APPENDIX

A MODEL SETTING

In this section, we give more details about the settings of our models.

A.1 HYPERPARAMETERS

The full set of hyperparameters can be seen in Table S1.

Table S1: Hyperparameters of our model

Name	Abbreviation	Value
RL parameters		
Minimum expression lengths	l_{min}	4
Maximum expression lengths	l_{max}	35
Maximum number of parameters	c_{max}	10
Length discount rate	η	0.99
Training rounds	t_r	50
UCT constant	c	$\sqrt{2}$
Minimum selected times	n_0	3
Learning rate of double Q-learning	lr	10^{-3}
Genetic Programming parameters		
GP rounds	t_{gp}	30
GP population	p_{gp}	500
GP number of best expressions	l_b	20
GP Mate rate	p_{mate}	0.5
GP Mutate rate	p_{mutate}	0.5
MSDB parameters		
Error of Symmetry	E_{sym}	10^{-5}
Selection ratio	k_s	0.1
Expression percentage ratio	k_p	0.1
Maximum select number	N	5

A.2 EXPRESSION CONSTRAINT

In this study, we incorporate a prior constraint inspired by the DSR (Petersen et al., 2019) method to effectively reduce the search space for expressions. The following constraints are applied:

- **Length constraint**: The length of expressions is restricted within pre-defined minimum and maximum values. If the current length falls below the minimum threshold, variables (x, y, C, etc.) and parameters will not be generated. Conversely, if the current length, combined with the number of nodes to be generated, reaches the maximum length, only these nodes will be considered.
- **Unary operator constraint**: The direct successor node of a unary operator should not be the inverse of that same operator. This constraint ensures that the generated expressions adhere to the intended structure and prevent redundant combinations.
- **Trigonometric function constraint**: The successor node of a trigonometric function node should not be another trigonometric function. This constraint prevents the generation of expression structures that lead to unnecessary complexity or redundancy.
- **Maximum parameter limit**: A specified maximum number of parameters is imposed to control the complexity of the expressions and prevent overfitting.

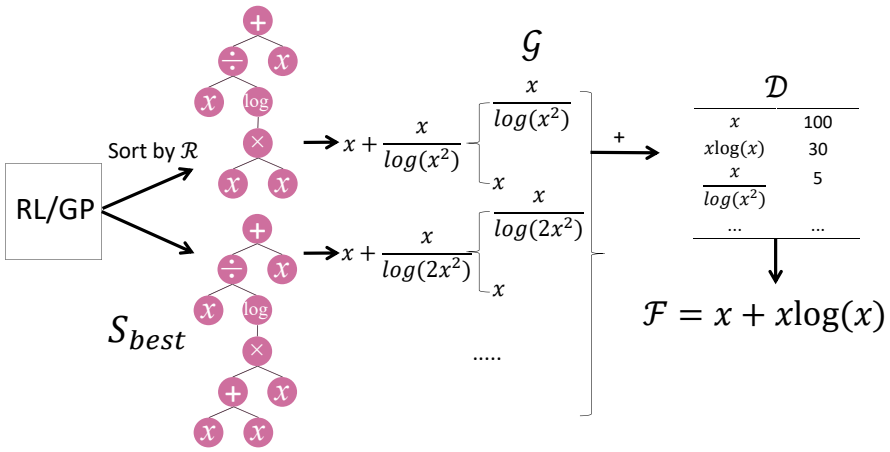

Figure S1: Schematic of the proposed Form-discovery. We first obtain the output of RL/GP and select the equations in which the loss is relatively small. After that, we separate these equations by plus and minus signs and count the number of times these sub-expressions occur overall, and use the smaller equations with more occurrences to cobble together to create new expression forms.

By applying these expression constraints, we aim to enhance the search efficiency and guide the generation of meaningful expressions that align with the desired properties of the target problem.

A.3 Genetic Programming

Following the generation of expressions by each reinforcement learning algorithm, we engage in the optimization of a predetermined set of expressions. For this purpose, we employ a genetic algorithm, utilizing the DEAP library in Python. This algorithm initializes half of the population using the outcomes of the reinforcement learning process, while the remaining half is generated randomly. Subsequently, we retain the most promising expressions and subject them to further analysis using a subtree analyzer. This process serves to update and refine the expression form, enhancing the overall efficacy of our approach.

A.4 Form Discovery

Presented below is an elucidative diagram (refer to Figure S1) pertaining to the process of form discovery as outlined in Algorithm 2. This visual aid serves to elucidate the sequential progression within the algorithm and delineates the roles and interpretations of variables such as S_{sym}, $\mathcal{G}$, $\mathcal{D}$, and more.

B Baseline Models

In this section, we give more details about the settings of baselines.

- **SPL** (Sun et al., 2023): Rooted in the use of MCTS, SPL employs various greedy strategies to ensure efficient exploration. It utilizes full expressions as sub-trees to maximize the use of past information. While excelling in shorter expressions, it falls short in handling longer ones.
- **DSR** (Petersen et al., 2019): DSR takes a gradient-based RL approach along with a recurrent neural network (RNN) that generates a probability distribution over expressions. While effective for longer expressions, it may exhibit limitations in generalization ability.
- **NGGP** (Mundhenk et al., 2021a): Building upon DSR, NGGP enhances its capabilities. Expressions sampled via probability distribution undergo further optimization using GP. The refined expressions then train the RNN with risk-seeking policy gradient.

- **uDSR** (Landajuela et al., 2022): The uDSR amalgamates DSR, AIFeynman, LSPT (Large-scale pre-training), GP, and LM (Linear models). It excels in discovering formulas with constants (favoring polynomial type expressions), but albeit at the cost of increased computation time.
- **DGSR** (Holt et al., 2022): This model leverages pre-trained deep generative models to capture the inherent patterns and structures within equations. This pre-training phase establishes a robust foundation for the subsequent optimization steps conducted through genetic programming.
- **gplearn**(Stephens, 2016): The gplearn offers an efficient and rapid GP-based SR implementation. While proficient in speed, it may exhibit instability and poor scalability.
- **AFP-FE** (Schmidt & Lipson, 2010): Age-Fitness Pareto Optimization is an optimization technique that combines two important factors, age, and fitness, to enhance the performance of evolutionary algorithms. FE means Co-evolved Fitness Predictors.

The full set of hyperparameters can be seen below.

- **SPL**: In line with the original paper, we maintain the same parameter settings for SPL. The discount rate is set to $\eta = 0.9999$, and the candidate operators include addition $(+)$, subtraction $(-)$, multiplication $(\times)$, division $(\div)$, cosine $(\cos(\cdot))$, sine $(\sin(\cdot))$, exponential $(\exp(\cdot))$, natural logarithm $(\log(\cdot))$, and square root $(\sqrt{\cdot})$. Other parameter values are as follows: Maximum Module Transplantation: 20, Episodes Between Module Transplantation: 50000, Maximum Tree Size: 50, and Maximum Augmented Grammars: 5.
- **DSR/NGGP/uDSR**: In our study, we adopt the standard parameter configurations as provided in the publicly available implementation of Deep Symbolic Optimization (DSO). This approach entails adjusting two primary hyperparameters. The entropy coefficient is set $\lambda H = 0.05$ and the risk factor is set $\epsilon = 0.005$. Candidate operators are the same as those employed in the SPL. Additionally, NGGP incorporates other hyperparameters related to hybrid methods based on genetic programming. The specific values are listed in Table S2.
- **Genetic Programming (GP)**: We employ the gplearn library for GP-based methods. The hyperparameters for genetic programming are identical to those presented in Table S2.
- **DGSR/AFP-FE**: For both of these models, we exclusively utilized the results obtained from the srbench dataset (La Cava et al., 2021), and the parameters were meticulously tested in accordance with the specifications provided in the official srbench dataset and its associated article.

Table S2: Genetic Programming Hyperparameters on baselines

Name	Value
Rounds	20
Population	1000
Mate rate	0.5
Mutate rate	0.5

C Basic Benchmark Result

C.1 Overall Result

In this section, we give more details about the Min Depth and Min Complexity of difficult equations in each benchmark in Table S3.

In this table, the performance of uDSR appears to be subpar, even falling short of its predecessor NGGP. This outcome can be attributed to two primary reasons. Firstly, our application of symbolic learning lacks parameter optimization, except for the Nyugen-c dataset, in which uDSR notably excels due to its compatibility with parameterized scenarios. Secondly, uDSR exhibits a stronger tendency towards generating polynomial functions, whereas

our tests predominantly involve a substantial number of trigonometric and exponential functions. For instance, expressions such as $x_1x_2x_3(\sin(x_4) + \cos(x_5))$ result in complex equations like $x_1\left[-0.0437x1^3 - 5.84x3 - 0.01x4^3 + 0.0038x_4^2x5 - 0.492x_4x_5 + x_4(x_2 + \sin(x_5))...\right]$.

C.2 NYUGEN BENCHMARK RESULT

Nyugen Benchmark is a standard benchmark for symbolic learning with one or two independent variables and equations randomly sampled over a range. And Nyugen^c is a parametric version of the Nguyen benchmark, allowing the use of parametric optimization to test equations with parameters. In this section, we provide additional details about the results obtained from the Nyugen and Nyugen^c Benchmark experiment.

By referring to Table S4, readers can obtain more detailed information about the performance of each model on each expression, their comparative analysis, and any other relevant insights derived from the experiment.

C.3 LIVERMORE BENCHMARK RESULT

LiverMore Benchmark contains challenging equations rarely encountered in symbolic learning, including high exponentials, trigonometric functions, and complex polynomials. In this section, we provide additional details about the results obtained from the LiverMore Benchmark experiment.

By referring to Table S5, readers can obtain more detailed information about the performance of each model on each expression, their comparative analysis, and any other relevant insights derived from the experiment.

C.4 R BENCHMARK RESULT

R Benchmark consists of three built-in rational equations with numerous polynomials as divisors and devisees, increasing the learning difficulty. In this section, we provide additional details about the results obtained from the R Rational Benchmark experiment.

By referring to Table S6, readers can obtain more detailed information about the performance of each model on each expression, their comparative analysis, and any other relevant insights derived from the experiment.

Table S3: Minimum Depth of expression tree and Minimum tokens of expression tree as Minimum Complexity of several difficult equations in each benchmark.

BenchMark	Equation	Min Depth	Min Complexity
Nguyen-5	$\sin(x_1^2)\cos(x_1) - 1$	6	12
Nguyen-12	$x_1^4 - x_1^3 - 0.5x_2^2 + x_2$	7	24
Nguyen-2^c	$0.48x_1^4 + 3.39x_1^3 + 2.12x_1^2 + 1.78x_1$	6	25
Nguyen-9^c	$\sin(1.5x_1) + \sin(0.5x_2^2)$	5	11
LiverMore-3	$\sin(x_1^3)\cos(x_1^2) - 1$	6	15
LiverMore-7	$\sinh(x_1)$	6	15
LiverMore-16	$x_1^{2/5}$	7	17
LiverMore-18	$\sin(x_1^2)\cos(x_1) - 5$	6	19
AIFeynman-9	$x_1 + x_2 + 2\sqrt{x_1x_2}\cos(x_3)$	6	17
AIFeynman-10	$\frac{1}{2}x_1(x_2^2 + x_3^2 + x_4^2)$	6	20
R-1	$(x_1 + 1)^3/(x_1^2 - x_1 + 1)$	6	21
R-2	$(x_1^5 - 3x_1^3 + 1)/(x_1^2 + 1)$	7	31
R-3	$(x_1^5 + x_1^6)/(x_1^4 + x_1^3 + x_1^2 + x_1 + 1)$	7	35

Table S4: Average Recovery Rate (%) of the Nyugen Benchmark over 100 parallel runs

Name	Equation	Ours	SPL	uDSR	NGGP	DSR	GP
Nguyen-1	$x_1^3 + x_1^2 + x_1$	**100**	**100**	**100**	**100**	**100**	99
Nguyen-2	$x_1^4 + x_1^3 + x_1^2 + x_1$	**100**	**100**	**100**	**100**	**100**	90
Nguyen-3	$x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$	**100**	**100**	**100**	**100**	**100**	34
Nguyen-4	$x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$	**100**	99	**100**	**100**	**100**	54
Nguyen-5	$\sin(x_1^2)\cos(x_1) - 1$	**100**	95	45	80	72	12
Nguyen-6	$\sin(x_1) + \sin(x_1 + x_1^2)$	**100**	**100**	**100**	**100**	**100**	11
Nguyen-7	$\log(x_1 + 1) + \log(x_1^2 + 1)$	**100**	**100**	98	**100**	35	17
Nguyen-8	$\sqrt{x_1}$	**100**	**100**	**100**	**100**	96	**100**
Nguyen-9	$\sin(x_1) + \sin(x_2^2)$	**100**	**100**	91	**100**	**100**	17
Nguyen-10	$\sin(x_1)\cos(x_2)$	**100**	**100**	**100**	**100**	**100**	86
Nguyen-11	$x_1^{x_2}$	**100**	**100**	87	**100**	**100**	13
Nguyen-12	$x_1^4 - x_1^3 - 0.5x_2^2 + x_2$	**100**	28	30	21	0	0
	Average	**100.00±0.0**	93.50±11.7	87.58±13.6	91.75±13.0	83.58±18.5	44.42±22.1
Nguyen-1^c	$3.39x_1^3 + 2.12x_1^2 + 1.78x_1$	**100**	**100**	58	**100**	**100**	0
Nguyen-2^c	$0.48x_1^4 + 3.39x_1^3 + 2.12x_1^2 + 1.78x_1$	**100**	94	**100**	**100**	**100**	0
Nguyen-5^c	$\sin(x_1^2)\cos(x_1) - 0.75$	**100**	95	67	98	0	1
Nguyen-7^c	$\log(x_1 + 1.4) + \log(x_1^2 + 1.3)$	**100**	0	**100**	**100**	93	2
Nguyen-8^c	$\sqrt{1.23x_1}$	**100**	**100**	**100**	**100**	**100**	56
Nguyen-9^c	$\sin(1.5x_1) + \sin(0.5x_2^2)$	**100**	98	0	96	0	0
Nguyen-10^c	$\sin(1.5x_1)\cos(0.5x_2)$	**100**	0	0	**100**	**100**	0
	Average	**100.00±0.0**	69.57±35.2	60.71±33.2	99.14±1.2	70.43±35.7	8.43±15.6

Table S5: Average Recovery Rate (%) of the LiverMore Benchmark over 100 parallel runs

Name	Equation	Ours	SPL	uDSR	NGGP	DSR	GP
Livermore-1	$1/3 + x_1 + \sin(x_1)$	**100**	94	**100**	**100**	67	**100**
Livermore-2	$\sin(x_1^2)\cos(x_1) - 2$	**100**	29	58	61	26	1
Livermore-3	$\sin(x_1^3)\cos(x_1^2) - 1$	**55**	50	0	2	0	0
Livermore-4	$\log(x_1 + 1) + \log(x_1^2 + x_1) + \log(x_1)$	**100**	61	**100**	**100**	72	**100**
Livermore-5	$x_1^4 - x_1^3 + x_1^2 - x_2$	**100**	**100**	**100**	**100**	55	**100**
Livermore-6	$4x_1^4 + 3x_1^3 + 2x_1^2 + x_1$	**100**	8	**100**	**100**	**100**	**100**
Livermore-7	$\sinh(x_1)$	**100**	18	0	24	0	0
Livermore-8	$\cosh(x_1)$	**100**	6	8	30	0	0
Livermore-9	$\sum_{i=1}^{9} x_1^i$	**100**	21	**100**	99	18	0
Livermore-10	$6\sin(x_1)\cos(x_2)$	**100**	75	**100**	**100**	70	23
Livermore-11	$(x_1^2 x_2^2)/(x_1 + x_2)$	**100**	0	**100**	**100**	78	95
Livermore-12	x_1^5/x_2^3	**100**	**100**	**100**	**100**	13	**100**
Livermore-13	$x_1^{1/3}$	**100**	12	**100**	**100**	59	0
Livermore-14	$x_1^3 + x_1^2 + x_1 + \sin(x_1) + \sin(x_1^2)$	**100**	**100**	**100**	**100**	91	**100**
Livermore-15	$x_1^{1/5}$	**100**	0	**100**	**100**	28	2
Livermore-16	$x_1^{2/5}$	**100**	0	60	26	0	0
Livermore-17	$4\sin(x_1)\cos(x_2)$	**100**	89	**100**	**100**	**100**	84
Livermore-18	$\sin(x_1^2)\cos(x_1) - 5$	**100**	18	59	33	37	0
Livermore-19	$x_1^5 + x_1^4 + x_1^2 + x_1$	**100**	89	98	**100**	**100**	**100**
Livermore-20	$\exp(-x_1^2)$	**100**	**100**	**100**	**100**	**100**	**100**
Livermore-21	$\sum_{i=1}^{8} x_1^i$	**100**	52	**100**	**100**	13	12
Livermore-22	$\exp(-0.5x_1^2)$	**100**	**100**	**100**	**100**	82	**100**
	Average	**97.95±4.0**	51.00±16.9	81.05±14.6	80.68±14.0	50.41±15.7	50.77±20.4

Table S6: Average Recovery Rate (%) of the R Rational Benchmark over 100 parallel runs

Name	Equation	Ours	SPL	uDSR	NGGP	DSR	GP
R^0-1	$(x_1 + 1)^3/(x_1^2 - x_1 + 1)$	5	0	**82**	15	0	0
R^0-2	$(x_1^5 - 3x_1^3 + 1)/(x_1^2 + 1)$	**80**	0	0	40	0	0
R^0-3	$(x_1^5 + x_1^6)/(x_1^4 + x_1^3 + x_1^2 + x_1 + 1)$	**100**	0	0	**100**	0	0
R^*-1	$(x_1 + 1)^3/(x_1^2 - x_1 + 1)$	**48**	0	17	2	0	0
R^*-2	$(x_1^5 - 3x_1^3 + 1)/(x_1^2 + 1)$	**89**	0	0	0	0	0
R^*-3	$(x_1^5 + x_1^6)/(x_1^4 + x_1^3 + x_1^2 + x_1 + 1)$	**91**	0	0	3	0	0
	Average	**68.83±28.9**	0.00±0.0	16.50±26.2	26.67±31.1	0.00±0.0	0.00±0.0

C.5 AIFEYNMAN BENCHMARK RESULT

AIFeynman Benchmark contains lots of equations with physical meaning (as part of SRBench (La Cava et al., 2021)), such as expressions for gravity, kinetic energy, and light intensity superposi-

tion. In this section, we provide additional details about the results obtained from the AIFeynman Benchmark experiment.

By referring to Table S7, readers can obtain more detailed information about the performance of each model on each expression, their comparative analysis, and any other relevant insights derived from the experiment.

The selection of 12 AI Feynman equations was based on a stratified representation of difficulty levels. The chosen expressions include easy ones, such as x_1x_2 and $\frac{3}{2}x_1x_2$; medium complexity expressions like $x_1x_2x_3\sin(x_4)$ and $x_1x_2+x_3x_4+x_5x_6$; medium-hard expressions such as $0.5x_1(x_2^2+x_3^2+x_4^2)$ and $x_1x_2x_3(\frac{1}{x_4}-\frac{1}{x_5})$; and hard expressions like $\frac{x_1x_2x_3}{(x_4-x_5)^2+(x_6-x_7)^2+(x_8-x_9)^2}$ and $1+\frac{x_1x_2}{1-x_1x_2/3}$, providing a comprehensive coverage of the AIFeynman benchmark.

The comprehensive coverage of the AIFeynman benchmark is reflected in Figure 3b of the results section, showcasing the outcomes for all AI Feynman equations considered in the problem set. Notably, our model achieves a state-of-the-art 80% recovery rate, indicating its proficiency in capturing the underlying mathematical structures across a diverse range of expressions.

Table S7: Average Recovery Rate (%) of the AIFeynman Benchmark over 100 parallel runs

Name	Equation	Ours	SPL	uDSR	NGGP	DSR	GP
AIFeynman-1	x_1x_2	**100**	**100**	**100**	**100**	**100**	**100**
AIFeynman-2	$\frac{3}{2}x_1x_2$	**100**	99	97	**100**	97	87
AIFeynman-3	$x_1x_2x_3$	**100**	**100**	**100**	**100**	**100**	**100**
AIFeynman-4	$x_1x_2x_3\sin(x_4)$	**100**	98	90	**100**	**100**	78
AIFeynman-5	$x_1x_2+x_3x_4+x_5x_6$	**100**	**100**	**100**	**100**	**100**	82
AIFeynman-6	$x_1(1+x_2\cos(x_3))$	**100**	**100**	80	**100**	**100**	**100**
AIFeynman-7	$x_1x_2x_3(\frac{1}{x_4}-\frac{1}{x_5})$	**100**	**100**	87	**100**	**100**	80
AIFeynman-8	$x_1(x_2+x_3x_4\sin(x_5))$	**100**	**100**	**100**	**100**	**100**	**100**
AIFeynman-9	$x_1+x_2+2\sqrt{x_1x_2}\cos(x_3)$	**67**	0	8	7	0	0
AIFeynman-10	$\frac{1}{2}x_1(x_2^2+x_3^2+x_4^2)$	**15**	0	0	0	0	0
AIFeynman-11	$\frac{x_1x_2x_3}{(x_4-x_5)^2+(x_6-x_7)^2+(x_8-x_9)^2}$	0	0	0	0	0	0
AIFeynman-12	$1+\frac{x_1x_2}{1-x_1x_2/3}$	0	0	0	0	0	0
	Average	**73.50±24.1**	66.42±27.8	63.50±26.0	67.25±27.4	66.42±27.8	60.58±25.7

C.6 CONST-OPTIMIZATION EXPERIMENT

We conducted an experiment considering different initialization approaches for constant optimization. Specifically, we explored seven methods:

- Case 1. Initializing constants with a vector of ones.
- Case 2. Initializing constants with a vector of random uniform values between 0 and 1.
- Case 3. Initializing constants with a vector of random uniform values between 0.5 and 1.5.
- Case 4. Initializing constants with a vector of random Gaussian values with a mean of 0 and standard deviation of 1.
- Case 5. Initializing constants with a vector of random Gaussian values with a mean of 1 and standard deviation of 1.
- Case 6. Initializing constants with a vector of random Gaussian values with a mean of 1 and standard deviation of 0.5.
- Case 7. Initializing constants with a vector of random Gaussian values with a mean of 0 and standard deviation of $\frac{1}{3}$.

We evaluated the recovery rate of each expression using different constant initialization methods across diverse benchmarks, employing ranges of input values such as [0,1], [-1,1], [0,10], [-10,10], [0,50], [-50,50], and data sizes of 20 or 500. Notably, Table S8 demonstrates that the average recovery rates across all initializing methods are remarkably close.

Examining the distribution of expression recovery rates below 50%, 10%, and 0%, it becomes apparent that the method employing a vector of ones exhibits the highest percentage in Table S9. This observation indicates that the vector of ones initialization method has the highest number of expressions unable to converge across 100 parallel runs.

Table S8: Average Recovery Rate (%) of the Const Optimization Benchmark over 100 parallel runs on each size and each range

Name	Equation	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Keijzer-1	$0.3x_1 \sin(6.28x_1)$	33.6	33.8	33.6	33.7	33.7	33.6	**34.3**
Keijzer-4	$(30.0x_1x_2x_3)/((x_1-10.0)x_2^2)$	**34.2**	13.3	35.7	14.8	20.7	24.3	24.1
Keijzer-14	$8.0/(2.0+x_1^2+x_2^2)$	**100.0**	99.8	**100.0**	82.0	95.8	99.6	99.8
Keijzer-15	$x_1^{0.6}+x_2^{1.5}-x_2-x_1$	88.0	83.2	**90.3**	75.5	83.8	88.3	88.7
Nguyen-1^c	$3.39x_1^3+2.12x_1^2+1.78x_1$	97.2	**97.5**	**97.5**	96.8	95.9	96.9	97.4
Nguyen-2^c	$0.48x_1^4+3.39x_1^3+2.12x_1^2+1.78x_1$	**98.0**	92.3	97.9	92.9	95.0	97.5	96.2
Nguyen-5^c	$\sin(x_1^2)\cos(x_1)-0.75$	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Nguyen-7^c	$\log(x_1+1.4)+\log(x_1^2+1.3)$	**100.0**	53.4	78.4	50.6	56.0	70.4	97.1
Nguyen-8^c	$\sqrt{1.23x_1}$	**100.0**	**100.0**	**100.0**	99.5	**100.0**	**100.0**	**100.0**
Nguyen-9^c	$\sin(1.5x_1)+\sin(0.5x_2^2)$	35.3	**47.9**	43.4	42.6	43.8	45.6	46.3
Nguyen-10^c	$\sin(1.5x_1)\cos(0.5x_2)$	35.1	**47.7**	38.7	42.8	42.7	44.3	42.3
Nguyen-11^c	$2.7x_1^{x_2}$	99.2	88.2	98.5	79.3	89.3	95.8	**100.0**
Jin*-1	$2.5x_1^4-1.3x_1^3+0.5x_1^2-1.7x_1$	**98.3**	97.5	96.3	97.0	94.7	97.7	97.4
Jin*-2	$8.0x_1^3-8.0x_1^2+15.0x_1$	81.0	83.0	81.4	83.2	83.1	82.9	82.5
Jin*-3	$0.7x_1^3-1.7x_1$	**100.0**	54.1	92.2	52.9	76.1	83.3	87.9
Jin*-4	$1.5\exp(x_1)+5.0\cos(x_1)$	87.7	93.8	**94.6**	88.1	92.3	94.5	93.6
Jin*-5	$6.0\sin(x_1)\cos(x_1)$	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Jin*-6	$1.35x_1^2+5.5\sin((x_1-1)^2)$	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	Average	81.06	76.19	80.90	73.46	77.17	79.74	**81.08**

Table S9: Persentage (%) of expressions under certain recovery rate about the Const Optimization Benchmark over each initializing method

Recovery Rate	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
<50%	18.04	25.26	18.04	26.29	19.07	17.53	**17.01**
<10%	15.98	**11.34**	13.92	11.86	11.86	**11.34**	12.37
0%	7.22	4.12	6.19	3.61	4.64	3.61	**3.09**

C.7 TRADE-OFF EXPERIMENT

We used five different configurations for symbolic regression on the Nyugen dataset and obtained five different sets of results. We calculated the curves of the average number of tests about the expression recovery rate for all data/ Nyugen-4/ Nyugen-5/ Nyugen-11/ Nyugen-12 for the five different data with different configurations as shown in Figure S2.

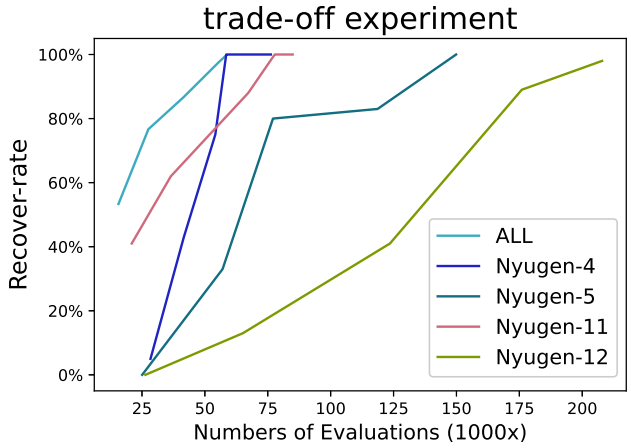

Figure S2: Trade-off between accuracy and number of evaluations in Nyugen Benchmark.

The evaluation count of the RMSM exhibits a discernible pattern, roughly falling within two ranges: the first encompasses around 80% of the recovery rate, while the second spans from 80% to 100%.

Table S10: Trade-off experiment: Average Evaluation Number /Average Recovery Rate (%) of the Nyugen Benchmark over 100 parallel runs

Name	5 epochs	15 epochs	25 epochs	35 epochs	45 epochs
ALL	15654/53	27543/77	41525/87	55395/98	58240/100
Nyugen-4	28389/5	41638/43	54151/75	58483/100	76216/100
Nyugen-5	25084/0	57033/33	77075/80	118661/83	149868/100
Nyugen-11	20932/41	36487/62	67237/88	77757/100	84902/100
Nyugen-12	26345/0	65122/13	123571/41	175970/89	207889/98

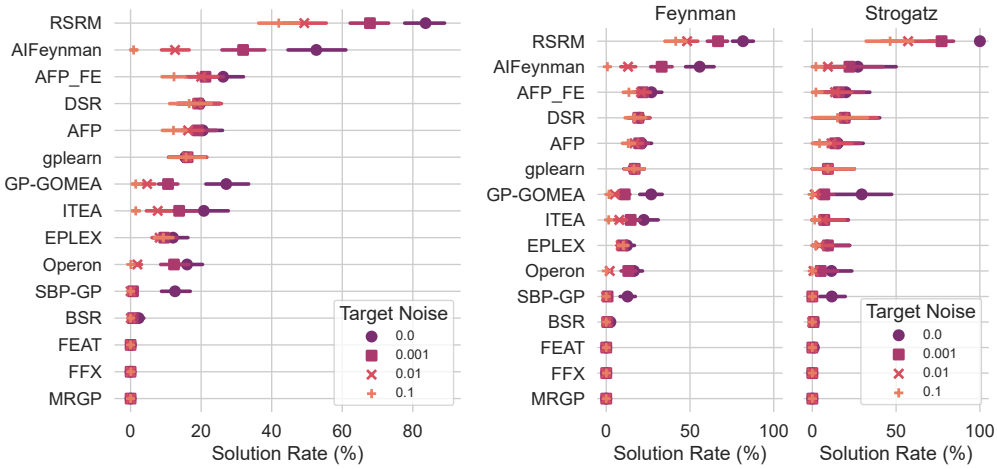

Figure S3: Result of SRBench with 10 parallel runs for each dataset.

Both ranges can be approximated with linear functions; however, the second range displays a notably steeper slope, indicating that achieving higher recovery rates beyond 80% becomes considerably more challenging.

We categorized RSRM into using 5, 15, 25, 35, 45 epochs and tested the average number of tests and the average recovery rate for each different environment and different data respectively in Table S10.

D SRBENCH RESULT

We used the full set of SRBench for testing, and the overall results are as follows in Figure S3, including both the AIFeynman dataset and the Strogatz dataset.

E FREE-FALLING BALLS DATASET RESULT

In this section, we provide additional details about the results obtained from the Falling-Balls dataset experiment. To improve the performance of the SPL model, we incorporated the operators $\log(\cosh(\cdot))$. This addition aimed to enhance the model's ability to capture the underlying patterns in the data.

The complete results of the experiment, including the functions found, can be found in Table S11.

Table S11: Functions generated in Falling-Balls Experiment

Name	Model	Equation
baseball	Ours	$-4.43t^2 + 0.36\sin(t^2 + 1.51)^2 + 47.35$
	Model A	$0.09t^3 - 5.47t^2 + 2.47t + 46.52 + \cos(t^2 - 2.5t)^{0.5}$
	SPL	$-4.54t^2 + 0.625t + 47.8$
blue basket ball	Ours	$-1.66t^3 - 4.95t^2\cos(\sqrt{t}) + 46.46$
	Model A	$-0.1t^3 - 4.49t^2 + 37.54t + 46.49 - t(\cos(t) + 36.77)$
	SPL	$-0.25t^4 + t^3 - 5.11t^2 + 46.47$
bowling ball	Ours	$-4.63t^2 + \sin(0.83t)\sin(t) + 46.13$
	Model A	$0.18t^3 - 6.0t^2 + 2.15t + 45.43 + \lvert t - 0.62\rvert$
	SPL	$-0.285t^3 - 3.82t^2 + 4.14 \times 10^{-5}\exp(20.74t^2 - 12.45t^3) + 46.1$
golf ball	Ours	$-0.09t^3 - 4.44t^2 + 5.26 \times 10^{-5}t/\log(t) + 49.51$
	Model A	$-2.18t^3 + 11.75t^2 + 1.96t + 25.86 - 2.36\exp(t) + 25.98\cos(t)$
	SPL	$-4.9633t^2 + \log(\cosh(t)) + 49.5087$
green basket ball	Ours	$46.34 - 4.15t^2$
	Model A	$-0.09t^3 - 4.59t^2 + 1.6t + 45.26 + (\frac{0.02\sqrt{t}}{t - \exp(\cos(t))} - t + 1)\cos(t)$
	SPL	$-4.1465t^2 + 45.9087 + \log(\cosh(1))$
tennis ball	Ours	$47.78\cos(0.43t - 0.02)$
	Model A	$0.33t^3 - 4.9t^2 + 0.66t + 47.74$
	SPL	$-4.0574t^2 + \log(\cosh(0.121t^3)) + 47.8577$
volleyball	Ours	$48.15 - 3.67(t + 0.03)^2$
	Model A	$1.59t^3 - 11.1t^2 + 0.93t + 58.53 - 10.53\cos(t)$
	SPL	$-3.78t^2 + 48.0744$
whiffle ball1	Ours	$-t^2(3.83 - 0.31t) + 47.07$
	Model A	$-0.08t^3 - 2.17t^2 - 1.69t + 46.29 + \sqrt{t + \sin(3t)}$
	SPL	$-t^3 + 4.16t^2 + 47.01\exp(-0.15t^2)$
whiffle ball2	Ours	$-2.18t^2 + 0.1t\cos(t) + 3.35\cos(t) + 43.88$
	Model A	$0.46t^3 - 4.39t^2 + 0.19t + 47.26 - 0.05\cos(\exp(t))$
	SPL	$65.86\exp(-0.0577t^2) - 18.61$
yellow whiffle ball	Ours	$(\cos(1.75t) + 47.59)\cos(0.36t)$
	Model A	$-0.27t^3 - 2.58t^2 - 2.5t + 48.25 + (t + 0.41)\exp(\sqrt{t + t^2 - 2\sqrt{t^3}})$
	SPL	$(148.99 - 14.58t^2 + 48.96\log(\cosh(x)))/(\log(\cosh(t)) + 3.065)$
orange whiffle ball	Ours	$-17.82t - 33.11/\exp(t)^{0.5} + 80.94$
	Model A	$0.42t^3 - 3.81t^2 - 1.4t + 47.84$
	SPL	$-1.66t + 47.86\exp(-0.0682t^2)$

F Generalization Experiment Result

To compare the generalization ability of our model with other methods, we conducted an experiment on generalization performance. The dataset was generated using the cumulative distribution function (CDF) defined as

$$F(x, \mu, \sigma) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt$$

with varying means (μ) and variances (σ). The dataset consisted of 201 points spanning the range from -100 to 100. Each dataset is divided into three subsets: a training set, a test set, and a validation set. The training set comprises points ranging from 30 to 80, while the test set consists of points ranging from 10 to 25. The validation set covers a broader range, spanning from 0 to 100.

Because this equation lacks an explicit elementary expression, it's impossible to obtain an analytical solution for the entire curve by reducing the training error to zero. Therefore, learning this function

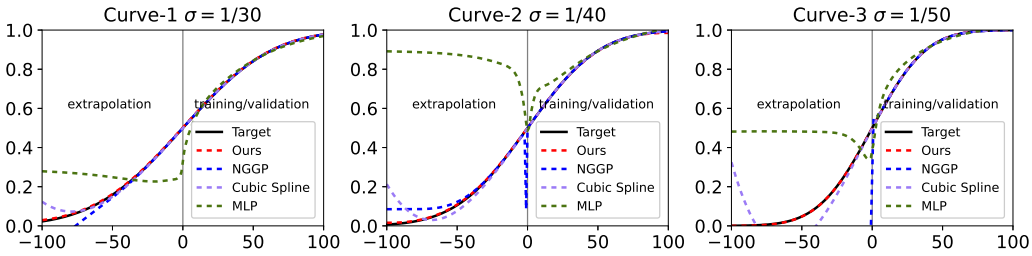

Figure S4: Result of generalization test experiment.

requires balancing between training error and generalization error, demanding a stronger ability to generalize. Moreover, only half of the curve data is provided, necessitating strong generalization skills from the model to extrapolate and accurately fit the missing portion of the curve. This calls for the proficiency of the model in capturing the distribution of the entire curve.

To evaluate the performance of our approach and compare it with baselines, we define the following settings for each method:

- **Ours**: The training set is utilized for generating expressions and calculating the corresponding rewards. The test set is employed to evaluate the quality of the generated expressions. Finally, the validation set is employed to select the most promising expressions from the outputs.
- **NGGP**(Mundhenk et al., 2021a): Both the training set and the test set are used for generating expressions and computing rewards. The HallOfFame, which contains the best expressions, is then leveraged to choose expressions using the validation set.
- **Linear regression**: The training set and the test set are employed for training the linear regression model.
- **Cubic splines**: The training set and the test set are used to train the cubic spline model.
- **Deep learning**: In the deep learning approach, we employ a Multilayer Perceptron (MLP) architecture with one input, one output, and a hidden layer ranging in size from 30 to 50. We set the learning rate to 10^{-3} and train 100 epochs. We experiment with different configurations of the hidden layer and select the model that yields the best performance. The MLP is trained using the training set, and the test set is used to evaluate the performance of each model. By varying the size of the hidden layer, we aim to find the optimal architecture that achieves the highest accuracy or lowest error on the given task.

The full results of the generalization experiment can be found in Table S12 and Figure S4. This table presents a detailed overview of the performance of the model and the baselines. Additionally, Table S13 presents the equations discovered by the model and the baselines. These tables demonstrate that our model outperforms the baseline methods in terms of generalization ability. The curves fitted by our model exhibit better accuracy and capture the underlying patterns in the data more effectively.

G Parity Determination Experiment

In this section, we conduct an experiment to compare the efficiency and effectiveness of form discovery by AIFeynman(Udrescu & Tegmark, 2020) and our method.

According to AIFeynman (Udrescu & Tegmark, 2020), it is also possible to use MLP as a learning curve to determine the parity of a function, so we compared the efficiency and effectiveness of MLP and cubic splines in determining parity.

The configuration of the MLP follows the structure employed in AIFeynman. It consists of a five-layer neural network with a balanced training set and validation set ratio of 5:5. The input layer accepts one variable and yields an output of 128. Subsequent to the input layer, the hidden layers encompass input-output feature pairs of 128-128, 128-64, and 64-64. The output layer produces a single variable. Optimization is executed using the Adam optimizer, and the activation function for each layer is set

Table S12: Mean Squared Error (MSE) of each method and each part of the curve in the Generalization Experiment

Name	Ours	NGGP	Linear	Cubic Splines	MLP
total error on curve 1	$\mathbf{1.05 \times 10^{-5}}$	0.00215	0.0114	0.000381	0.0142
total error on curve 2	$\mathbf{9.79 \times 10^{-6}}$	0.00163	0.0297	0.00162	0.261
total error on curve 3	$\mathbf{2.61 \times 10^{-7}}$	0.327	0.0821	0.00563	0.0762
extrapolation error on curve 1	$\mathbf{1.65 \times 10^{-5}}$	0.00429	0.0215	0.000758	0.0278
extrapolation error on curve 2	$\mathbf{1.41 \times 10^{-5}}$	0.00324	0.0565	0.00323	0.518
extrapolation error on curve 3	$\mathbf{2.67 \times 10^{-7}}$	0.65	0.158	0.0112	0.151
validation error on curve 1	4.46×10^{-6}	$\mathbf{2.59 \times 10^{-10}}$	0.00129	2.76×10^{-10}	0.000532
validation error on curve 2	5.45×10^{-6}	$\mathbf{1.08 \times 10^{-10}}$	0.00265	2.87×10^{-9}	0.00177
validation error on curve 3	2.55×10^{-7}	2.94×10^{-5}	0.00518	$\mathbf{3.75 \times 10^{-8}}$	0.00106
test error on curve 1	6.95×10^{-6}	9.09×10^{-12}	0.000545	$< \mathbf{1 \times 10^{-12}}$	0.000253
test error on curve 2	2.25×10^{-7}	$< \mathbf{1 \times 10^{-12}}$	0.00127	$< \mathbf{1 \times 10^{-12}}$	0.00344
test error on curve 3	8.8×10^{-8}	$< \mathbf{1 \times 10^{-12}}$	0.00272	$< \mathbf{1 \times 10^{-12}}$	0.00358
training error on curve 1	3.88×10^{-6}	3.57×10^{-12}	0.000254	$< \mathbf{1 \times 10^{-12}}$	6.93×10^{-5}
training error on curve 2	1.26×10^{-6}	$< \mathbf{1 \times 10^{-12}}$	0.000524	$< \mathbf{1 \times 10^{-12}}$	8.6×10^{-5}
training error on curve 3	3.34×10^{-7}	7.14×10^{-12}	0.000905	$< \mathbf{1 \times 10^{-12}}$	7.9×10^{-5}

Table S13: Functions generated in Generalization Experiment. Our functions are easier to calculate and shorter than NGGP's.

Name	Model	Equation
curve-1	Ours	$0.503 + (117.088x)/(x^2 + 14702)$
	NGGP	$\cos(\exp((0.49x \log(0.028x + 29.5/(0.039x + 9.82)) - 2.78)/(0.115x - 60.5)))$
curve-2	Ours	$(6.08x + 0.785)/(0.0639x^2 + 615.179) + 0.50003$
	NGGP	$\cos(2.95 \exp(-0.68 \exp(0.41 \exp(5.5x \exp(24.67/(115.6 \exp((4.75x + 13.3)/x) + 3.2))/(2x + 214.3)))))$
curve-3	Ours	$(x \sin(371.57/(13928/x + x)) + x)/(0.0024 + 2x)$
	NGGP	$\cos(\log(1 + 1.67 \exp(-1.56/(\exp(17.7 \exp(\exp((-12.9 + 4.95 \log(x)/x)/x))/x) - 1.16 + 0.656/x))))$

to the hyperbolic tangent (tanh). The training process encompasses 2000 rounds, initiated with a learning rate of 0.01, which is subsequently reduced by a factor of 0.1 whenever the loss increases.

We used seven functions, the top three are odd functions, from easy to hard, then the next three are even functions, and the last one is a non-odd non-even function.

We first tested the speed of both methods. The cubic spline method takes 0.216, 0.217, 0.219, 0.220, 0.226 milliseconds to run on 10, 100, 1000, 10000, and 100000 data points, respectively. Correspondingly, MLP takes 3.28, 3.64, 8.97, 64.1, 549.7 seconds to run.

We then tested two loss equations for the corresponding functions, as shown in the Table S14, and the loss functions are as follows: $L_{odd} = \sum_{i=1}^{n}(y(x) + y(-x))^2/n, L_{even} = \sum_{i=1}^{n}(y(x) - y(-x))^2/n$. It can be seen in Figure S5 that if 10^{-4} of MSE is used as the cutoff for whether it is an odd/even function or not, the amount of data required by MLP is about 100–1000 times more than that of the spline.

H Ablation Experiment Result

In this section, we provide more detailed information about the results obtained from the ablation experiment on LiverMore benchmark. The full results of the ablation experiment can be found in Table S15. This table presents a comprehensive overview of the performance of the model under different ablation settings.

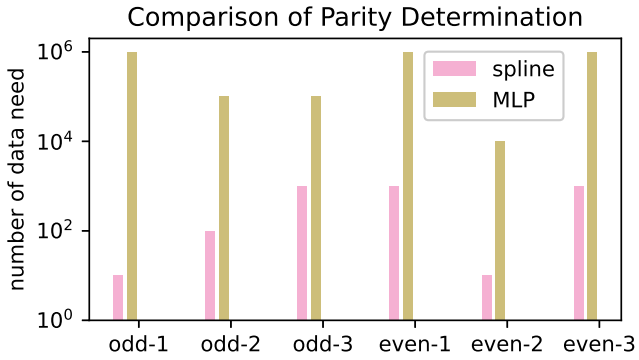

Figure S5: The parity determination performance test. We test the ability of parity determination of two models by three odd functions and even functions from easy to hard. Compete Setting of this experiment is shown in Table S14.

Table S14: The average loss results for the Parity Determination Experiment across 10 parallel runs are presented in the table below. Each cell in the table contains two values: the upper value, situated above the horizontal line, signifies the loss value of the MLP, while the lower value indicates the loss of the cubic splines.

Name	Equation	Input Range	10	100	1000	10000	100000
odd-1	x	$[-1,1]$	$\frac{2.84\times10^{-1}}{1.99\times10^{-16}}$	$\frac{2.57\times10^{-3}}{5.58\times10^{-17}}$	$\frac{1.15\times10^{-3}}{2.78\times10^{-17}}$	$\frac{5.88\times10^{-4}}{3.47\times10^{-18}}$	$\frac{1.21\times10^{-4}}{1.35\times10^{-20}}$
		$[-5,5]$	$\frac{3.04\times10^{-1}}{1.40\times10^{-15}}$	$\frac{3.31\times10^{-3}}{3.18\times10^{-17}}$	$\frac{3.37\times10^{-3}}{0.00\times10^{0}}$	$\frac{9.79\times10^{-4}}{1.55\times10^{-17}}$	$\frac{1.22\times10^{-4}}{4.85\times10^{-19}}$
odd-2	$x+\sinh(x)+x^3$	$[-1,1]$	$\frac{2.15\times10^{-1}}{5.59\times10^{-4}}$	$\frac{2.04\times10^{-3}}{7.02\times10^{-7}}$	$\frac{1.58\times10^{-3}}{1.77\times10^{-9}}$	$\frac{1.40\times10^{-4}}{4.51\times10^{-13}}$	$\frac{1.11\times10^{-4}}{2.16\times10^{-12}}$
		$[-5,5]$	$\frac{7.70\times10^{-1}}{2.85\times10^{0}}$	$\frac{5.37\times10^{-2}}{2.05\times10^{-3}}$	$\frac{4.53\times10^{-2}}{5.22\times10^{-7}}$	$\frac{1.17\times10^{-3}}{2.70\times10^{-10}}$	$\frac{3.96\times10^{-4}}{1.99\times10^{-12}}$
odd-3	x^3+x+x^5+ $\sin(x)\times\cosh(x)$	$[-1,1]$	$\frac{1.64\times10^{-1}}{3.10\times10^{-2}}$	$\frac{1.09\times10^{-3}}{8.76\times10^{-5}}$	$\frac{5.76\times10^{-4}}{2.52\times10^{-8}}$	$\frac{2.38\times10^{-4}}{8.57\times10^{-12}}$	$\frac{8.04\times10^{-5}}{4.00\times10^{-11}}$
		$[-5,5]$	$\frac{7.60\times10^{-2}}{3.45\times10^{2}}$	$\frac{7.07\times10^{-4}}{8.16\times10^{-2}}$	$\frac{7.95\times10^{-4}}{6.28\times10^{-5}}$	$\frac{1.55\times10^{-4}}{2.75\times10^{-8}}$	$\frac{6.95\times10^{-5}}{1.62\times10^{-9}}$
even-1	x^2	$[-1,1]$	$\frac{2.60\times10^{-1}}{2.64\times10^{-14}}$	$\frac{4.33\times10^{-3}}{5.11\times10^{-15}}$	$\frac{7.44\times10^{-4}}{3.01\times10^{-15}}$	$\frac{1.59\times10^{-4}}{1.78\times10^{-13}}$	$\frac{8.76\times10^{-5}}{7.81\times10^{-13}}$
		$[-5,5]$	$\frac{1.85\times10^{-1}}{1.33\times10^{-13}}$	$\frac{2.13\times10^{-3}}{2.73\times10^{-13}}$	$\frac{6.02\times10^{-4}}{1.38\times10^{-13}}$	$\frac{3.53\times10^{-4}}{1.20\times10^{-12}}$	$\frac{2.22\times10^{-4}}{1.95\times10^{-11}}$
even-2	$x\times\sinh(x)$	$[-1,1]$	$\frac{7.25\times10^{-2}}{2.43\times10^{-4}}$	$\frac{1.20\times10^{-2}}{2.93\times10^{-7}}$	$\frac{9.35\times10^{-4}}{5.77\times10^{-10}}$	$\frac{2.65\times10^{-4}}{3.23\times10^{-13}}$	$\frac{7.32\times10^{-5}}{2.70\times10^{-12}}$
		$[-5,5]$	$\frac{9.94\times10^{-1}}{5.54\times10^{0}}$	$\frac{7.90\times10^{-1}}{1.58\times10^{-3}}$	$\frac{8.06\times10^{-1}}{1.55\times10^{-5}}$	$\frac{2.88\times10^{-1}}{1.00\times10^{-9}}$	$\frac{3.15\times10^{-4}}{2.02\times10^{-10}}$
even-3	$x^4+\log(x^2+1)+$ $\cos(x)\times\exp(0.1x^2)$	$[-1,1]$	$\frac{1.09\times10^{-1}}{3.68\times10^{-2}}$	$\frac{2.61\times10^{-3}}{7.85\times10^{-6}}$	$\frac{1.52\times10^{-3}}{1.53\times10^{-8}}$	$\frac{3.09\times10^{-4}}{3.20\times10^{-12}}$	$\frac{1.48\times10^{-4}}{6.55\times10^{-12}}$
		$[-5,5]$	$\frac{9.49\times10^{-1}}{5.20\times10^{0}}$	$\frac{9.30\times10^{-1}}{3.61\times10^{-2}}$	$\frac{9.21\times10^{-1}}{2.10\times10^{-5}}$	$\frac{6.65\times10^{-1}}{2.54\times10^{-9}}$	$\frac{7.76\times10^{-4}}{3.09\times10^{-10}}$
none	$x^3+\log(x^2+1)+$ $x^7+\sinh(x)$	$[-1,1]$	$\frac{9.90\times10^{-1}}{1.68\times10^{0}}$	$\frac{4.99\times10^{-1}}{7.05\times10^{0}}$	$\frac{5.13\times10^{-1}}{2.23\times10^{1}}$	$\frac{5.13\times10^{-1}}{6.81\times10^{1}}$	$\frac{5.16\times10^{-1}}{2.16\times10^{2}}$
		$[-5,5]$	$\frac{9.11\times10^{-1}}{6.95\times10^{2}}$	$\frac{9.16\times10^{-1}}{4.22\times10^{1}}$	$\frac{9.22\times10^{-1}}{1.30\times10^{2}}$	$\frac{6.81\times10^{-1}}{4.16\times10^{2}}$	$\frac{4.20\times10^{-3}}{1.31\times10^{3}}$

Table S15: Average Recovery Rate (%) of the Ablation Experiment over 100 parallel runs

Name	Equation	Ours	ModelA	ModelB	ModelC	ModelD
Livermore-1	$1/3 + x_1 + \sin(x_1)$	**100**	**100**	**100**	**100**	**100**
Livermore-2	$\sin(x_1^2)\cos(x_1) - 2$	**100**	**100**	**100**	6	**100**
Livermore-3	$\sin(x_1^3)\cos(x_1^2) - 1$	**55**	20	0	0	**55**
Livermore-4	$\log(x_1 + 1) + \log(x_1^2 + x_1) + \log(x_1)$	**100**	**100**	**100**	**100**	**100**
Livermore-5	$x_1^4 - x_1^3 + x_1^2 - x_2$	**100**	**100**	**100**	**100**	**100**
Livermore-6	$4x_1^4 + 3x_1^3 + 2x_1^2 + x_1$	**100**	**100**	**100**	**100**	**100**
Livermore-7	$\sinh(x_1)$	**100**	**100**	**100**	**100**	10
Livermore-8	$\cosh(x_1)$	**100**	**100**	**100**	**100**	3
Livermore-9	$\sum_{i=1}^{9} x_1^i$	**100**	83	**100**	88	**100**
Livermore-10	$6\sin(x_1)\cos(x_2)$	**100**	**100**	**100**	**100**	**100**
Livermore-11	$(x_1^2 x_2^2)/(x_1 + x_2)$	**100**	91	**100**	**100**	**100**
Livermore-12	x_1^5/x_2^3	**100**	**100**	**100**	**100**	**100**
Livermore-13	$x_1^{1/3}$	**100**	**100**	**100**	67	**100**
Livermore-14	$x_1^3 + x_1^2 + x_1 + \sin(x_1) + \sin(x_1^2)$	**100**	**100**	**100**	**100**	**100**
Livermore-15	$x_1^{1/5}$	**100**	**100**	**100**	97	**100**
Livermore-16	$x_1^{2/5}$	**100**	**100**	**100**	12	**100**
Livermore-17	$4\sin(x_1)\cos(x_2)$	**100**	**100**	**100**	**100**	**100**
Livermore-18	$\sin(x_1^2)\cos(x_1) - 5$	**100**	89	90	0	**100**
Livermore-19	$x_1^5 + x_1^4 + x_1^2 + x_1$	**100**	**100**	**100**	**100**	**100**
Livermore-20	$\exp(-x_1^2)$	**100**	**100**	**100**	**100**	**100**
Livermore-21	$\sum_{i=1}^{8} x_1^i$	**100**	**100**	**100**	**100**	**100**
Livermore-22	$\exp(-0.5x_1^2)$	**100**	**100**	**100**	**100**	**100**
	Average	**97.95±4.0**	94.68±7.2	95.00±8.9	80.45±15.6	89.45±11.9