restates its core theorem, with detailed proofs available in the original paper.

A.1 Continuous-time Markov chains

This section briefly reviews the basic definitions and probability evolution of continuous time Markov chains, providing the foundation for the flow-matching and edit-process constructions in Section 3.

A continuous time Markov chain (CTMC) is a stochastic process that describes random jumps among discrete states, with the jumps occurring on a continuous time axis. It consists of a discrete state space, a set of rate functions, and a state evolution driven by these rates. The state space is the set of values the process can take, and a CTMC describes how the current state X_t evolves over time under random jumps. The Markov property is the central assumption, under which the next state of the system depends only on the current state.

Rate functions characterize the intensity of transitions, as introduced in the theory of Markov chains [Norris, 1998]. For all X', define the rate $u_t(X' \mid X)$ as the infinitesimal transition intensity from state X to state X' at time t, i.e.,

$$u_t(X' \mid X) := \lim_{h \to 0} \frac{\mathbb{P}(X_{t+h} = X' \mid X_t = X) - \delta_X(X')}{h}, \tag{7}$$

which naturally yields the infinitesimal transition probability

$$\mathbb{P}(X_{t+h} = X' \mid X_t = X) = \delta_X(X') + h\, u_t(X' \mid X) + o(h), \tag{8}$$

where $\delta_X(X')$ is the Kronecker delta and $o(h)$ is a higher-order infinitesimal in h. The rate functions form the generator matrix, whose entries $u_t(X' \mid X)$ are non-negative for any $X' \neq X$, and whose diagonal entries satisfy $u_t(X \mid X) = -\sum_{X' \neq X} u_t(X' \mid X)$, so that each row of rates sums to zero, i.e., $\sum_{X'} u_t(X' \mid X) = 0$. This condition guarantees that the total probability mass remains one at every time. A set of rate functions u_t pushes an initial distribution forward into the state distribution $p_t(X)$ at time t; the rates u_t are said to generate the probability path p_t. The evolution of $p_t(X)$ satisfies the Kolmogorov forward equation, whose derivation follows. By the Markov property, the distribution at time $t+h$ is obtained by summing over the state at time t:

$$p_{t+h}(X') = \sum_X p_t(X)\, \mathbb{P}(X_{t+h} = X' \mid X_t = X), \tag{9}$$

where the infinitesimal transition probability $\mathbb{P}(X_{t+h} = X' \mid X_t = X)$ is given by (8). Substituting and splitting the sum into the terms $X = X'$ and $X \neq X'$ gives

$$p_{t+h}(X') = p_t(X')\big(1 + h\, u_t(X' \mid X')\big) + h \sum_{X \neq X'} p_t(X) u_t(X' \mid X) + o(h). \tag{10}$$

Subtracting $p_t(X')$ from both sides, dividing by h, and letting $h \to 0$, using the diagonal entry $u_t(X' \mid X') = -\sum_{X \neq X'} u_t(X \mid X')$, yields the Kolmogorov forward equation

$$\frac{d\, p_t(X')}{dt} = \sum_{X \neq X'} u_t(X' \mid X) p_t(X) - p_t(X') \sum_{X \neq X'} u_t(X \mid X'), \tag{11}$$

where the first term is the total inflow rate at which other states X enter X' at rate $u_t(X' \mid X)$, and the second term is the total outflow rate at which state X' leaves toward other states at rate $u_t(X \mid X')$.

A.2 Discrete flow matching

Discrete flow matching (DFM) learns transition rates that transport a source distribution to a target distribution over discrete states [Gat et al., 2024]. This section provides further details on the DFM formulation introduced in Section 3.

Let p and q denote the source and target distributions on a discrete sequence space, respectively. DFM specifies a coupling distribution $\pi(X_0, X_1)$ whose marginals are p and q, so that $X_0 \sim p$ and $X_1 \sim q$

for $(X_0, X_1) \sim \pi$. For each endpoint pair, DFM defines a conditional probability path $p_t(X \mid X_0, X_1)$ that starts from X_0 and ends at X_1:

$$p_0(X \mid X_0, X_1) = \delta_{X_0}(X), \qquad p_1(X \mid X_0, X_1) = \delta_{X_1}(X).$$

A simple and useful choice is the token-wise interpolation path. For a sequence of fixed length M, each position is interpolated independently. Let κ_t be a differentiable, nondecreasing schedule with $\kappa_0 = 0$ and $\kappa_1 = 1$. At position i, the intermediate token retains the source token X_0^i with probability $1 - \kappa_t$ and takes the target token X_1^i with probability κ_t:

$$p_t(X \mid X_0, X_1) := \prod_{i=1}^{M} \left[(1 - \kappa_t)\delta_{X_0^i}(X^i) + \kappa_t \delta_{X_1^i}(X^i) \right]. \tag{12}$$

Thus, sampling from this path amounts to independently choosing the source or target token at every position, and the resulting sequence changes progressively from X_0 to X_1 as t increases. The conditional transition rate is the infinitesimal rate induced by this path. For one position, it is

$$u_t(X^i \mid X_t^i, X_0^i, X_1^i) := \frac{\dot{\kappa}_t}{1 - \kappa_t} \left(\delta_{X_1^i}(X^i) - \delta_{X_t^i}(X^i) \right), \tag{13}$$

where the rate is defined for $t < 1$ and the endpoint at $t = 1$ is understood in the limiting sense. In the sequence space, the rate is non-zero only for transitions that change one position at a time.

The conditional paths are averaged over endpoint pairs according to the coupling to obtain the marginal probability path

$$p_t(X) := \sum_{X_0, X_1} p_t(X \mid X_0, X_1)\, \pi(X_0, X_1), \tag{14}$$

which connects the source and target distributions, satisfying $p_0(X) = p(X)$ and $p_1(X) = q(X)$. One can verify that the marginal rate

$$u_t(X \mid X_t) := \mathbb{E}_{p_t(X_0, X_1 \mid X_t)}\, u_t(X \mid X_t, X_0, X_1) \tag{15}$$

generates the marginal probability path p_t.

A.3 Training Edit Flows

Edit Flows generalizes beyond the token-wise paths reviewed in Appendix A.2. The main difficulty in constructing a conditional rate directly on the sequence space $\mathcal{X}$ is the need to account for all possible transitions between sequences, including multiple insertions that produce the same successor sequence. Havasi et al. [Havasi et al., 2025] extend the DFM training recipe with an auxiliary Markov process, making it possible to train Edit Flows using Bregman divergences.

Discrete flow matching with auxiliary Markov processes. Suppose a CTMC on $\mathcal{X}$ follows the marginals of a CTMC on the augmented space $\mathcal{X} \times \mathcal{Z}$. The auxiliary process supplies a tractable training signal for a model that operates directly on $\mathcal{X}$. The following result restates Theorem 3.1 of [Havasi et al., 2025] in our notation.

Theorem A.1 (Flow Matching with Auxiliary Processes)**.** *Let $\tilde{u}_t(X, z \mid X_t, z_t)$ be a rate on $\mathcal{X} \times \mathcal{Z}$ that generates a probability path $\tilde{p}_t(X, z)$. Then the marginal rate*

$$u_t(X \mid X_t) := \sum_z \mathbb{E}_{\tilde{p}_t(z_t \mid X_t)} \left[\tilde{u}_t(X, z \mid X_t, z_t) \right] \tag{16}$$

generates the marginal probability path $p_t(X) := \sum_z \tilde{p}_t(X, z)$. Furthermore, for any Bregman divergence

$$D_\phi(a, b) := \phi(a) - \phi(b) - \langle a - b, \nabla\phi(b) \rangle$$

defined by a differentiable convex function ϕ, and a model rate $u_t^\theta(\cdot \mid X_t)$ that does not depend on z_t,

$$\begin{aligned} &\nabla_\theta \mathbb{E}_{(X_t, z_t) \sim \tilde{p}_t} D_\phi \left(\sum_z \tilde{u}_t(\cdot, z \mid X_t, z_t), u_t^\theta(\cdot \mid X_t) \right) \\ &\qquad = \nabla_\theta \mathbb{E}_{X_t \sim p_t} D_\phi \left(u_t(\cdot \mid X_t), u_t^\theta(\cdot \mid X_t) \right). \end{aligned} \tag{17}$$

The divergence is evaluated on rates to successor sequences $X \neq X_t$, with all auxiliary transitions producing the same X summed before evaluation.

The theorem allows auxiliary processes that depend on time and guarantees equality of the expected gradients. Detailed proofs are given in Appendix B of [Havasi et al., 2025].

Training with an auxiliary alignment process. Even with fixed endpoints X_0 and X_1, directly constructing a conditional rate on $\mathcal{X}$ requires accounting for multiple possible sets of edits. An alignment specifies a precise set of edits transforming one sequence into the other, allowing a simpler conditional process in an augmented space. Let $\varepsilon \notin \mathcal{T}$ be a blank token and $\mathcal{Z} = (\mathcal{T} \cup \{\varepsilon\})^M$ the space of aligned sequences. The map $f_{\text{rm-}\varepsilon} : \mathcal{Z} \to \mathcal{X}$ removes all blank tokens. The blank token is used only to define the auxiliary process and its supervision, and is absent from the model inputs and outputs. Changes $\varepsilon \to a$, $a \to \varepsilon$, and $a \to b$ with $a \neq b$ correspond to insertion, deletion, and substitution, respectively.

Given source and target samples X_0 and X_1, an alignment produces $z_0, z_1 \in \mathcal{Z}$ of the same length with $f_{\text{rm-}\varepsilon}(z_0) = X_0$ and $f_{\text{rm-}\varepsilon}(z_1) = X_1$. This induces a coupling $\pi(z_0, z_1)$ satisfying

$$\begin{aligned} p(X) &= \sum_{z_0, z_1} \pi(z_0, z_1) \delta_{f_{\text{rm-}\varepsilon}(z_0)}(X), \\ q(X) &= \sum_{z_0, z_1} \pi(z_0, z_1) \delta_{f_{\text{rm-}\varepsilon}(z_1)}(X). \end{aligned} \tag{18}$$

In this work, the alignment is computed by the minimum edit-distance dynamic program in Appendix B.3.

For $(z_0, z_1) \sim \pi$, the conditional probability path on $\mathcal{X} \times \mathcal{Z}$ is

$$\tilde{p}_t(X, z \mid z_0, z_1) := p_t(z \mid z_0, z_1) \delta_{f_{\text{rm-}\varepsilon}(z)}(X), \tag{19}$$

where $p_t(z \mid z_0, z_1)$ is the token-wise mixture path of Equation (12) applied to the aligned endpoints. A conditional rate generating this path is

$$\begin{aligned} &\tilde{u}_t(X, z \mid X_t, z_t, z_0, z_1) \\ &\quad := \delta_{f_{\text{rm-}\varepsilon}(z)}(X) \sum_{i=1}^{M} \frac{\dot{\kappa}_t}{1 - \kappa_t} \left[\delta_{z_1^i}(z^i) - \delta_{z_t^i}(z^i) \right] \prod_{j \neq i} \delta_{z_t^j}(z^j). \end{aligned} \tag{20}$$

Each transition changes one aligned position and maps to one edit of the sequence $X_t = f_{\text{rm-}\varepsilon}(z_t)$. Applying Theorem A.1, the marginal rate transporting p to q is

$$u_t(X \mid X_t) := \sum_{z} \mathbb{E}_{\tilde{p}_t(z_0, z_1, z_t \mid X_t)} \left[\tilde{u}_t(X, z \mid X_t, z_t, z_0, z_1) \right]. \tag{21}$$

This marginal rate is learned with a Bregman divergence. Choosing $\phi(v) = \sum_{X \neq X_t} v(X) \log v(X)$ and omitting terms independent of θ gives the training objective in Equation (4), with the model rate conditioned on c. Its first term sums all outgoing model rates, and its second term is a weighted cross-entropy over edits that bring X_t closer to X_1. The theorem applies for each fixed condition c, so training with the sampled alignment supervision has the same expected gradient as training against the corresponding marginal rate on $\mathcal{X}$.

B EditFlowSR Implementation Details

This appendix provides the technical details of the EditFlowSR implementation, complementing the method description in Section 4.

B.1 Model Configuration

B.1.1 Point embedding

The numerical input uses a vocabulary of scalar representation tokens. Each token is mapped to a 64-dimensional embedding. With maximum input and output dimensions of 10 and 1, respectively, each sample contains $(2+1)(10+1) = 33$ token positions after padding, so the flattened embedded sample has dimension $33 \times 64 = 2112$. The point embedder applies one linear layer $2112 \to 2112$, followed by a ReLU activation, and a final linear layer $2112 \to 512$. The expansion factor of the hidden layer is one. Thus, each sample is represented by a 512-dimensional point embedding before entering the data encoder.

B.1.2 Time embedding

The editing decoder represents time t with a fixed 512-dimensional sinusoidal time embedding. This vector is passed through a linear layer $512 \to 512$, a SiLU activation, and a second linear layer $512 \to 512$. The resulting 512-dimensional time representation is combined with the editing decoder hidden state at each prediction head.

B.1.3 Data encoder and editing decoder

The data encoder and editing decoder both have width 512 and 16 attention heads, and each Transformer layer contains a feed-forward subnetwork with one hidden layer of width $4 \times 512 = 2048$.

The data encoder receives the 512-dimensional point embeddings and has no positional embeddings, preserving permutation invariance over samples. The data encoder outputs contextualized point embeddings. The contextualized point embeddings form the data condition c.

The editing decoder maps each input token to a 512-dimensional token embedding and adds a learnable 512-dimensional positional embedding, a table of 4096 positions, to form the input representations. This capacity is never approached, as expression sequences in training and inference contain at most a few dozen tokens. Its 8 Transformer layers transform the input representations into 512-dimensional editing decoder hidden states. Each Transformer layer in the editing decoder applies non-causal self-attention across all positions, cross-attention to the contextualized point embeddings in the data condition c, and the feed-forward subnetwork in this order. A residual connection and layer normalization follow each sublayer. For the repair trajectory, the data condition is replaced by a single learned 512-dimensional null-condition token, initialized to zero.

B.1.4 Prediction heads

At each position, the editing decoder hidden state of dimension 512 is combined with the 512-dimensional time representation at the first linear layer of each prediction head. The three heads have the form $1024 \to 512 \to d_{\text{out}}$, with a SiLU activation between the linear layers. The rate head has $d_{\text{out}} = 3$ and outputs the raw rates for insertion, deletion, and substitution. The insertion and substitution heads each have $d_{\text{out}} = |\mathfrak{T}|$, where $|\mathfrak{T}|$ is the vocabulary size, and output the corresponding token logits. The rate outputs are transformed with softplus into the total rates, while the token logits are normalized with softmax in the rate decomposition of (3).

Token embeddings are initialized from a normal distribution with standard deviation $512^{-1/2}$, and padding rows are initialized to zero. The linear layers in the three prediction heads use Xavier uniform initialization with gain 0.1 and zero biases.

B.2 Repair-Trajectory Expansion Rules

The expansion rules for the repair trajectory in Section 4.3 create the following redundancy patterns.

Identity perturbations. Algebraic identities preserve the value while increasing length, such as $f \to f + 0$, $f \to f \times 1$, or $f \to f/1$.

Locally ineffective substructures. Zero-valued or identity-like subexpressions are inserted, such as $f \to f + (x - x)$ or $f \to f + (0 \times x)$. The model must use surrounding context to decide whether a substructure affects the expression value.

Reflexive patterns. Reflexive structures such as $f \to -(-f)$ produce redundant expressions that differ only by reversible local transformations.

Rule composition and length control. Multiple expansion rules are composed in random order to produce longer and more varied redundancy patterns.

B.3 Edit-distance Alignment

In our experiments, the alignment of Section 3 that fixes the training path of Section 4.3 is instantiated with the minimum edit-distance dynamic program described below.

Algorithm 1 EditFlowSR Pretraining

Require: Model parameters θ, trajectory probability ρ, learning rate α

for each training iteration **do**
  $s \sim \text{Bernoulli}(\rho)$
  if $s = 0$ (discovery) **then**
    $\left(f, \{(\mathbf{x}_i, y_i)\}_{i=1}^N\right) \leftarrow \text{SampleData}(), \quad X_0 \leftarrow \text{RandomTokens}(), \quad X_1 \leftarrow \text{Tok}(f), \quad c \leftarrow \text{Enc}\left(\{(\mathbf{x}_i, y_i)\}_{i=1}^N\right)$
  else
    $f \leftarrow \text{SampleExpression}(), \quad X_1 \leftarrow \text{Tok}(f), \quad X_0 \leftarrow \text{AddRedundancy}(X_1), \quad c \leftarrow \emptyset$
  end if
  $(z_0, z_1) \leftarrow \text{Align}(X_0, X_1)$
  $t \sim \mathcal{U}(0,1), \quad z_t \sim p_t(\cdot \mid z_0, z_1)$ of (2), $\quad X_t \leftarrow f_{\text{rm-}\varepsilon}(z_t)$
  $\theta \leftarrow \text{Adam}\left(\theta, \nabla_\theta \mathcal{L}_{\text{traj}}(\theta, c); \alpha\right)$
end for

Given X_0 and X_1, the table $d_{i,j}$ stores the minimum edit distance between the prefixes $X_0^{1:i}$ and $X_1^{1:j}$, with boundary $d_{i,0} = i$, $d_{0,j} = j$ and the recurrence

$$d_{i,j} = \begin{cases} d_{i-1,j-1}, & X_0^i = X_1^j, \\ 1 + \min(d_{i-1,j}, d_{i,j-1}, d_{i-1,j-1}), & \text{otherwise}, \end{cases} \tag{22}$$

so that a match costs zero and an insertion, deletion, or substitution costs one.

Backtracking from $(|X_0|, |X_1|)$ to $(0,0)$ produces the alignment. At each step the diagonal move is taken when the two tokens match or when $d_{i,j} = d_{i-1,j-1} + 1$, which records a substitution; otherwise the move with $d_{i,j} = d_{i-1,j} + 1$ is taken, which records a deletion by aligning X_0^i with the blank token ε, and the remaining moves record insertions by aligning ε with X_1^j. Among the possibly multiple minimum-cost alignments, this backtracking rule keeps as many tokens as possible aligned between the two endpoints. The result is the equal length aligned endpoints $z_0, z_1 \in \mathcal{Z}$ of Section 3, whose length is at most $|X_0| + |X_1|$, and each aligned position carries the corresponding insertion, deletion, or substitution operation used to construct the training path by applying (2) to these aligned endpoints.

B.4 Pretraining Procedure

The pretraining procedure of Section 4.3 is summarized in Algorithm 1, with its configuration reported in Table 4. Training data are generated from the distribution of Section 4.1.

The initial input length ℓ_0 is drawn uniformly from $\{10, \ldots, 40\}$. The random token source X_0 consists of $\ell_0 - 1$ tokens drawn independently and uniformly from the equation vocabulary, with `BOS` prepended in the model input.

Here, SampleData draws equations and their corresponding data samples from the distribution of Section 4.1, Tok converts an expression into its token sequence, Enc maps the samples to the data condition c using the point embedder of Section 4.1 and the data encoder of Section 4.2, RandomTokens draws the random token sequence described above, SampleExpression draws a concise expression from the same generator, AddRedundancy expands the target expression X_1 into an algebraically redundant source X_0 using the rules in Appendix B.2, and Align computes the edit-distance alignment of Appendix B.3. The loss $\mathcal{L}_{\text{traj}}$ is the trajectory objective of (4).

EditFlowSR is implemented in PyTorch 2.10. EditFlowSR pretraining, EditFlowSR inference, and all baseline evaluations were conducted on the same machine, an Intel Xeon Gold 6226R CPU with 2×16 cores and 64 threads, 128 GB of memory, and NVIDIA RTX 3090 GPUs with 24 GB memory each. EditFlowSR pretrains on three of the RTX 3090 GPUs with DDP and FP16 mixed precision, generating data online; the effective batch size is 336 (112 per GPU), and the run completes 195 epochs in about 18 wall-clock hours, for a total of 65,520,000 online-generated training samples.

B.5 Inference Procedure

The inference procedure of Section 4.4 is summarized in Algorithm 2.

Table 4: EditFlowSR pretraining configuration.

Optimization	
Optimizer	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, smoothing constant 10^{-8}, no weight decay)
Peak learning rate α	4×10^{-5}
Learning rate schedule	linear warmup over the first 10,000 steps, then decay proportional to $\text{step}^{-1/2}$
Gradient clipping	global norm 0.5
Batch and precision	
Number of GPUs	3
Per-GPU batch size	112
Effective batch size	336
Precision	FP16 mixed precision
Training	
Epochs	195
Training samples	65,520,000 (generated online)
Trajectory	
Trajectory probability ρ	0.2

Algorithm 2 Iterative Edit Inference

Require: Samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, number of sampling steps n_{steps}

$c \leftarrow \text{Enc}(\{(\mathbf{x}_i, y_i)\}_{i=1}^N)$
$X_0 \leftarrow \text{RandomTokens}()$
$t \leftarrow 0$, $\Delta t \leftarrow 1/n_{\text{steps}}$
while $t < 1$ **do**
  $\{\lambda_{t,i}^{\text{ins}}, \lambda_{t,i}^{\text{sub}}, \lambda_{t,i}^{\text{del}}, Q_{t,i}^{\text{ins}}, Q_{t,i}^{\text{sub}}\}_i \leftarrow \text{Decoder}(X_t, c, t)$
  for each insertion anchor $i = 0, \ldots, |X_t|$ **do**
    Sample insertion trigger $b_{i,\text{ins}} \sim \text{Bernoulli}(\Delta t\, \lambda_{t,i}^{\text{ins}})$
    if $b_{i,\text{ins}} = 1$ **then**
      Sample inserted token $a_{i,\text{ins}} \sim Q_{t,i}^{\text{ins}}$
    end if
  end for
  for each token position $i = 1, \ldots, |X_t|$ **do**
    Sample deletion-or-substitution trigger $b_{i,\text{ds}} \sim \text{Bernoulli}\big(\Delta t\, (\lambda_{t,i}^{\text{sub}} + \lambda_{t,i}^{\text{del}})\big)$
    if $b_{i,\text{ds}} = 1$ **then**
      Sample $v_i \sim \text{Bernoulli}\big(\lambda_{t,i}^{\text{del}} / (\lambda_{t,i}^{\text{sub}} + \lambda_{t,i}^{\text{del}})\big)$
      if $v_i = 1$ **then**
        Mark a deletion at position i
      else
        Sample replacement token $a_{i,\text{sub}} \sim Q_{t,i}^{\text{sub}}$
      end if
    end if
  end for
  Apply substitutions first
  Merge positions where both insertion and deletion are triggered into replacements
  Apply the remaining insertions and deletions in one pass using cumulative position shifts to obtain $X_{t+\Delta t}$
  $t \leftarrow t + \Delta t$
end while
$f_{\text{raw}} \leftarrow \text{PrefixToExpr}(X_1)$
$f_{\text{final}} \leftarrow \text{BFGS_Refine}(f_{\text{raw}}, \{(\mathbf{x}_i, y_i)\}_{i=1}^N)$
return f_{final}

Inference uses $n_{\text{steps}} = 100$ sampling steps and draws the initial input length ℓ_0 uniformly from $\{10, \ldots, 40\}$, as in pretraining. Appendix D.3 reports the sensitivity to these hyperparameters.

C Experimental Details

C.1 Baselines

The generation-based symbolic regression baselines comprise NeSymReS [Biggio et al., 2021] and E2E [Kamienny et al., 2022]. Both rely on synthetic data and the power of large-scale pretrained priors to generate equations in a single forward pass, leading to considerably faster inference time compared to the search-based methods. TPSR [Shojaee et al., 2023] further incorporates external search into the decoding of these generative models.

The search-based symbolic regression baselines comprise DSR [Petersen et al., 2021], the evolutionary methods GPlearn [Stephens, 2015], GP-GOMEA [Virgolin et al., 2021], Operon [Burlacu et al., 2020], MRGP [Arnaldo et al., 2014], ITEA [de França and Aldeia, 2021], and SBP-GP [Virgolin et al., 2019], and the neural-guided search methods SNIP [Meidani et al., 2024], GenSR [Li et al., 2026], and SR4MDL [Yu et al., 2025]. These methods iteratively update candidate expressions via mutation and crossover operations, risk-seeking policy gradients, or neural-guided search that enhances search efficiency.

Large language models are also incorporated into the search process: LLM-SR [Shojaee et al., 2025] performs evolutionary search over LLM-proposed equation skeletons, RESTART [Li and Pan, 2026] refines candidates in an analyze loop with a reusable structure library, and SR-SCIENTIST [Xia et al., 2026] applies autonomous agentic optimization with code-interpreter tools. Shojaee et al. [Shojaee et al., 2025] have shown that LLMs may exploit memorized equations in standard SR benchmarks rather than discover them from data. Since the Feynman and Strogatz datasets are drawn from established published collections, the LLM-based baselines are evaluated only on the Black-box dataset, where the unknown functions must be inferred from observations alone.

FFX [McConaghy, 2011] provides a deterministic sparse regression baseline. The conventional non-symbolic baselines comprise XGBoost [Chen and Guestrin, 2016], LightGBM [Ke et al., 2017], AdaBoost [Freund and Schapire, 1997], Random Forest [Breiman, 2001], and kernel ridge regression [Saunders et al., 1998], which provide accuracy references without an explicit symbolic expression.

All baselines are evaluated with the hyperparameters and operation settings recommended by their official repositories, following the best configurations publicly released by the respective authors. In particular, constant optimization is enabled for every expression-based baseline under the BFGS setting specified by its paper and official repository. EditFlowSR uses the same BFGS constant optimization settings as E2E. NeSymReS [Biggio et al., 2021], E2E [Kamienny et al., 2022], SNIP [Meidani et al., 2024], TPSR [Shojaee et al., 2023], GenSR [Li et al., 2026], and SR4MDL [Yu et al., 2025] are run with the publicly released pretrained weights from the respective authors. All baseline evaluations use the same hardware configuration, with details in Appendix B.4.

C.2 Noise Injection

All datasets are evaluated under four noise levels $\sigma \in \{0, 0.001, 0.01, 0.1\}$. For each task, 75% of the data points form the fit portion supplied to the method for expression generation and constant optimization, while the remaining 25% form the held-out portion used to compute test R^2. For $\sigma > 0$, multiplicative Gaussian noise is injected into the target values of the fit data supplied to the method, $\tilde{y}_i = y_i(1 + \xi_i)$ with $\xi_i \sim \mathcal{N}(0, \sigma^2)$; the inputs are unchanged, and the held-out inputs and targets remain clean for evaluation. The noise is sampled independently for each dataset, and the perturbation magnitude is proportional to the target value.

C.3 Complete Numerical Results

Full results for all methods on the Feynman, Strogatz, and Black-box datasets are given below. Each method has three rows: R^2, inference time (seconds), and expression complexity. Shojaee et al. [Shojaee et al., 2025] have shown that LLMs may exploit memorized equations in standard SR benchmarks rather than discover them from data. Since the Feynman and Strogatz datasets are drawn

from established published collections, the LLM-based baselines are evaluated only on the Black-box dataset, where the unknown functions must be inferred from observations alone.

Table 5: Performance on the Feynman dataset. σ denotes the noise level. Values are mean $\pm$ standard deviation across the evaluated datasets. Unavailable expression complexity values are marked as —.

Method	Metric	$\sigma = 0$	$\sigma = 0.001$	$\sigma = 0.01$	$\sigma = 0.1$
E2E	$R^2 \uparrow$	$0.903_{\pm 0.23}$	$0.917_{\pm 0.21}$	$0.900_{\pm 0.23}$	$0.852_{\pm 0.21}$
	Time/s $\downarrow$	$8.9_{\pm 2.9}$	$13.0_{\pm 4.7}$	$13.4_{\pm 3.8}$	$14.0_{\pm 4.0}$
	Compl. $\downarrow$	$59.7_{\pm 25.1}$	$61.4_{\pm 26.1}$	$65.0_{\pm 21.8}$	$82.5_{\pm 21.0}$
NeSymReS	$R^2 \uparrow$	$0.681_{\pm 0.35}$	$0.684_{\pm 0.34}$	$0.681_{\pm 0.35}$	$0.661_{\pm 0.32}$
	Time/s $\downarrow$	$28.9_{\pm 13.6}$	$38.3_{\pm 20.8}$	$38.3_{\pm 21.1}$	$41.4_{\pm 20.5}$
	Compl. $\downarrow$	$6.4_{\pm 2.5}$	$6.7_{\pm 2.2}$	$6.7_{\pm 2.3}$	$7.2_{\pm 2.2}$
SNIP	$R^2 \uparrow$	$0.934_{\pm 0.17}$	$0.944_{\pm 0.14}$	$0.934_{\pm 0.18}$	$0.925_{\pm 0.18}$
	Time/s $\downarrow$	$95.3_{\pm 158.5}$	$96.6_{\pm 200.2}$	$86.3_{\pm 115.4}$	$172.7_{\pm 294.6}$
	Compl. $\downarrow$	$29.5_{\pm 8.0}$	$29.1_{\pm 7.3}$	$29.4_{\pm 6.3}$	$32.5_{\pm 8.2}$
SR4MDL	$R^2 \uparrow$	$0.942_{\pm 0.13}$	$0.942_{\pm 0.13}$	$0.907_{\pm 0.22}$	$0.848_{\pm 0.25}$
	Time/s $\downarrow$	$377.2_{\pm 322.3}$	$344.4_{\pm 273.5}$	$522.8_{\pm 114.5}$	$524.5_{\pm 124.1}$
	Compl. $\downarrow$	$37.1_{\pm 21.6}$	$42.9_{\pm 17.9}$	$53.4_{\pm 10.4}$	$53.5_{\pm 10.5}$
TPSR	$R^2 \uparrow$	$0.971_{\pm 0.15}$	$0.962_{\pm 0.17}$	$0.971_{\pm 0.14}$	$0.891_{\pm 0.27}$
	Time/s $\downarrow$	$135.1_{\pm 111.0}$	$167.2_{\pm 130.7}$	$186.9_{\pm 127.5}$	$228.4_{\pm 110.5}$
	Compl. $\downarrow$	$58.0_{\pm 26.9}$	$56.9_{\pm 25.9}$	$59.8_{\pm 24.1}$	$67.2_{\pm 21.6}$
GenSR	$R^2 \uparrow$	$0.986_{\pm 0.02}$	$0.985_{\pm 0.02}$	$0.984_{\pm 0.02}$	$0.982_{\pm 0.03}$
	Time/s $\downarrow$	$301.0_{\pm 348.9}$	$281.7_{\pm 359.0}$	$286.3_{\pm 311.1}$	$537.6_{\pm 322.6}$
	Compl. $\downarrow$	$23.9_{\pm 7.3}$	$23.6_{\pm 6.9}$	$23.5_{\pm 7.1}$	$21.9_{\pm 7.8}$
FFX	$R^2 \uparrow$	$0.624_{\pm 0.43}$	$0.633_{\pm 0.43}$	$0.657_{\pm 0.42}$	$0.606_{\pm 0.43}$
	Time/s $\downarrow$	$19.6_{\pm 9.0}$	$19.6_{\pm 8.4}$	$19.9_{\pm 9.0}$	$21.2_{\pm 8.3}$
	Compl. $\downarrow$	$759.8_{\pm 716.0}$	$771.7_{\pm 733.1}$	$736.0_{\pm 710.1}$	$1008.0_{\pm 636.2}$
GP-GOMEA	$R^2 \uparrow$	$0.959_{\pm 0.16}$	$0.957_{\pm 0.16}$	$0.962_{\pm 0.15}$	$0.894_{\pm 0.18}$
	Time/s $\downarrow$	$872.7_{\pm 111.0}$	$2496.9_{\pm 271.0}$	$2773.7_{\pm 410.4}$	$2126.8_{\pm 260.2}$
	Compl. $\downarrow$	$23.0_{\pm 5.2}$	$25.6_{\pm 4.5}$	$25.8_{\pm 4.2}$	$26.6_{\pm 3.5}$
Operon	$R^2 \uparrow$	$0.973_{\pm 0.104}$	$0.966_{\pm 0.132}$	$0.972_{\pm 0.105}$	$0.974_{\pm 0.097}$
	Time/s $\downarrow$	$6791.7_{\pm 602.6}$	$6761.2_{\pm 655.7}$	$6728.7_{\pm 632.7}$	$6669.7_{\pm 771.2}$
	Compl. $\downarrow$	$53.4_{\pm 10.5}$	$53.3_{\pm 10.1}$	$53.8_{\pm 10.2}$	$55.8_{\pm 5.1}$
GPlearn	$R^2 \uparrow$	$0.668_{\pm 0.31}$	$0.666_{\pm 0.31}$	$0.671_{\pm 0.32}$	$0.671_{\pm 0.31}$
	Time/s $\downarrow$	$364.6_{\pm 62.3}$	$364.2_{\pm 62.2}$	$366.1_{\pm 61.8}$	$364.7_{\pm 63.3}$
	Compl. $\downarrow$	$20.8_{\pm 17.2}$	$20.3_{\pm 15.2}$	$20.1_{\pm 14.2}$	$22.9_{\pm 19.0}$
MRGP	$R^2 \uparrow$	$0.908_{\pm 0.16}$	$0.908_{\pm 0.16}$	$0.904_{\pm 0.17}$	$0.904_{\pm 0.17}$
	Time/s $\downarrow$	$142.3_{\pm 27.6}$	$141.2_{\pm 28.3}$	$128.0_{\pm 34.3}$	$144.7_{\pm 29.8}$
	Compl. $\downarrow$	$42.3_{\pm 18.8}$	$38.8_{\pm 17.7}$	$42.2_{\pm 19.7}$	$40.9_{\pm 16.6}$
SBP-GP	$R^2 \uparrow$	$0.948_{\pm 0.21}$	$0.964_{\pm 0.16}$	$0.958_{\pm 0.17}$	$0.771_{\pm 0.32}$
	Time/s $\downarrow$	$2361.2_{\pm 390.1}$	$880.0_{\pm 90.9}$	$2971.3_{\pm 380.6}$	$1349.6_{\pm 160.8}$
	Compl. $\downarrow$	$327.4_{\pm 139.1}$	$406.1_{\pm 93.6}$	$427.0_{\pm 100.4}$	$446.2_{\pm 73.0}$
DSR	$R^2 \uparrow$	$0.628_{\pm 0.239}$	$0.630_{\pm 0.241}$	$0.624_{\pm 0.246}$	$0.627_{\pm 0.244}$
	Time/s $\downarrow$	$3277.6_{\pm 1789.5}$	$3296.3_{\pm 1795.2}$	$3281.7_{\pm 1812.9}$	$3283.4_{\pm 1765.7}$
	Compl. $\downarrow$	$11.7_{\pm 4.5}$	$12.3_{\pm 4.4}$	$12.6_{\pm 4.9}$	$13.1_{\pm 6.0}$
ITEA	$R^2 \uparrow$	$0.913_{\pm 0.166}$	$0.914_{\pm 0.166}$	$0.913_{\pm 0.169}$	$0.913_{\pm 0.169}$
	Time/s $\downarrow$	$2404.6_{\pm 668.6}$	$2419.1_{\pm 663.6}$	$2430.1_{\pm 663.1}$	$2437.9_{\pm 674.4}$
	Compl. $\downarrow$	$17.2_{\pm 6.2}$	$17.3_{\pm 6.1}$	$17.3_{\pm 5.9}$	$17.5_{\pm 6.2}$
AdaBoost	$R^2 \uparrow$	$0.691_{\pm 0.27}$	$0.695_{\pm 0.27}$	$0.704_{\pm 0.26}$	$0.710_{\pm 0.24}$

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
	Time/s ↓	$0.16_{\pm 0.056}$	$0.16_{\pm 0.052}$	$0.17_{\pm 0.057}$	$0.15_{\pm 0.049}$
	Compl. ↓	—	—	—	—
KernelRidge	R^2 ↑	$0.053_{\pm 0.15}$	$0.053_{\pm 0.15}$	$0.053_{\pm 0.15}$	$0.053_{\pm 0.15}$
	Time/s ↓	$0.028_{\pm 0.052}$	$0.027_{\pm 0.038}$	$0.026_{\pm 0.035}$	$0.021_{\pm 0.033}$
	Compl. ↓	—	—	—	—
LGBM	R^2 ↑	$0.802_{\pm 0.22}$	$0.802_{\pm 0.22}$	$0.803_{\pm 0.22}$	$0.799_{\pm 0.22}$
	Time/s ↓	$38.5_{\pm 9.0}$	$38.0_{\pm 9.2}$	$37.4_{\pm 8.8}$	$35.8_{\pm 9.1}$
	Compl. ↓	—	—	—	—
RandomForest	R^2 ↑	$0.774_{\pm 0.25}$	$0.773_{\pm 0.25}$	$0.773_{\pm 0.26}$	$0.771_{\pm 0.26}$
	Time/s ↓	$0.29_{\pm 0.092}$	$0.30_{\pm 0.096}$	$0.29_{\pm 0.084}$	$0.28_{\pm 0.086}$
	Compl. ↓	—	—	—	—
XGB	R^2 ↑	$0.762_{\pm 0.27}$	$0.763_{\pm 0.27}$	$0.761_{\pm 0.27}$	$0.762_{\pm 0.27}$
	Time/s ↓	$0.31_{\pm 0.064}$	$0.32_{\pm 0.056}$	$0.30_{\pm 0.065}$	$0.30_{\pm 0.075}$
	Compl. ↓	—	—	—	—
EditFlowSR (Ours)	R^2 ↑	$0.903_{\pm 0.16}$	$0.898_{\pm 0.17}$	$0.902_{\pm 0.16}$	$0.877_{\pm 0.16}$
	Time/s ↓	$37.1_{\pm 57.8}$	$36.2_{\pm 54.9}$	$34.4_{\pm 48.1}$	$51.2_{\pm 64.8}$
	Compl. ↓	$27.9_{\pm 8.6}$	$27.3_{\pm 7.8}$	$27.7_{\pm 8.5}$	$28.0_{\pm 7.8}$

Table 6: Performance on the Strogatz dataset. σ denotes the noise level. Values are mean $\pm$ standard deviation across the evaluated datasets. Unavailable expression complexity values are marked as —.

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
E2E	R^2 ↑	$0.600_{\pm 0.34}$	$0.639_{\pm 0.32}$	$0.550_{\pm 0.40}$	$0.584_{\pm 0.35}$
	Time/s ↓	$7.5_{\pm 2.5}$	$9.2_{\pm 4.0}$	$7.2_{\pm 2.3}$	$8.9_{\pm 3.4}$
	Compl. ↓	$57.9_{\pm 19.4}$	$54.9_{\pm 20.2}$	$60.6_{\pm 17.7}$	$67.6_{\pm 18.4}$
NeSymReS	R^2 ↑	$0.858_{\pm 0.16}$	$0.837_{\pm 0.17}$	$0.833_{\pm 0.17}$	$0.811_{\pm 0.16}$
	Time/s ↓	$31.2_{\pm 12.5}$	$34.9_{\pm 18.1}$	$33.7_{\pm 14.2}$	$35.9_{\pm 15.8}$
	Compl. ↓	$5.6_{\pm 2.0}$	$5.8_{\pm 1.7}$	$5.5_{\pm 1.2}$	$6.4_{\pm 2.0}$
SNIP	R^2 ↑	$0.934_{\pm 0.13}$	$0.947_{\pm 0.06}$	$0.909_{\pm 0.25}$	$0.954_{\pm 0.05}$
	Time/s ↓	$47.2_{\pm 44.3}$	$45.5_{\pm 39.7}$	$47.4_{\pm 34.1}$	$108.2_{\pm 193.3}$
	Compl. ↓	$25.8_{\pm 7.4}$	$25.4_{\pm 6.2}$	$27.7_{\pm 8.0}$	$30.1_{\pm 9.9}$
SR4MDL	R^2 ↑	$0.940_{\pm 0.20}$	$0.876_{\pm 0.27}$	$0.874_{\pm 0.27}$	$0.863_{\pm 0.28}$
	Time/s ↓	$207.0_{\pm 207.1}$	$233.7_{\pm 204.7}$	$437.7_{\pm 28.4}$	$439.0_{\pm 29.8}$
	Compl. ↓	$16.9_{\pm 9.3}$	$20.3_{\pm 11.7}$	$23.5_{\pm 16.1}$	$23.5_{\pm 16.1}$
TPSR	R^2 ↑	$0.920_{\pm 0.26}$	$0.986_{\pm 0.04}$	$0.991_{\pm 0.02}$	$0.944_{\pm 0.05}$
	Time/s ↓	$154.8_{\pm 92.2}$	$193.8_{\pm 126.9}$	$204.1_{\pm 110.8}$	$262.3_{\pm 128.4}$
	Compl. ↓	$52.1_{\pm 20.9}$	$59.6_{\pm 23.6}$	$59.9_{\pm 20.6}$	$61.1_{\pm 18.2}$
GenSR	R^2 ↑	$0.934_{\pm 0.16}$	$0.923_{\pm 0.24}$	$0.925_{\pm 0.24}$	$0.960_{\pm 0.09}$
	Time/s ↓	$67.1_{\pm 90.8}$	$75.5_{\pm 123.5}$	$77.0_{\pm 90.2}$	$335.5_{\pm 224.9}$
	Compl. ↓	$20.4_{\pm 7.9}$	$20.6_{\pm 8.4}$	$20.9_{\pm 8.4}$	$20.7_{\pm 7.5}$
FFX	R^2 ↑	$0.932_{\pm 0.11}$	$0.915_{\pm 0.12}$	$0.936_{\pm 0.10}$	$0.947_{\pm 0.10}$
	Time/s ↓	$11.7_{\pm 4.6}$	$10.8_{\pm 3.0}$	$9.7_{\pm 2.6}$	$14.0_{\pm 4.0}$
	Compl. ↓	$182.5_{\pm 204.0}$	$181.8_{\pm 209.7}$	$199.7_{\pm 214.8}$	$456.4_{\pm 212.0}$
GP-GOMEA	R^2 ↑	$0.888_{\pm 0.26}$	$0.918_{\pm 0.23}$	$0.923_{\pm 0.20}$	$0.912_{\pm 0.19}$
	Time/s ↓	$1303.1_{\pm 190.4}$	$1638.7_{\pm 230.7}$	$822.8_{\pm 70.7}$	$1348.8_{\pm 180.4}$
	Compl. ↓	$22.1_{\pm 5.4}$	$23.4_{\pm 5.2}$	$24.1_{\pm 5.0}$	$24.1_{\pm 4.1}$
Operon	R^2 ↑	$0.919_{\pm 0.257}$	$0.958_{\pm 0.123}$	$0.932_{\pm 0.171}$	$0.808_{\pm 0.359}$
	Time/s ↓	$23.2_{\pm 9.6}$	$24.0_{\pm 6.7}$	$25.8_{\pm 7.2}$	$28.1_{\pm 8.7}$

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
	Compl. ↓	$48.4_{\pm 13.6}$	$48.4_{\pm 12.0}$	$56.1_{\pm 4.8}$	$57.5_{\pm 4.9}$
GPlearn	R^2 ↑	$0.723_{\pm 0.33}$	$0.738_{\pm 0.33}$	$0.692_{\pm 0.38}$	$0.726_{\pm 0.32}$
	Time/s ↓	$397.2_{\pm 88.9}$	$392.4_{\pm 62.7}$	$417.2_{\pm 95.4}$	$365.1_{\pm 45.9}$
	Compl. ↓	$31.4_{\pm 25.4}$	$25.9_{\pm 18.4}$	$27.1_{\pm 19.8}$	$22.0_{\pm 14.7}$
MRGP	R^2 ↑	$0.983_{\pm 0.03}$	$0.971_{\pm 0.05}$	$0.986_{\pm 0.03}$	$0.945_{\pm 0.11}$
	Time/s ↓	$49.2_{\pm 65.9}$	$53.7_{\pm 91.6}$	$45.1_{\pm 67.9}$	$65.8_{\pm 88.8}$
	Compl. ↓	$41.9_{\pm 20.2}$	$34.0_{\pm 18.1}$	$38.1_{\pm 14.4}$	$47.8_{\pm 13.3}$
SBP-GP	R^2 ↑	$0.927_{\pm 0.26}$	$0.917_{\pm 0.26}$	$0.848_{\pm 0.33}$	$0.862_{\pm 0.25}$
	Time/s ↓	$869.4_{\pm 130.5}$	$1342.3_{\pm 220.1}$	$1016.8_{\pm 40.0}$	$2597.7_{\pm 390.7}$
	Compl. ↓	$360.3_{\pm 136.3}$	$414.6_{\pm 80.3}$	$426.6_{\pm 70.5}$	$452.1_{\pm 81.8}$
DSR	R^2 ↑	$0.616_{\pm 0.336}$	$0.662_{\pm 0.338}$	$0.654_{\pm 0.325}$	$0.636_{\pm 0.343}$
	Time/s ↓	$587.5_{\pm 276.6}$	$664.0_{\pm 235.6}$	$644.3_{\pm 220.7}$	$636.4_{\pm 228.6}$
	Compl. ↓	$15.6_{\pm 9.1}$	$15.1_{\pm 9.3}$	$12.9_{\pm 6.6}$	$13.7_{\pm 6.8}$
ITEA	R^2 ↑	$0.783_{\pm 0.271}$	$0.784_{\pm 0.267}$	$0.783_{\pm 0.271}$	$0.799_{\pm 0.256}$
	Time/s ↓	$27.4_{\pm 7.5}$	$27.6_{\pm 7.0}$	$26.6_{\pm 7.1}$	$29.1_{\pm 7.5}$
	Compl. ↓	$11.1_{\pm 1.8}$	$11.0_{\pm 1.8}$	$11.0_{\pm 1.8}$	$11.5_{\pm 2.3}$
AdaBoost	R^2 ↑	$0.753_{\pm 0.35}$	$0.770_{\pm 0.34}$	$0.792_{\pm 0.29}$	$0.810_{\pm 0.28}$
	Time/s ↓	$0.088_{\pm 0.076}$	$0.11_{\pm 0.075}$	$0.087_{\pm 0.041}$	$0.083_{\pm 0.042}$
	Compl. ↓	—	—	—	—
KernelRidge	R^2 ↑	$0.233_{\pm 0.34}$	$0.233_{\pm 0.34}$	$0.233_{\pm 0.34}$	$0.235_{\pm 0.34}$
	Time/s ↓	$0.022_{\pm 0.031}$	$0.052_{\pm 0.061}$	$0.058_{\pm 0.087}$	$0.031_{\pm 0.052}$
	Compl. ↓	—	—	—	—
LGBM	R^2 ↑	$0.740_{\pm 0.35}$	$0.739_{\pm 0.35}$	$0.739_{\pm 0.35}$	$0.730_{\pm 0.36}$
	Time/s ↓	$38.0_{\pm 7.1}$	$38.0_{\pm 9.1}$	$39.4_{\pm 9.7}$	$40.1_{\pm 9.1}$
	Compl. ↓	—	—	—	—
RandomForest	R^2 ↑	$0.980_{\pm 0.03}$	$0.980_{\pm 0.03}$	$0.979_{\pm 0.03}$	$0.887_{\pm 0.21}$
	Time/s ↓	$0.28_{\pm 0.068}$	$0.27_{\pm 0.069}$	$0.27_{\pm 0.075}$	$0.24_{\pm 0.065}$
	Compl. ↓	—	—	—	—
XGB	R^2 ↑	$0.972_{\pm 0.03}$	$0.971_{\pm 0.03}$	$0.971_{\pm 0.03}$	$0.825_{\pm 0.32}$
	Time/s ↓	$0.30_{\pm 0.015}$	$0.28_{\pm 0.027}$	$0.28_{\pm 0.040}$	$0.29_{\pm 0.050}$
	Compl. ↓	—	—	—	—
EditFlowSR (Ours)	R^2 ↑	$0.952_{\pm 0.06}$	$0.955_{\pm 0.09}$	$0.942_{\pm 0.08}$	$0.934_{\pm 0.09}$
	Time/s ↓	$33.6_{\pm 43.4}$	$17.5_{\pm 13.7}$	$40.6_{\pm 54.1}$	$43.1_{\pm 59.6}$
	Compl. ↓	$19.9_{\pm 5.2}$	$21.9_{\pm 6.0}$	$21.9_{\pm 5.8}$	$22.9_{\pm 5.9}$

Table 7: Performance on the Black-box dataset. σ denotes the noise level. Values are mean $\pm$ standard deviation across the evaluated datasets. Unavailable expression complexity values are marked as —.

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
E2E	R^2 ↑	$0.441_{\pm 0.33}$	$0.484_{\pm 0.30}$	$0.481_{\pm 0.29}$	$0.390_{\pm 0.31}$
	Time/s ↓	$7.6_{\pm 2.3}$	$15.0_{\pm 6.1}$	$14.7_{\pm 6.7}$	$14.7_{\pm 6.1}$
	Compl. ↓	$93.5_{\pm 33.9}$	$95.7_{\pm 38.4}$	$93.7_{\pm 32.9}$	$95.5_{\pm 36.9}$
NeSymReS	R^2 ↑	$0.168_{\pm 0.28}$	$0.173_{\pm 0.29}$	$0.175_{\pm 0.29}$	$0.151_{\pm 0.25}$
	Time/s ↓	$24.7_{\pm 11.4}$	$36.3_{\pm 25.2}$	$36.2_{\pm 25.0}$	$36.1_{\pm 21.3}$
	Compl. ↓	$7.8_{\pm 1.6}$	$7.6_{\pm 1.7}$	$7.6_{\pm 1.7}$	$7.8_{\pm 1.6}$
SNIP	R^2 ↑	$0.670_{\pm 0.30}$	$0.690_{\pm 0.30}$	$0.685_{\pm 0.28}$	$0.618_{\pm 0.32}$
	Time/s ↓	$989.1_{\pm 856.9}$	$864.9_{\pm 771.1}$	$888.3_{\pm 863.8}$	$1244.2_{\pm 1309.6}$

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
	Compl. ↓	$40.2_{\pm 11.2}$	$40.0_{\pm 10.8}$	$41.0_{\pm 12.4}$	$42.6_{\pm 12.4}$
SR4MDL	R^2 ↑	$0.556_{\pm 0.31}$	$0.558_{\pm 0.32}$	$0.546_{\pm 0.31}$	$0.507_{\pm 0.30}$
	Time/s ↓	$582.5_{\pm 161.9}$	$595.7_{\pm 186.5}$	$590.1_{\pm 179.7}$	$583.1_{\pm 196.3}$
	Compl. ↓	$46.3_{\pm 12.5}$	$46.2_{\pm 12.6}$	$47.3_{\pm 12.0}$	$45.5_{\pm 12.0}$
TPSR	R^2 ↑	$0.713_{\pm 0.34}$	$0.489_{\pm 0.43}$	$0.482_{\pm 0.43}$	$0.430_{\pm 0.42}$
	Time/s ↓	$299.8_{\pm 183.2}$	$407.9_{\pm 270.5}$	$414.8_{\pm 253.2}$	$407.6_{\pm 239.4}$
	Compl. ↓	$82.3_{\pm 34.7}$	$82.2_{\pm 33.2}$	$84.3_{\pm 33.3}$	$82.3_{\pm 29.8}$
GenSR	R^2 ↑	$0.486_{\pm 0.43}$	$0.507_{\pm 0.43}$	$0.496_{\pm 0.43}$	$0.445_{\pm 0.42}$
	Time/s ↓	$1683.1_{\pm 2320.4}$	$1609.5_{\pm 2204.2}$	$1534.9_{\pm 2391.4}$	$1715.6_{\pm 2810.2}$
	Compl. ↓	$34.4_{\pm 12.0}$	$34.0_{\pm 13.0}$	$34.4_{\pm 14.0}$	$36.7_{\pm 13.9}$
FFX	R^2 ↑	$0.377_{\pm 0.37}$	$0.379_{\pm 0.38}$	$0.378_{\pm 0.38}$	$0.318_{\pm 0.37}$
	Time/s ↓	$21.5_{\pm 17.8}$	$29.5_{\pm 25.6}$	$21.9_{\pm 18.2}$	$24.0_{\pm 20.7}$
	Compl. ↓	$1254.0_{\pm 725.4}$	$1250.0_{\pm 706.3}$	$1238.8_{\pm 688.4}$	$1308.3_{\pm 702.0}$
GP-GOMEA	R^2 ↑	$0.688_{\pm 0.27}$	$0.685_{\pm 0.26}$	$0.682_{\pm 0.27}$	$0.608_{\pm 0.29}$
	Time/s ↓	$2375.3_{\pm 270.8}$	$1180.7_{\pm 91.0}$	$2751.8_{\pm 220.3}$	$2230.2_{\pm 270.8}$
	Compl. ↓	$23.8_{\pm 3.8}$	$23.4_{\pm 4.4}$	$23.7_{\pm 4.4}$	$23.5_{\pm 4.3}$
Operon	R^2 ↑	$0.721_{\pm 0.333}$	$0.720_{\pm 0.332}$	$0.732_{\pm 0.336}$	$0.730_{\pm 0.307}$
	Time/s ↓	$556.8_{\pm 1576.0}$	$555.2_{\pm 1573.1}$	$565.1_{\pm 1592.5}$	$557.8_{\pm 1571.9}$
	Compl. ↓	$56.7_{\pm 5.5}$	$57.0_{\pm 7.0}$	$57.4_{\pm 5.7}$	$57.0_{\pm 6.3}$
GPlearn	R^2 ↑	$0.507_{\pm 0.28}$	$0.513_{\pm 0.28}$	$0.502_{\pm 0.29}$	$0.496_{\pm 0.28}$
	Time/s ↓	$376.1_{\pm 72.2}$	$375.9_{\pm 65.0}$	$373.2_{\pm 67.1}$	$369.0_{\pm 62.2}$
	Compl. ↓	$19.9_{\pm 15.1}$	$19.7_{\pm 15.3}$	$19.3_{\pm 15.7}$	$19.3_{\pm 14.0}$
MRGP	R^2 ↑	$0.651_{\pm 0.32}$	$0.622_{\pm 0.33}$	$0.646_{\pm 0.31}$	$0.653_{\pm 0.30}$
	Time/s ↓	$62.3_{\pm 255.1}$	$49.0_{\pm 195.8}$	$46.6_{\pm 180.7}$	$48.0_{\pm 181.9}$
	Compl. ↓	$51.2_{\pm 30.6}$	$53.1_{\pm 36.2}$	$50.3_{\pm 29.8}$	$55.9_{\pm 35.2}$
SBP-GP	R^2 ↑	$0.659_{\pm 0.36}$	$0.640_{\pm 0.37}$	$0.613_{\pm 0.37}$	$0.518_{\pm 0.38}$
	Time/s ↓	$1474.4_{\pm 440.3}$	$2034.4_{\pm 290.3}$	$1828.8_{\pm 610.5}$	$2447.4_{\pm 340.6}$
	Compl. ↓	$419.0_{\pm 80.5}$	$434.9_{\pm 77.4}$	$422.3_{\pm 78.2}$	$428.8_{\pm 79.4}$
DSR	R^2 ↑	$0.578_{\pm 0.233}$	$0.585_{\pm 0.225}$	$0.573_{\pm 0.238}$	$0.576_{\pm 0.229}$
	Time/s ↓	$1596.9_{\pm 4354.6}$	$1378.1_{\pm 3388.9}$	$1345.7_{\pm 3179.8}$	$1445.6_{\pm 3999.3}$
	Compl. ↓	$12.2_{\pm 6.8}$	$12.4_{\pm 6.9}$	$12.6_{\pm 6.9}$	$13.1_{\pm 7.2}$
ITEA	R^2 ↑	$0.483_{\pm 0.308}$	$0.486_{\pm 0.309}$	$0.481_{\pm 0.305}$	$0.467_{\pm 0.292}$
	Time/s ↓	$532.1_{\pm 3341.4}$	$536.2_{\pm 3384.8}$	$571.5_{\pm 3608.6}$	$559.2_{\pm 3571.8}$
	Compl. ↓	$15.8_{\pm 10.8}$	$15.5_{\pm 10.9}$	$15.9_{\pm 10.9}$	$14.9_{\pm 9.5}$
AdaBoost	R^2 ↑	$0.647_{\pm 0.23}$	$0.639_{\pm 0.23}$	$0.646_{\pm 0.24}$	$0.643_{\pm 0.24}$
	Time/s ↓	$0.064_{\pm 0.028}$	$0.094_{\pm 0.059}$	$0.065_{\pm 0.028}$	$0.067_{\pm 0.034}$
	Compl. ↓	—	—	—	—
KernelRidge	R^2 ↑	$0.166_{\pm 0.23}$	$0.166_{\pm 0.23}$	$0.167_{\pm 0.23}$	$0.167_{\pm 0.23}$
	Time/s ↓	$0.001_{\pm 0.002}$	$0.008_{\pm 0.013}$	$0.001_{\pm 0.003}$	$0.002_{\pm 0.005}$
	Compl. ↓	—	—	—	—
LGBM	R^2 ↑	$0.574_{\pm 0.32}$	$0.566_{\pm 0.32}$	$0.574_{\pm 0.32}$	$0.571_{\pm 0.32}$
	Time/s ↓	$0.030_{\pm 0.007}$	$4.7_{\pm 8.2}$	$0.029_{\pm 0.007}$	$0.050_{\pm 0.18}$
	Compl. ↓	—	—	—	—
RandomForest	R^2 ↑	$0.697_{\pm 0.21}$	$0.692_{\pm 0.21}$	$0.697_{\pm 0.21}$	$0.695_{\pm 0.21}$
	Time/s ↓	$0.15_{\pm 0.11}$	$0.23_{\pm 0.19}$	$0.16_{\pm 0.11}$	$0.16_{\pm 0.13}$
	Compl. ↓	—	—	—	—
XGB	R^2 ↑	$0.651_{\pm 0.25}$	$0.647_{\pm 0.26}$	$0.649_{\pm 0.26}$	$0.648_{\pm 0.26}$
	Time/s ↓	$0.10_{\pm 0.067}$	$0.16_{\pm 0.14}$	$0.10_{\pm 0.071}$	$0.12_{\pm 0.12}$

Method	Metric	$\sigma=0$	$\sigma=0.001$	$\sigma=0.01$	$\sigma=0.1$
	Compl. ↓	—	—	—	—
LLM-SR	R^2 ↑	$0.559_{\pm 0.29}$	$0.557_{\pm 0.28}$	$0.541_{\pm 0.28}$	$0.513_{\pm 0.26}$
	Time/s ↓	$139.0_{\pm 72.9}$	$143.3_{\pm 69.5}$	$137.9_{\pm 63.8}$	$141.7_{\pm 70.2}$
	Compl. ↓	$15.9_{\pm 9.2}$	$17.2_{\pm 7.8}$	$15.5_{\pm 8.6}$	$15.7_{\pm 9.1}$
RESTART	R^2 ↑	$0.501_{\pm 0.34}$	$0.499_{\pm 0.34}$	$0.490_{\pm 0.33}$	$0.462_{\pm 0.31}$
	Time/s ↓	$83.0_{\pm 66.9}$	$85.6_{\pm 81.0}$	$82.8_{\pm 84.1}$	$80.9_{\pm 85.4}$
	Compl. ↓	$145.3_{\pm 60.8}$	$152.3_{\pm 66.1}$	$142.8_{\pm 56.1}$	$141.2_{\pm 54.3}$
SR-SCIENTIST	R^2 ↑	$0.528_{\pm 0.37}$	$0.543_{\pm 0.36}$	$0.478_{\pm 0.38}$	$0.487_{\pm 0.33}$
	Time/s ↓	$2821.3_{\pm 1609.2}$	$2816.3_{\pm 1903.1}$	$3061.7_{\pm 1746.8}$	$2856.8_{\pm 1701.8}$
	Compl. ↓	$18.6_{\pm 16.0}$	$17.4_{\pm 15.4}$	$17.0_{\pm 17.4}$	$21.4_{\pm 23.1}$
EditFlowSR (Ours)	R^2 ↑	$0.707_{\pm 0.21}$	$0.717_{\pm 0.21}$	$0.736_{\pm 0.21}$	$0.673_{\pm 0.20}$
	Time/s ↓	$130.5_{\pm 123.2}$	$120.0_{\pm 108.9}$	$123.6_{\pm 113.1}$	$134.8_{\pm 108.3}$
	Compl. ↓	$38.4_{\pm 11.6}$	$36.8_{\pm 11.0}$	$37.7_{\pm 11.6}$	$36.8_{\pm 11.8}$

C.4 Redundancy Localization and Approximate Calibration Protocol

Starting from each concise base expression f, we construct a positive probe $\tilde{f}^+$ by injecting one synthetic algebraic redundancy and a matched negative probe $\tilde{f}^-$ by keeping the same local syntactic template but replacing the injected fragment with a non-redundant control. The evaluation covers simple wrappers, double negation, cancellation, and multiply-by-zero patterns, for a total of 689 base formulas and 5512 matched positive-negative probe pairs.

For a positive probe of length T, let $S_+ \subseteq \{1,\dots,T\}$ denote the token positions belonging to the injected redundant region, and let S_{ctrl} denote the matched token positions in its negative-control counterpart. Let d_i be the token-level deletion score and r_i the aggregate substitution score produced by the edit decoder at position i. We evaluate two ranking scores,

$$g_i = d_i, \qquad g_i^{\text{rob}} = d_i + r_i,$$

where the main score g_i measures pure deletion preference and the robust score also credits rewrite behavior.

Given a token score $q_i \in \{g_i, g_i^{\text{rob}}\}$ and threshold τ, the predicted redundant set is

$$\hat{S}_q(\tau) = \{i \mid q_i \geq \tau\}.$$

Precision and recall are computed as

$$\text{Precision}_q(\tau) = \frac{|\hat{S}_q(\tau) \cap S_+|}{|\hat{S}_q(\tau)|}, \qquad \text{Recall}_q(\tau) = \frac{|\hat{S}_q(\tau) \cap S_+|}{|S_+|}.$$

The token-level AUPRC for one formula is the area under this precision-recall curve. Macro-AUPRC averages token-level AUPRC across formulas, whereas micro-AUPRC pools all token decisions across the full evaluation set before computing a single precision-recall curve.

We also compute

$$\text{Gap}_q = \frac{1}{|S_+|}\sum_{i \in S_+} q_i - \frac{1}{|S_{\text{ctrl}}|}\sum_{i \in S_{\text{ctrl}}} q_i,$$

$$\text{GapRatio}_q = \frac{\frac{1}{|S_+|}\sum_{i \in S_+} q_i}{\frac{1}{|S_{\text{ctrl}}|}\sum_{i \in S_{\text{ctrl}}} q_i}.$$

Larger values indicate better redundancy sensitivity. Top-k hit is the fraction of positive probes for which at least one token in S_+ appears among the top-k positions ranked by q_i; we report $k \in \{1,3,5\}$. Section 5.3 reports the main-score results; the robust-score variant uses the same protocol.

The same benchmark also evaluates calibration under small semantic perturbations. Starting from the same concise base expressions f, we construct approximate probes by injecting one coefficient-controlled perturbation from four families,

$$f + \eta, \qquad f + \eta x_j, \qquad f - \eta x_j, \qquad f + \eta x_j x_j,$$

with $\eta \in \{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, giving $689 \times 4 \times 6 = 16536$ approximate probes per model. For each probe, let $c(\eta)$ denote the deletion score assigned to the injected coefficient token. We summarize three aggregate quantities:

$$\text{NearScore} = \mathbb{E}\Big[c(\eta) \mid \eta \in \{10^{-5}, 10^{-4}\}\Big],$$

$$\text{FarScore} = \mathbb{E}\big[c(\eta) \mid \eta \in \{10^{-2}, 10^{-1}\}\big],$$

$$\text{NearFarGap} = \text{NearScore} - \text{FarScore}.$$

Higher NearScore indicates stronger detection of very small perturbations as approximately redundant, whereas lower FarScore indicates better tolerance to clearly non-redundant perturbations. Section 5.3 reports the aggregated results on all four perturbation families.

D Additional Analyses

D.1 Expression Nesting Depth

Table 8 lists the maximum expression nesting depth on PMLB. Shallower expressions are easier to read and inspect. EditFlowSR remains among the lowest-depth methods across the PMLB subsets.

Table 8: Summary of maximum expression nesting depth on PMLB. Lower values indicate shallower expressions.

Method	Feynman	Strogatz	Black-box
E2E	12.64	13.59	15.30
SR4MDL	9.90	5.52	10.43
TPSR	11.34	11.75	12.87
GenSR	6.91	6.82	10.19
SNIP	8.84	8.88	10.71
EditFlowSR (Ours)	8.62	7.50	9.75

D.2 Structural Stability Analysis

We evaluate structural stability with the Effective Information Criterion (EIC) [Yu et al., 2026]. Unlike complexity, which merely counts tokens, EIC treats the expression as a recursive information processing channel and measures the amplification of rounding noise through the computation tree. For node v in the computation tree, its cumulative error amplification factor satisfies

$$\sigma_v^2(\mathbf{x}) = 1 + \sum_{i \in \text{ch}(v)} \gamma_{v,i}^2(\mathbf{x}) \sigma_i^2(\mathbf{x}),$$

where $\text{ch}(v)$ denotes the child-node set, and $\gamma_{v,i}(\mathbf{x})$ denotes the relative condition number of child i with respect to parent v. The final definition is

$$\text{EIC}(f) = \max_{v \in T[f]} \log_{10} \bar{\sigma}_v, \qquad \bar{\sigma}_v^2 = \mathbb{E}_{\mathbf{x}}[\sigma_v^2(\mathbf{x})],$$

where the expectation is taken over the observed inputs. A lower EIC indicates that the expression amplifies less rounding noise during recursive computation, has a more stable internal structure, and is less likely to contain pathological nesting or redundant transformations. Table 9 reports the EIC comparison, and the overall results support EditFlowSR's advantage in structural stability.

D.3 Sensitivity of Inference Hyperparameters

We evaluated EditFlowSR on a representative dataset over a grid of the two inference hyperparameters, the initial input length $\ell_0 \in \{10, 20, \ldots, 70\}$ and the number of edit steps $n_{\text{steps}} \in \{20, 40, \ldots, 180\}$, with 5 runs per configuration. Table 10 reports the mean test R^2 of every configuration; the last column and the last row marginalize over the other hyperparameter.

Table 9: EIC comparison across representative baselines and internal EditFlowSR variants. Values follow $\sigma = 0/0.001/0.01/0.1$ and aggregate Feynman, Strogatz, and Black-box results.

Method	Mean EIC ↓ (∞ = failure)	EIC < 2 (%) ↑
E2E	3.833 / 3.878 / 4.872 / 4.938	2.7 / 2.7 / 1.4 / 0.9
TPSR	3.804 / 3.881 / 3.927 / 5.023	2.2 / 2.2 / 1.9 / 0.7
SR4MDL	6.849 / 8.479 / 9.464 / 9.085	17.6 / 15.0 / 5.6 / 3.7
GenSR	3.331 / 3.298 / 3.547 / 3.273	17.1 / 20.4 / 20.4 / 20.4
SNIP	3.442 / 3.167 / 3.510 / 3.845	16.9 / 14.4 / 10.0 / 6.0
FFX	7.092 / 6.718 / 4.262 / 3.649	5.9 / 8.3 / 4.8 / 1.9
GPlearn	4.470 / 3.560 / 4.855 / 3.511	22.5 / 25.1 / 24.7 / 22.1
DSR	3.430 / 4.193 / 4.320 / 4.566	23.6 / 18.8 / 22.5 / 22.5
MRGP	5.854 / 4.778 / 7.889 / 4.742	3.7 / 4.4 / 1.8 / 2.2
Operon	4.226 / 4.181 / 4.102 / 4.368	14.4 / 14.0 / 14.8 / 11.8
EditFlowSR (No Repair)	3.284 / 3.195 / 3.086 / 3.139	16.4 / 15.7 / 17.6 / 15.4
EditFlowSR (Non-equivalent Shortening)	3.476 / 3.594 / 3.359 / 3.298	15.4 / 10.9 / 12.0 / 12.4
EditFlowSR	2.981 / 3.151 / 3.025 / 3.493	21.3 / 15.7 / 17.2 / 14.6

Table 10: Sensitivity of EditFlowSR to the inference hyperparameters on a representative dataset. Each cell reports the mean test R^2 over 5 runs for one combination of the initial input length ℓ_0 and the number of edit steps n_{steps}. The last column and the last row show means over the other hyperparameter, and the bottom-right entry is the grand mean. Bold marks the best value in each marginal.

	Number of edit steps n_{steps}									
Input length ℓ_0	20	40	60	80	100	120	140	160	180	Mean
10	0.218	0.381	0.432	0.436	0.456	0.405	0.433	0.419	0.429	0.401
20	0.416	0.535	0.540	0.592	0.580	0.592	0.610	0.594	0.599	0.562
30	0.374	0.520	0.524	0.536	0.584	0.525	0.554	0.542	0.557	0.524
40	0.405	0.574	0.612	0.648	0.625	0.605	0.624	0.619	0.616	**0.592**
50	0.350	0.477	0.544	0.565	0.554	0.528	0.549	0.543	0.552	0.518
60	0.321	0.477	0.522	0.527	0.540	0.499	0.535	0.514	0.529	0.496
70	0.296	0.452	0.515	0.483	0.476	0.486	0.503	0.506	0.504	0.469
Mean	0.340	0.488	0.527	0.541	**0.545**	0.520	0.544	0.534	0.541	0.509

For the initial input length, the marginal mean R^2 peaks in $\ell_0 \in [20,40]$ (up to 0.592 at $\ell_0 = 40$) and decreases for longer initial drafts, down to 0.469 at $\ell_0 = 70$. An overlong initial draft places a heavier burden on the editor: more positions must be revised before the expression converges to a compact form, and the larger structure leaves more room for spurious terms, while a very short draft ($\ell_0 = 10$) also limits performance. For the number of edit steps, the mean R^2 rises sharply from 0.340 at $n_{\text{steps}} = 20$ to 0.541 at $n_{\text{steps}} = 80$ and then saturates, with all marginal means beyond $n_{\text{steps}} = 80$ within a range of 0.025; 100 steps thus already discretize the editing process finely enough. The inference configuration of Appendix B.5 (ℓ_0 drawn uniformly from $\{10,\ldots,40\}$ and $n_{\text{steps}} = 100$) lies inside this stable region, and performance varies little around it.

D.4 Qualitative Edit-Path Example

Table 11 traces a 100-step edit path for `strogatz_lv1`, whose ground-truth expression is

$$y = 3x_1 - 2x_1x_2 - x_1^2. \tag{23}$$

The path starts from a random sequence state whose tokens are random numeric constants and operators. At most steps the model leaves the sequence state unchanged, and the checkpoints in Table 11 are exactly the steps at which the model jumps to a substantially different state. Through insertions, deletions, and substitutions, the sequence state gradually concentrates the tokens into a meaningful structure. The operators and variables are progressively relocated into the prefix positions,

the random constants are replaced by numeric coefficients, and the redundant trailing tokens are removed. After a constant-refinement step, the expression is recovered as `x_0 * (-x_0 + 3 - 2 * x_1)`, which matches the ground truth of (23) with $R^2 = 1$.

Table 11: Checkpoints from a 100-step illustrative edit path on `strogatz_lv1`. Only the steps at which the sequence state changes are recorded. The tokens `add` and `mul` denote prefix addition and multiplication; `x_0` and `x_1` are input variables corresponding to x_1 and x_2. Numeric constants use a sign token, mantissa tokens of the form `Nxxxx`, and an exponent token `Ek`, representing $\pm\text{mantissa} \times 10^k$.

Step	Input trace
1	`BOS,N2172,N2609,N7326,N6848,N6804,N6194,N8427,N2876,N4808,+,N3486,N5681,N9832,N0056,N0942,N4177,N8080,N3540,N4622,N3085,N0899,N9472,N5593,N3815,N9196,N6994,N1878,N3976`
10	`BOS,mul,N2172,N2609,mul,N6848,N6804,N6194,x_1,N2876,add,+,N3486,N5681,N9832,N0056,N0942,E-4,N8080,N3540,N4622,N3085,N0899,N9472,N5593,N3815,N9196,N6994,N1878,N3976`
26	`BOS,mul,mul,N2172,N2609,mul,-,N6848,N6194,x_1,N2876,add,+,N3486,N5681,N9832,+,N0942,E-4,N8080,N3540,N4622,N3085,N0899,N9472,N5593,N3815,N9196,N6994,x_0`
41	`BOS,mul,N2172,N2609,mul,N6848,N6804,N6194,x_1,N2876,add,+,N3486,N5681,N9832,N0056,N0942,E-4,N8080,N3540,N4622,N3085,N0899,N9472,N5593,N3815,N9196,N6994,x_0`
50	`BOS,mul,x_0,add,N6848,N6804,mul,N6194,x_1,N2876,add,+,N3486,E-3,N9832,+,N0942,E-4,N8080,x_0`
58	`BOS,mul,x_0,add,-,x_0,add,+,N3486,E-3,mul,-,N6194,E-3,x_1,N2876`
77	`BOS,mul,x_0,add,-,x_0,add,+,N3010,E-3,mul,-,N1930,E-3,x_1`
100	`BOS,mul,x_0,add,mul,-,N1000,E-3,x_0,add,+,N3062,E-3,mul,-,N1979,E-3,x_1`
after BFGS	`x_0 * (-x_0 + 3 - 2 * x_1)`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction claim that EditFlowSR reformulates end-to-end symbolic regression as iterative expression editing, uses dual-trajectory supervision to couple discovery with structural repair, and shows empirical advantages in redundancy correction and the trade-off among predictive accuracy, expression simplicity, and inference time. These claims are grounded in the model, training, and inference overview in Figure 1, and in the benchmark, ablation, and redundancy-localization results in Figure 2, Figure 3, Table 3, and Table 9. The repair-expansion mechanism and metrics are further specified in Appendix B.2, Appendix D.2, and Appendix C.3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper explicitly states in Section 6 that EditFlowSR is trained for end-to-end generation and that adapting its editing dynamics to guide optimization may require task-specific training. Its empirical discussion is supported by the benchmark results in Figure 3, Appendix C.3, and Table 9. The iterative-edit inference procedure itself is stochastic, as shown in Algorithm 2.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Appendix A.3 restates Theorem 3.1 of [Havasi et al., 2025] and cites its complete proof in the original paper. The remaining mathematical content in Section 3 and Section 4 defines the edit-flow process, the dual-trajectory objective, and the inference procedure; these are definitions and algorithmic specifications rather than new theorem statements.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The model architecture, dual-trajectory training procedure, and iterative inference procedure are specified in Figure 1 and Algorithm 2; the implementation details, hyperparameters, hardware, and evaluation protocol are listed in Appendix B.1, Appendix B.4, Appendix D.2, and Section 5. The benchmark datasets are publicly available and cited, and the detailed per-dataset results are reported in Appendix C.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper provides a code repository at https://github.com/SCUT-EIPI/EditFlowSR. The benchmark data come from public sources cited in Section 5 and Appendix C, and the implementation details needed to run the reported experiments are given in Appendix B.1 and Appendix B.4.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The training and test setup is specified across Section 5, Appendix C, Appendix B.1, and Appendix B.4. These sections cover the benchmark suites and noise settings, the data-generation and evaluation protocol, the optimizer (Adam), effective batch size (336), learning rate (4×10^{-5}), training duration (195 epochs, $\sim$18 hours), total training exposure (65,520,000 online-generated samples), and the hardware/runtime environment. The baseline configurations and metric definitions are also stated explicitly.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports uncertainty information for the main empirical results as mean $\pm$ standard deviation across the evaluated datasets in the benchmark tables in Appendix C.3, and as 95% confidence intervals in the Strogatz noise-robustness figure and Figure 4. The submission does not use formal hypothesis tests, but it does provide quantitative variability summaries for the experiments supporting the main claims.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix B.4 specifies the compute environment used for the experiments: an Intel Xeon Gold 6226R CPU with 2×16 cores and 64 threads, 128 GB of memory, NVIDIA RTX 3090 GPUs (24 GB each), and PyTorch 2.10. It also reports the training setup and cost for EditFlowSR, including 3-GPU DDP training with FP16 mixed precision, and approximately 18 hours for 195 epochs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work uses only public benchmark datasets described in Section 5, does not involve human subjects or personal data, and includes a broader-impacts discussion in Section 6. The paper studies symbolic regression on mathematical expressions and does not raise obvious dual-use or participant-risk concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 6 includes a Broader Impacts paragraph discussing the positive value of interpretable symbolic modeling and edit-path transparency. For this work, the potential negative societal impact is limited because the study concerns symbolic regression on public mathematical benchmark datasets rather than human, personal, or safety-critical data, as reflected by the experimental setting in Section 5.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release models or datasets with a high risk of misuse. It studies symbolic regression on public mathematical benchmarks in Section 5, and the conclusion in Section 6 does not introduce a high-risk model or dataset that would require release safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The reused assets are all public academic benchmarks and baseline methods cited in Section 5, Appendix C, and Section 6 (including Feynman [Udrescu and Tegmark, 2020], Strogatz/SRBench [Strogatz, 2024, Cava et al., 2021], and PMLB [Olson et al., 2017]). The paper does not redistribute third-party assets or claim ownership of them, and it uses them in the form described by their original sources.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces a released code package for EditFlowSR, with the implementation, algorithms, hyperparameters, and evaluation protocol documented in Section 5, Appendix B.1, Appendix B.4, and the inference algorithm in Algorithm 2. Trained weights are planned for later release, and no new dataset is introduced.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The experiments use only public mathematical benchmark data and do not involve crowd workers, participant instructions, compensation, personal data collection, or other human-subject procedures; see Section 5.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Section 5 studies only public mathematical benchmarks and does not involve participant recruitment, human data collection, intervention, or any other human-subject procedure, so IRB review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

Justification: The core method is specified in Section 4 and evaluated in Section 5 without any LLM component; any LLM assistance was limited to writing/editing and auxiliary coding support outside the scientific contribution.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.