FunctionEvolve: Structure-Guided Symbolic Regression with LLMs

Zeyu Xia[1,2] **Jun Zhu**[2,3] **Dong Yan**[1]*
[1]Bosch Center for Artificial Intelligence
[2]Department of Computer Science and Technology, Tsinghua University
[3]Tsinghua-Bosch Joint Center for ML, Tsinghua University
xia-zy25@mails.tsinghua.edu.cn
dcszj@mail.tsinghua.edu.cn
sproblvem@gmail.com

Abstract

Symbolic regression aims to uncover explicit scientific laws from data. Recent methods use LLMs to guide mutation from background text, which is more directed than random genetic programming. However, exact symbolic recovery requires both semantic guidance and explicit structure, so that domain-informed search operates over valid symbolic representations. Current LLM-driven systems remain structure-blind: they select among opaque candidates, lack explicit mechanisms for local mutation, and rely on brittle coefficient fitting that can undervalue correct skeletons. We propose FunctionEvolve, an evolutionary framework using expression trees to organize the whole search: structural summaries promote diverse parent selection, local tree edits preserve useful subexpressions, and structure-aware fitting decomposes, constrains, and simplifies coefficient optimization for more reliable scoring. It uses only elementary function families, without additional domain-specific rules limiting generalization. On the 129-task synthetic subset of LLM-SRBench, FunctionEvolve with *Claude Opus 4.6* recovers 107 exact forms, reaching 82.9% SA@50, 4.5x above same-backbone baselines, and 55.8% SA@1, 3.6x above the strongest previously published top-1 result. Ablations show that structure-visible search is central to reliable recovery, with LLM-guided refinements and structure-aware coefficient optimization serving as essential proposal and scoring mechanisms. We also audit the benchmark and show that collinearity in its materials-science subset creates identifiability issues.

1 Introduction

Scientific discovery increasingly requires methods that infer interpretable laws from experimental data [Kramer et al., 2023]. Black-box models can fit observations well, but they often obscure the mechanisms governing natural phenomena. Symbolic Regression (SR) addresses this gap by recovering explicit mathematical expressions from data [Langley, 1981], enabling interpretation, extrapolation, and knowledge transfer [Schmidt and Lipson, 2009].

SR searches a combinatorial space of formulas, and genetic programming (GP) remains a standard way to represent expressions as abstract syntax trees (ASTs) and evolve them through local operators [Koza, 1994]. However, SR is NP-hard [Virgolin and Pissis, 2022], and GP driven mainly by random variation and data-fit selection often lacks the scientific priors needed to navigate large hypothesis spaces efficiently. Recent LLM-driven methods therefore combine model knowledge with background information, such as variable meanings and descriptions of the scientific process being modeled, to

*Corresponding author.

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

guide equation discovery. However, new synthetic benchmarks show that semantic guidance alone still leaves a large gap in exact symbolic recovery [Shojaee et al., 2025].

This gap prompts a necessary re-examination of current SR methodologies. We identify three shared bottlenecks in traditional GP and its LLM-driven variants. First, the search often starts from generic seeds rather than useful structural priors. Second, parent selection is usually driven by fixed heuristics over opaque candidates, making it difficult to allocate search budget across genuinely different skeletons. Third, unreliable coefficient optimization can make structurally correct skeletons appear unpromising, causing them to be discarded prematurely.

Explicit expression structure offers a direct way to address these bottlenecks, especially for mutation, where progress often comes from local edits rather than full rewrites. For example, if the current candidate is $a_1 \sin(x) + a_2 x$ and the target is $b_1 \sin(b_2 x + b_3) + b_4 x^{b_5}$, the remaining progress mainly requires editing the sine argument and the power term while preserving the rest. This is precisely where structure matters. GP can modify subexpressions without rewriting the whole formula, but its edits are largely random and do not exploit task context. Prior LLM-driven systems are more directed, but their textual or code-level edits are not necessarily AST-local, so semantically plausible revisions can still overwrite useful parent structure. FunctionEvolve therefore makes AST structure explicit and operative: AST-rule edits provide systematic local add/delete operations, while LLM-guided proposals use the exposed tree structure to make context-dependent refinements that preserve useful parent substructures. Beyond mutation, the same explicit AST structure helps the Selector enforce structural diversity and gives the structure-aware coefficient optimizer the skeleton needed to separate linear from nonlinear parameters, snap structurally constrained exponents, and simplify algebraically equivalent forms, reducing coefficient-search difficulty.

Core insight: Explicit AST structure makes mutation follow SR's local-editing nature and also benefits parent-diversity selection and coefficient fitting.

Building on this insight, we develop FunctionEvolve as an equation-discovery framework organized around AST-visible search. Our contributions are as follows:

- We present FunctionEvolve, a structure-guided LLM framework for equation discovery that uses explicit expression trees to coordinate selection, mutation, and coefficient optimization.
- On the 129-task synthetic subset of LLM-SRBench, FunctionEvolve achieves 82.9% $\mathrm{SA@50}$ and 55.8% $\mathrm{SA@1}$ with *Claude Opus 4.6*, corresponding to a 4.5x improvement over same-backbone baselines and a 3.6x top-1 improvement over strongest published PiT-PO.
- Ablations show that structure-visible search, LLM-guided local refinements, and structure-aware coefficient optimization are all critical for reliable symbolic recovery.
- We identify critical limitations in LLM-SRBench, including input-variable linear dependence and weak nonlinear factor variation that undermine structural identifiability.

2 Related Work

Modern SR spans symbolic neural networks, reinforcement learning, physics-inspired methods, and evolutionary systems [Martius and Lampert, 2016, Sahoo et al., 2018, Petersen et al., 2021, Udrescu and Tegmark, 2020, Cranmer, 2023]. Among them, GP-style methods remain strong baselines in comparisons [Cava et al., 2021]: they represent expressions as trees [Koza, 1994], enabling structured variation operators such as context-preserving, grammar-based, and strongly typed GP [D'haeseleer, 1994, Whigham, 1995, Montana, 1995]. Related work on prior-guided seeding, diversity, and coefficient fitting shows that symbolic recovery benefits from both expression-structure search and reliable numerical parameter identification [Lu et al., 2015, Mundhenk et al., 2021, Burlacu et al., 2023, Kommenda et al., 2020, dos Reis et al., 2024].

Recent work applies LLMs to SR by using model knowledge and task context to guide equation proposal, refinement, or selection. Representative systems include LLM-SR, LLM4Ed, DRSR, SR-Scientist, PiT-PO, and LaSR [Shojaee et al., 2024, Du et al., 2024, Wang et al., 2025, Xia et al., 2025, Wang et al., 2026, Grayeli et al., 2024]; evolutionary coding agents such as OpenEvolve, AlphaEvolve, and ShinkaEvolve also demonstrate LLM-guided program evolution [Sharma, 2025, Novikov et al., 2025, Lange et al., 2025]. These methods add semantic guidance beyond random GP, but often treat expressions as strings, programs, or opaque candidates rather than explicit expression trees shared across selection, mutation, and fitting.

The rise of LLM-based SR also makes benchmark design more important. Literature-derived benchmarks such as AI-Feynman, SRBench, and EmpiricalBench are valuable but may overlap with LLM pretraining data [Udrescu and Tegmark, 2020, Cava et al., 2021, Cranmer, 2023]. LLM-SRBench reduces this risk with synthetic scientific tasks and shows that existing LLM-driven SR methods still struggle with exact symbolic recovery [Shojaee et al., 2025].

3 Method

3.1 Problem Formulation

We consider LLM-driven symbolic regression over a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ together with a task-level context T, where $\mathbf{x}_i \in \mathbb{R}^d$ is the numerical input and $y_i \in \mathbb{R}$ is the target. The natural-language context T, shared across samples, may describe the scientific scenario, data source, and variable names, meanings, and units; dimension-only AI-Feynman tasks provide only names and units. The goal is to recover a concise, generalizable expression $\tilde{f}$ with $\tilde{f}(\mathbf{x}_i) \approx y_i$, using T to guide the search.

3.2 Overview

Our method formulates SR as one-shot initialization followed by iterative evolution. FunctionEvolve first uses the Generator to synthesize task-conditioned domain knowledge and seed expressions. It then uses the Selector to choose promising parents, the Mutator to expand them through AST-rule and LLM-guided mutations, and the structure-aware Optimizer to fit, simplify, and score valid offspring before reinsertion. Unlike prior LLM-driven methods, FunctionEvolve uses LLMs for seed generation, parent selection, and mutation, with each part anchored to explicit expression structure. The workflow is illustrated in Figure 1, and the complete algorithm is in Appendix C1.

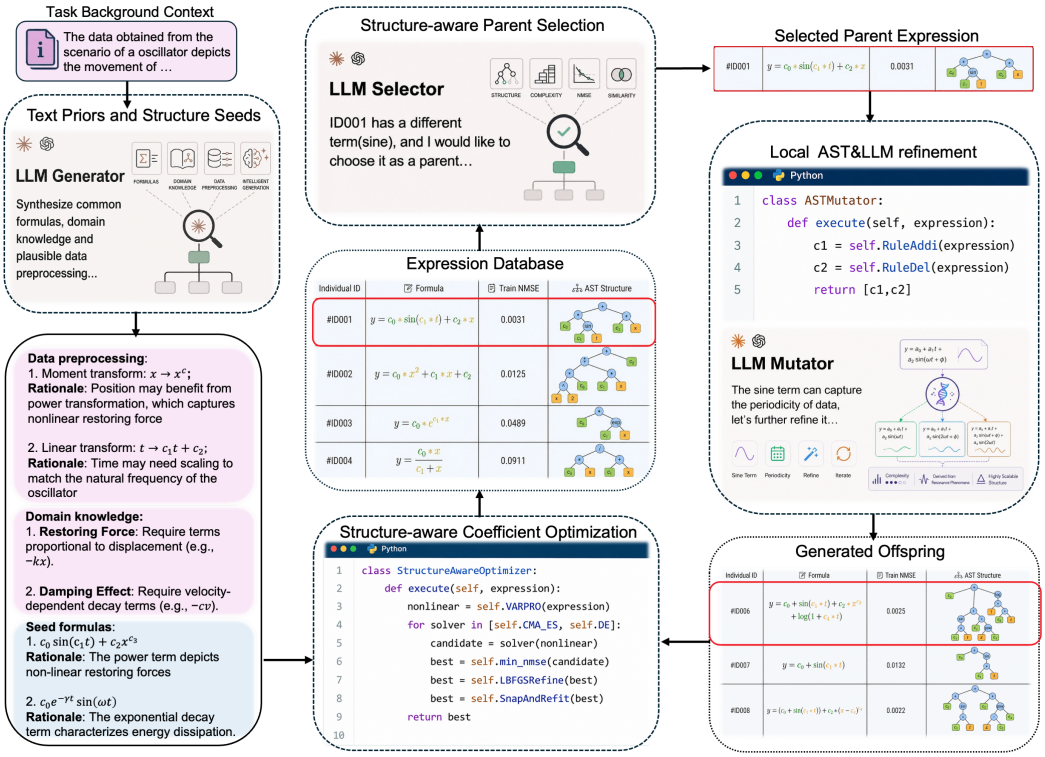


Figure 1: FunctionEvolve initializes candidate expressions from task context and then iteratively selects, mutates, optimizes, and evaluates them over an evolving expression database.

3.3 Generator

The Generator provides task-conditioned structural starting points through two roles: (1) it uses the available task context and LLM knowledge to synthesize domain formulas and structural heuristics; and (2) it initializes seed expressions from these patterns so the search does not begin from an unstructured pool. These outputs provide initial starting points and reusable contextual guidance

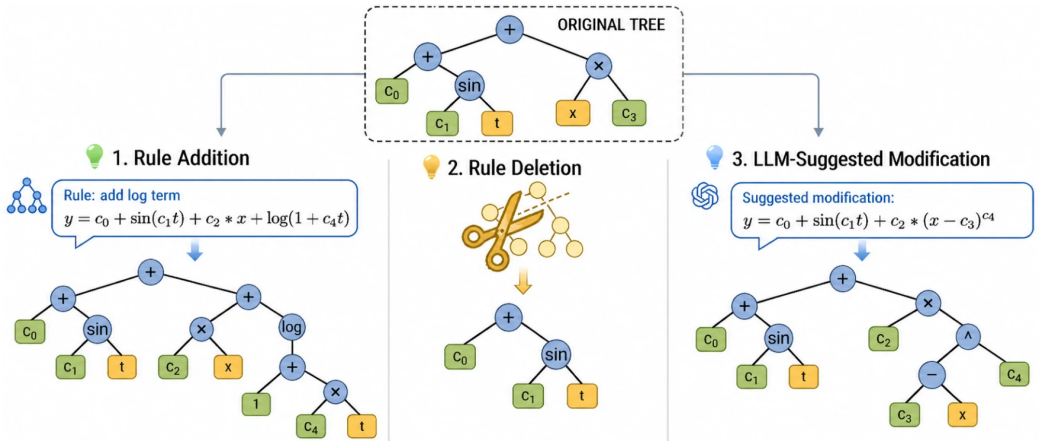


Figure 2: AST rule add/del and LLM-guided local mutation generate offspring from selected parents.

for evolution, rather than encoded domain laws or fixed mutation rules. Details of the Generator procedure and prompt are provided in Appendix C2.

3.4 Selector

The Selector determines which expressions to expand next. Each iteration, the system extracts an evolution-tree summary and provides it to the Selector with the task context and previous selections. For each candidate node, this summary combines structural information (AST structure, parameter count, tree depth, and operator counts) with numerical information (training NMSE and fitted parameter values). Based on this information, the Selector returns a small set of parent nodes, allocating limited search budget to more promising directions. Details of the evolutionary tree and Selector are provided in Appendices C3 and C4, respectively.

The Selector is not purely greedy: it favors low-error, simple candidates while using selection history to avoid repeatedly expanding similar branches. To prevent data leakage, the tree summary exposes only training error, never test or out-of-distribution error.

3.5 Mutator

The Mutator generates candidates by locally adjusting parent nodes selected by the Selector. For each selected parent expression $f(\mathbf{x}; \mathbf{c})$, it applies three complementary operations, as illustrated in Figure 2: (1) *Rule addition*, which preserves the parent and attaches fixed elementary functions (Table 1) through a small set of attachment operators detailed below; (2) *Rule deletion*, which prunes or unwraps AST substructures to test whether current formula parts are redundant; and (3) *LLM-guided proposal*, which uses parent structure, fitted information, rule-generated candidates, search history, and domain knowledge to suggest non-template, context-dependent refinements.

Table 1: Elementary content library used by rule-addition. For multivariate tasks, single-variable templates instantiate for each x_j, while pair families use ordered variable pairs (x_i, x_j).

Family	Library content g	Multivariate handling
Linear	$c_1 x_j + c_2$	–
Trigonometric	$c_1 \sin(c_2 x_j + c_3)$	–
Power	$c_1 x_j^{c_2}$	$c_1 (x_i + c_2)^{c_3} x_j$
Exponential	$c_1 \exp(c_2 x_j)$	–
Logarithmic	$c_1 \log(1 + c_2 x_j)$	–
Rational-fraction	$(c_1 x_j + c_2)/(c_3 x_j + c_4)$	$(c_1 x_i + c_2)/(c_3 x_j + c_4)$

The deterministic library uses only generic elementary content and common algebraic attachment operators, without domain-specific rules. Rule addition combines this content with the parent through addition, multiplication, and division; unary wraps additionally apply $\exp(cf)$, $\log(1 + cf)$, $\sin(cf)$, $|cf|$, and f^c. After mutation, candidates are normalized and deduplicated by structural fingerprints before entering the expression database. Details of the AST representation, rule-based and LLM-guided mutation, and normalization procedures are provided in Appendix C5.

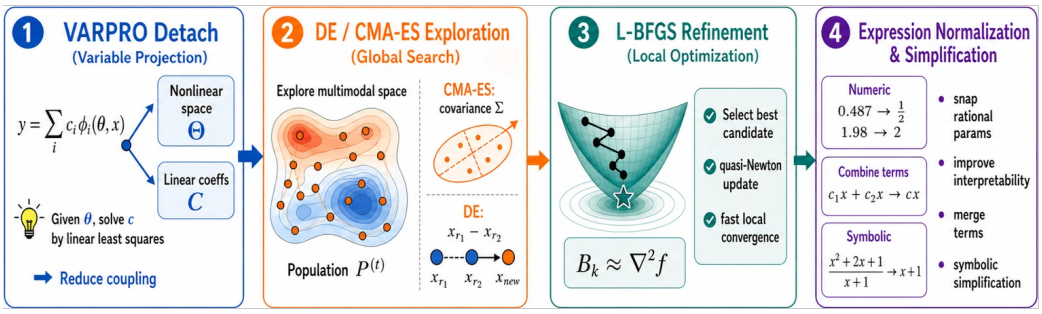


Figure 3: Structure-aware optimizer pipeline: variable projection, DE/CMA-ES global exploration, L-BFGS local refinement, parameter snapping, and symbolic simplification.

3.6 Optimizer

The Optimizer fits coefficients for each fixed candidate skeleton and returns its score. Rather than treating coefficient fitting as black-box optimization, FunctionEvolve exploits expression structure throughout the pipeline (Figure 3). Coefficients at different structural positions carry algebraic priors: linear parameters can be solved by variable projection and OLS, remaining nonlinear parameters are searched with global and local optimizers, structurally constrained parameters can be snapped to stable integer or rational values, and redundant forms can be removed through coefficient merging and symbolic simplification before refitting. These priors reduce redundant dimensionality and make correct skeletons less likely to be discarded because of poor exploration or local optima. They are generic SR operations and therefore do not encode benchmark shortcuts. See Appendix C6 for details.

4 Experiments

4.1 Main Results

The main comparison in this subsection is conducted on the synthetic benchmark set of LLM-SRBench [Shojaee et al., 2025]. We choose this benchmark because its synthetic construction reduces the risk of contamination from memorized formulas in LLM pretraining, thereby providing a cleaner test bed for the primary symbolic-regression comparison. No web search or external tools were used in the search-stage LLM calls.

Symbolic regression is commonly evaluated using three complementary metrics: (1) **Symbolic accuracy at** k **(**$\mathrm{SA@}k$**)**, for which candidates are ranked by training NMSE, each task records its first matching rank, and the task counts toward $\mathrm{SA@}k$ exactly when that rank is at most k; symbolic matches are usually evaluated using an LLM-as-a-judge protocol; (2) **Accuracy to tolerance** Acc_τ, the number of tasks whose top-1 candidate reaches maximum relative error over all evaluation points at most τ, where τ is typically set to 0.1 or 0.01; and (3) **Normalized mean squared error (NMSE)**, a metric that measures aggregate numerical error.

Since $\mathrm{SA@}k$ is calculated based on LLMs, we further audit the reliability of its LLM-as-a-judge protocol. On 500 sampled verification cases, *GPT-5.2* and *Claude Opus 4.6* agree on 476 decisions; manual auditing over these 500 cases gives judge accuracies of 96.4% and 98.6%, respectively. In the actual experiments, each candidate is verified by both models: agreeing verdicts are accepted, while disagreements are manually adjudicated (See Appendix D9).

Table 2 reports symbolic accuracy and numerical accuracy for each domain and for the full 129-task benchmark. We emphasize both SA@50 and SA@1: SA@50 measures whether the search generates a correct symbolic form within its top-50 training-NMSE-ranked candidates, while SA@1 measures whether the method's single selected expression already recovers the target structure. Numerical metrics are complementary: NMSE can be saturated by structurally incorrect surrogate expressions, and worst-point Acc_τ can be sensitive to small numerical errors when target values are close to zero. The ground-truth row is therefore included only as a dataset reference ceiling, not as a competing method.[2]

[2]An additional baseline, SR-Scientist [Xia et al., 2025], is compared separately in Appendix D8 rather than in this table: its headline Acc_τ discards the worst 5% of evaluation points before taking the maximum, making it a relaxed variant aligned with our $95\%\mathrm{Acc}_\tau$ rather than our strict all-point Acc_τ, and it reports symbolic accuracy only for a few representative methods, without an $\mathrm{SA@}k$ analysis.

Table 2: Main results on the 129-task synthetic subset of LLM-SRBench. Entries report SA@50 (SA@1), $\mathrm{Acc}_{0.1}$ task counts, and median test NMSE; table headers abbreviate these as SA, Acc, and NMSE. Parenthesized-only SA values are published top-1 results without reported SA@50. Bold marks the best non-reference values, while the ground-truth row is a reference upper bound. * and ** denote results cited from LLM-SRBench [Shojaee et al., 2025] and PiT-PO [Wang et al., 2026].

	Chemistry (36)			Biology (24)			Physics (44)			MatSci (25)			Total (129)		
Model	SA↑	Acc↑	NMSE↓	SA↑	Acc↑	NMSE↓	SA↑	Acc↑	NMSE↓	SA↑	Acc↑	NMSE↓	SA↑	Acc↑	NMSE↓
Direct Prompting															
*GPT-4o-mini**	(0)	5	0.0221	(0)	1	0.4648	(2)	4	0.0647	(0)	0	0.0484	(2)	10	–
Claude Opus 4.6	0 (0)	3	1.50e-2	1 (1)	3	2.43e-4	0 (0)	4	2.18e-3	1 (0)	8	2.86e-5	2 (1)	18	1.36e-3
PySR [Cranmer, 2023]															
PySR*	(0)	15	–	(0)	6	–	(2)	13	–	(0)	17	–	(2)	51	–
OpenEvolve [Sharma, 2025]															
Claude Opus 4.6	7 (2)	11	8.19e-7	8 (1)	2	9.37e-6	0 (0)	5	1.21e-4	9 (2)	9	1.07e-7	24 (5)	27	4.30e-6
LaSR [Grayeli et al., 2024]															
Llama-3.1-8B*	(0)	10	2.77e-4	(1)	4	2.73e-4	(2)	11	0.0018	(2)	16	7.44e-5	(5)	41	–
*GPT-4o-mini**	(1)	14	9.11e-5	(2)	5	1.53e-4	(4)	14	9.94e-4	(**7**)	18	9.23e-6	(14)	51	–
LLM-SR [Shojaee et al., 2024]															
*GPT-4o-mini**	(4)	19	4.12e-6	(4)	7	3.06e-6	(4)	16	7.62e-5	(5)	22	3.21e-9	(17)	64	–
Claude Opus 4.6[3]	9 (4)	14	4.13e-7	8 (5)	6	2.15e-7	0 (0)	3	7.11e-5	7 (2)	17	2.90e-9	24 (11)	40	9.80e-7
PiT-PO [Wang et al., 2026]															
Llama-3.1-8B (RL)**	(5)	28	4.13e-7	(7)	17	9.37e-8	(5)	18	6.57e-5	(3)	**23**	1.18e-8	(20)	86	–
FunctionEvolve (Ours)															
No LLM	7 (3)	32	1.76e-12	12 (8)	16	6.66e-13	7 (5)	11	1.89e-5	3 (0)	19	2.83e-14	29 (16)	78	1.47e-10
Llama-3.1-8B	20 (5)	32	7.52e-13	20 (9)	20	3.85e-13	17 (6)	18	4.59e-8	5 (3)	20	**1.20e-14**	62 (23)	90	1.17e-12
Qwen3.6-27B	27 (8)	**35**	1.99e-13	21 (11)	**23**	2.08e-13	29 (16)	30	5.19e-13	9 (3)	21	1.44e-14	86 (38)	109	2.06e-13
DeepSeek-V4-Pro	**32** (**21**)	34	2.18e-13	20 (14)	**23**	2.08e-13	33 (30)	**33**	5.15e-13	14 (6)	21	**1.20e-14**	99 (71)	111	1.78e-13
GPT-5.2 medium	30 (13)	34	2.08e-13	20 (15)	**23**	3.80e-13	37 (**36**)	**33**	4.03e-13	**16** (5)	21	1.32e-14	103 (69)	111	2.08e-13
Claude Opus 4.6	31 (16)	**35**	**1.85e-13**	**22** (**16**)	**23**	**2.06e-13**	**39** (34)	**33**	**2.44e-13**	15 (6)	22	1.44e-14	**107** (**72**)	**113**	**1.38e-13**
Reference: GT	36 (36)	35	1.49e-13	24 (24)	23	1.43e-13	44 (44)	37	1.14e-13	25 (25)	25	1.87e-14	129 (129)	120	4.98e-14

FunctionEvolve greatly improves symbolic recovery under both SA@50 and SA@1. The No LLM variant uses generic seed generation without semantic priors, training-NMSE-ranked selection, programmatic AST mutation, and the same structure-aware optimizer. It already recovers 29 SA@50 forms and 16 SA@1 forms, close to the strongest published PiT-PO top-1 result of 20. This shows that GP-style rule mutation combined with reliable structure-aware fitting is itself a strong symbolic-search baseline. Adding LLM-guided generation, selection, and mutation raises SA@50 to 107 with *Claude Opus 4.6*, while the strongest same-backbone rerun baselines, LLM-SR and OpenEvolve, each reach 24. At top-1, FunctionEvolve reaches 72 SA@1, exceeding PiT-PO by 3.6x.

Additional backbones show the same trend: *DeepSeek-V4-Pro* recovers 99 (71) tasks and is especially strong on chemistry, the mid-sized open-weight *Qwen3.6-27B* reaches 86 (38), and *Llama-3.1-8B* reaches 62 (23), improving substantially over the No LLM setting in SA@50 but only slightly in SA@1. This is notable because PiT-PO also builds on the Llama-3.1-8B backbone but further applies reinforcement learning, whereas our Llama-3.1-8B run uses no RL and still surpasses PiT-PO. Structured-output validation and fallback details are provided in Appendix C1.

Gains extend across all four scientific domains; relatively weaker materials-science recovery is partly explained by benchmark identifiability issues (Section 5). The gap between SA@50 and SA@1 shows that correct equations are often generated but not always ranked first by training NMSE. Our inspection suggests this ambiguity arises because some expressions are difficult to distinguish from high-order rational approximations or series expansions, especially over restricted data ranges.

[3]The public LLM-SRBench code reports only top-1 symbolic accuracy. We additionally audit the top-50 training-NMSE-ranked candidates for our *Claude Opus 4.6* rerun and report entries as SA@50 (SA@1). With this released implementation, our rerun is weaker at SA@1 than the originally reported *gpt-4o-mini* result, possibly because the authors used a different evaluation method; related reproducibility concerns are noted by other researchers at `https://openreview.net/forum?id=OL4RWQV8Qa`.

Main result: The main limitation of LLM-driven SR is exact symbolic recovery rather than numerical fit. FunctionEvolve closes much of this gap while preserving near-GT numerical precision.

Numerically, FunctionEvolve with *Claude Opus 4.6* reaches 113 tasks under $\mathrm{Acc}_{0.1}$, close to the GT reference value of 120, and maintains median test NMSE at 1.38×10^{-13}. Because Acc_{τ} divides by y_{true} pointwise, very small y_{true} values can amplify tiny numerical perturbations into threshold failures, so even GT equations may not satisfy the relative-error threshold; Appendix D8 discusses the alternative 95% Acc_{τ} metric and stricter numerical thresholds. These gains are not due to larger LLM budgets: FunctionEvolve averages 66.86 LLM calls per task, versus 1000.27 for LLM-SR and 201.50 for OpenEvolve, with lower price-weighted token cost than LLM-SR and comparable cost to OpenEvolve (Appendix B).

Numerical error decreases during search on both test and OOD splits; detailed trajectories, per-task comparisons, and ground-truth error distributions are reported in Appendix D10.

4.2 Evaluation beyond LLM-SRBench

AI-Feynman. On the full AI-Feynman family [Udrescu and Tegmark, 2020], comprising 100 original equations and 20 bonus equations, FunctionEvolve recovers all 120 at top-1. Table 3 retains the native denominators because prior studies differ in evaluation subsets and use of retrieval knowledge; these entries therefore do not form a same-protocol leaderboard.

Table 3: Reported exact-recovery results on AI-Feynman-style benchmarks. Protocols differ, so entries keep their original denominators.

Method	Evaluation set	Recovery
LaSR [Grayeli et al., 2024]	100 original	72/100
SR-LLM [Guo et al., 2025]	90 held-out	69/90
QDSR [Bruneton, 2025]	117-target subset	107/117
FunctionEvolve (Ours)	100+20 full set	SA@1 120/120

Figure 4 records when a correct expression first appears; round 0 denotes a match already present in the LLM-generated seeds before mutation. AI-Feynman matches are more concentrated at round 0 than those on synthetic LLM-SRBench, consistent with possible prior exposure to literature-derived equations.

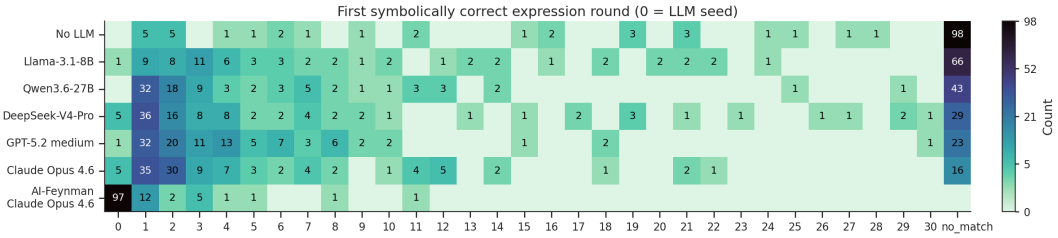


Figure 4: First-appearance round of the first symbolically correct expression for clean runs. Round 0 denotes a correct expression already present in the LLM-proposed seed set, while `no_match` denotes runs without a symbolic match. The last row shows AI-Feynman results, while the others show LLM-SRBench results. The greater concentration at round 0 is consistent with possible prior exposure to AI-Feynman equations during pretraining.

These results demonstrate broad coverage of established physics equations, but possible pretraining exposure limits their interpretation as evidence of evolutionary discovery. LLM-SRBench therefore remains our primary lower-contamination test bed.

LangLaw. We evaluate two LangLaw materials-property tasks based on first-principles calculations [Guan et al., 2026]: bulk modulus (22 candidate inputs) and band gap (27). With no known ground-truth expressions, we report held-out prediction error on fixed 80/20 train/test splits (403/101 and 596/149 samples); search and final-candidate selection use training data only.

Full outperforms both ablations and the published LangLaw formulas on these splits (Table 4), supporting LLM guidance and search-side AST structure on empirical materials data. Evidence is limited to two tasks and does not establish physical correctness. On band gap, XGBoost and an MLP achieve lower RMSEs of 0.396 and 0.417 eV than FunctionEvolve's 0.579 eV; its selected expression uses 10 of 27 inputs. Appendix D2 details the comparison protocol and limitations.

Table 4: LangLaw results on the same fixed splits. "Variables" denotes the number used in the selected expression over the number available. Published LangLaw formulas use their printed coefficients without refitting.

	Bulk modulus		Band gap	
Method	Variables	MAE (eV/Å^3)	Variables	RMSE (eV)
Published LangLaw formula	5/22	0.0837	5/27	0.673
No LLM	6/22	0.0695	7/27	0.756
w/o AST Structure	7/22	0.0620	10/27	0.654
Full	8/22	**0.0558**	10/27	**0.579**

4.3 Ablation Study

We ablate individual components as follows (details in Appendices C2–C6): *w/o Generator* removes domain knowledge and uses fixed seeds; *w/o Selector* uses Boltzmann sampling over fitness ranks; *w/o LLM Mutator* and *w/o AST Mutator* disable LLM proposals and AST-rule edits, respectively; and *w/o Structure-aware Optimizer* replaces structure-aware fitting with L-BFGS. *w/o All* combines removal of the Generator, Selector, AST Mutator, and structure-aware optimizer while retaining LLM mutation, resembling a parallel multi-offspring variant of LLM-SR [Shojaee et al., 2024]. Resource comparisons are provided in Appendix B.

Table 5: Ablation study on 129 tasks. Entries report total SA@50 (SA@1). Domain-wise SA results are reported in Appendix D7.

Method	*GPT-5.2-medium*	*Claude Opus 4.6*
Full	**103** (**69**)	**107** (**72**)
w/o All	35 (6)	34 (10)
w/o Generator	79 (49)	91 (58)
w/o Selector	62 (44)	74 (46)
w/o LLM Mutator	45 (27)	46 (31)
w/o AST Mutator	70 (42)	84 (50)
w/o Struct. Optimizer	46 (16)	53 (22)

Table 5 reports both backbones. With *Claude Opus 4.6*, removing the LLM Mutator or structure-aware optimizer reduces recovery from 107 (72) to 46 (31) or 53 (22), while the combined ablation reaches 34 (10). Removing the AST Mutator or Selector yields 84 (50) or 74 (46). These results support complementary roles for LLM-guided proposals, explicit local edits, parent selection, and reliable coefficient fitting. We next isolate the effects of search-side AST information and structure-aware optimization.

Ablation takeaway: FunctionEvolve succeeds by combining explicit AST structure, reliable structure-aware optimization, and frontier-model symbolic proposals.

4.4 Analysis of AST Structure

To isolate the role of search-side AST structure more directly, we use a stricter *w/o AST Structure* variant under *Claude Opus 4.6*. This variant keeps the LLM Selector and LLM Mutator, but removes AST-derived fields from the Selector, disables the AST Mutator, and removes the annotated AST and AST-specific mutation guidance from the LLM Mutator. The structure-aware coefficient optimizer remains unchanged, so this variant does not remove all structural mechanisms from the framework.

Table 6: Effect of AST structure under *Claude Opus 4.6*. Entries report SA@50 (SA@1).

Variant	Chemistry (36)	Biology (24)	Physics (44)	Materials (25)	Total (129)
Full	**31** (**16**)	22 (16)	**39** (**34**)	**15** (**6**)	**107** (**72**)
w/o AST Mutator	30 (**16**)	22 (11)	23 (20)	9 (3)	84 (50)
w/o AST Structure	17 (10)	**23** (**17**)	20 (13)	10 (3)	70 (43)

As shown in Table 6, removing search-side AST guidance and programmatic mutation reduces total SA@50 (SA@1) from 107 (72) to 70 (43). This is worse than only removing the AST Mutator, which still obtains 84 (50) when the LLM components can observe AST-derived structure. The gap indicates that AST structure is useful not only as a source of deterministic rule mutations, but also as an explicit representation that helps the LLM reason about expression complexity, reusable subtrees, and meaningful local edits. The degradation is especially severe on the Physics split, where recovery falls from 39 (34) to 20 (13), consistent with the greater structural complexity of its equations. Appendix D3 separately tests seed quality while retaining domain knowledge and all subsequent search components.

4.5 Effect of Task Context

We compare Full and *w/o AST Structure* on all 129 LLM-SRBench tasks using *Claude Opus 4.6* under three contexts. *Original* retains the task identity, variable names, and scientific description. *Blank* removes task identity, anonymizes variables as $y, x_1, x_2, \ldots$, and supplies no description. *Misleading* uses the same anonymization but supplies a description from an unrelated, same-arity task in another scientific domain. Numerical data, target expressions, search budgets, and operator vocabulary remain unchanged. No LLM does not consume task text, so its result is repeated as a context-independent reference.

Table 7: Effect of task context on all 129 LLM-SRBench tasks. Entries report SA@50 (SA@1).

Method	Original	Blank	Misleading
Full	**107 (72)**	**86 (50)**	**77 (35)**
w/o AST Structure	70 (43)	70 (41)	46 (26)
No LLM	29 (16)	29 (16)	29 (16)

Full outperforms *w/o AST Structure* in every context (Table 7). Correct task semantics increase Full's SA@50 from 86 to 107, whereas the corresponding count remains 70 without search-side AST structure. From Blank to Misleading, Full loses 9 recovered tasks, compared with 24 for *w/o AST Structure*. These comparisons support the role of explicit structure in using textual priors and reducing sensitivity to misleading descriptions. Original–Blank measures the joint effect of variable names, task identity, and description; Blank–Misleading isolates the effect of an incorrect description under the same anonymization. They do not fully separate pretrained knowledge from reasoning within the LLM.

4.6 Analysis of Structure-aware Coefficient Optimization

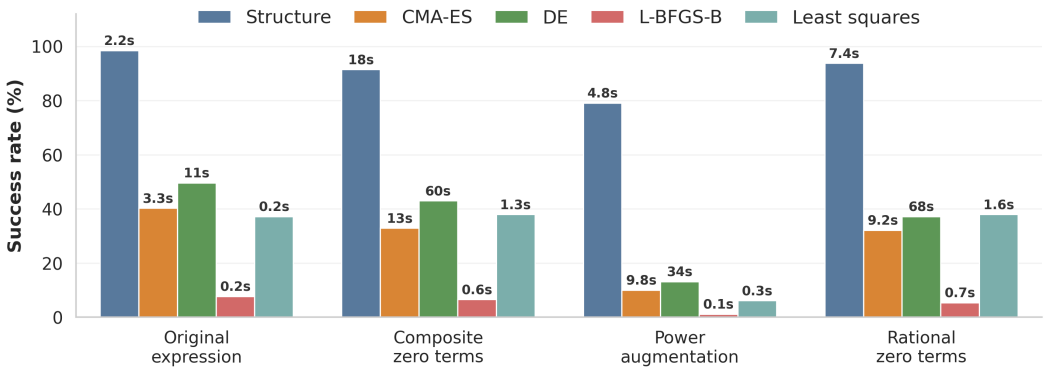


Figure 5: Optimizer benchmark on ground-truth skeletons and transformed variants. Success is defined as NMSE $<$ 1e-10; numbers above bars report median runtime.

To separate coefficient-fitting failure from symbolic-search failure, we benchmark optimizers on ground-truth skeletons and controlled variants that preserve the target function under known parameter assignments (Appendix C7). Success requires training NMSE $< 10^{-10}$. Figure 5 shows that the structure-aware optimizer outperforms standalone optimizers across all four groups with modest median runtime, solving 98.4% of original-skeleton cases. Because skeletons are supplied, failures directly reveal that valid symbolic structures can receive poor scores through inadequate coefficient fitting. Structural decomposition, parameter constraints, and simplification help prevent such false negatives during fitness evaluation.

4.7 Complexity-aware Final Candidate Selection

The gap between SA@50 and SA@1 indicates that the search often generates a correct expression but does not rank it first by training NMSE. We therefore examine complexity-aware final-candidate selectors for the *Claude Opus 4.6* full run, with the rules and schematic provided in Appendix D4. After search terminates, these selectors are applied to the complete expression trajectory to choose the final candidates to report. They are distinct from the evolutionary Selector in Section 3: they do not choose parents, guide mutation, affect coefficient fitting, or otherwise change the search trajectory.

The final-candidate selectors use only training NMSE and expression complexity, never test or OOD NMSE. Pareto performs non-dominated sorting over training NMSE and complexity and fills the reporting budget from successive Pareto layers. The Occam variant first restricts candidates to a near-best training-NMSE neighborhood and then favors simpler expressions using tree size, operator count, parameter count, and special-function count. The MDL variant uses a linear penalty on training loss and complexity features. As shown in Table 8, the five-candidate Pareto and Occam selections recover 102 and 101 exact forms, already above the 95 exact forms obtained by the top-10 training-NMSE ranking and close to the top-50 result of 107. The approximate $\mathrm{SA@all}$ value is 113, showing that a few additional exact forms appear deeper in the search trajectory. This suggests that many apparent top-1 failures are final-ranking failures caused by numerically strong but structurally less concise surrogate expressions. However, a gap remains to $\mathrm{SA@all}$, indicating that complexity-aware ranking still cannot always identify the correct form within a very small reporting budget.

Table 8: Final-candidate selection for the *Claude Opus 4.6* full run on 129 clean LLM-SRBench tasks.

Ranking rule	Shortlist	Exact matches
Train NMSE	1	72
Train NMSE	5	89
Train NMSE	10	95
Pareto	5	102
Occam	5	101
MDL	5	97
Pareto	10	102
Occam	10	104
MDL	10	101
Train NMSE	50	107
Train NMSE	all[4]	113

5 Discussion

To diagnose weaker MatSci results in Section 4, we identify two dataset identifiability issues. Across all 25 tasks and train/test/OOD splits, $T \approx 273 + 500\epsilon$, with a maximum absolute residual of 4.43×10^{-5}. This near-collinearity weakens identification of independent temperature effects, allowing structurally different expressions to achieve similar NMSE. In eight tasks, Arrhenius factors $\exp(-c/T)$ vary by only 0.11%–2.06% on train/test and 0.006%–0.108% on OOD, making $\epsilon^p \exp(-c/T)$ difficult to distinguish from a rescaled strain term $C\epsilon^p$. Appendix D12 provides the residual plot, detailed correlation audit, and a representative MatSci5 example.

In other dataset splits, the collinear issue is not dataset-wide as in MatSci, but individual tasks still exhibit strong pairwise dependence. Appendix D5 reports the 12 FunctionEvolve failure cases outside MatSci and shows that most still contain a variable pair with large R^2. These dependencies can again weaken structural identifiability, because different symbolic mechanisms may become difficult to distinguish over the sampled trajectory even when the overall split is not globally degenerate.

Although our audit highlights limitations in LLM-SRBench and the need for more reliable datasets, it remains the latest synthetic SR benchmark and reduces memorization risk relative to literature-derived benchmarks like AI-Feynman [Udrescu and Tegmark, 2020]; constructing a new benchmark is beyond the scope of this work.

6 Conclusion

We introduced FunctionEvolve, an LLM-driven SR framework combining semantic initialization, diversity-aware selection, local mutation by AST rule and LLM proposal, and structure-aware coefficient optimization. On the 129-task synthetic subset of LLM-SRBench, FunctionEvolve recovers 107 exact symbolic forms with *Claude Opus 4.6*, substantially improving symbolic accuracy while preserving near-GT numerical precision; on the complementary 120-task AI-Feynman evaluation, it recovers the correct symbolic form for all 120 tasks as its top-1 prediction. Our ablations explain why: LLM-extracted domain priors provide semantic guidance that steers the search toward plausible expressions, explicit AST structure organizes symbolic exploration, and structure-aware optimization prevents promising skeletons from being discarded due to poorly fitted constants.

Beyond these results, our audit identifies critical limitations in LLM-SRBench itself: the MatSci split has severe collinearity between T and ϵ, making key mechanisms unidentifiable and allowing structurally incorrect expressions to attain very low NMSE. Taken together, these findings suggest that progress in scientific SR requires joint attention to symbolic representation, coefficient fitting, and benchmark design. Open directions include dynamical systems without closed forms, larger candidate-variable sets, and benchmarks that are both reliable and realistic.

[4]Here, "all" denotes $\mathrm{SA@all}$ over roughly the top 1000 full-trajectory candidates ranked by training NMSE.

Acknowledgments and Disclosure of Funding

This work was supported in part by the NSFC Special Fund and Major Program under Grants 62550004 and 62595773; in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM101). We also thank Bosch (China) Investment Ltd. for covering all API costs incurred in this study, totaling approximately USD 58,000.

References

Anthropic. Introducing Claude Opus 4.6, February 2026. URL https://www.anthropic.com/news/claude-opus-4-6. Released February 5, 2026.

Jean-Philippe Bruneton. Enhancing symbolic regression with quality-diversity and physics-inspired constraints, 2025. URL https://arxiv.org/abs/2503.19043.

Bogdan Burlacu, Kaifeng Yang, and Michael Affenzeller. Population diversity and inheritance in genetic programming for symbolic regression. *Natural Computing*, 23, 01 2023. doi: 10.1007/s11047-022-09934-x.

William 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, 2021. URL https://arxiv.org/abs/2107.14351.

Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl, 2023. URL https://arxiv.org/abs/2305.01582.

Patrik D'haeseleer. Context preserving crossover in genetic programming. 1:256 – 261 vol.1, 07 1994. doi: 10.1109/ICEC.1994.350006.

L. G. A dos Reis, V. L. P. S. Caminha, and T. J. P. Penna. Benchmarking symbolic regression constant optimization schemes, 2024. URL https://arxiv.org/abs/2412.02126.

Mengge Du, Yuntian Chen, Zhongzheng Wang, Longfeng Nie, and Dongxiao Zhang. LLM4Ed: Large language models for automatic equation discovery, 2024. URL https://arxiv.org/abs/2405.07761.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library, 2024. URL https://arxiv.org/abs/2409.09359.

Yifeng Guan, Chuyi Liu, Dongzhan Zhou, Lei Bai, Wan-jian Yin, Jingyuan Li, and Mao Su. Discovery of interpretable physical laws in materials via language-model-guided symbolic regression, 2026. URL https://arxiv.org/abs/2602.22967.

Zelin Guo, Siqi Wang, Yonglin Tian, Jing Yang, Hui Yu, Xiaoxiang Na, Levente Kovacs, Li Li, Petros A. Ioannou, and Fei-Yue Wang. SR-LLM: An incremental symbolic regression framework driven by LLM-based retrieval-augmented generation. *Proceedings of the National Academy of Sciences*, 122(52):e2516995122, 2025. doi: 10.1073/pnas.2516995122.

Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21, 09 2020. doi: 10.1007/s10710-019-09371-3.

John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, June 1994. ISSN 1573-1375. doi: 10.1007/BF00175355. URL https://doi.org/10.1007/BF00175355.

Stefan Kramer, Mattia Cerrato, Jannis Brugger, Sašo Džeroski, and Ross King. Automated scientific discovery: From equation discovery to autonomous discovery systems, 2023. URL https://arxiv.org/abs/2305.02251.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

Pat Langley. Data-driven discovery of physical laws. *Cognitive Science*, 5(1):31–54, 1981. doi: https://doi.org/10.1111/j.1551-6708.1981.tb00869.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1551-6708.1981.tb00869.x.

Qiang Lu, Jun Ren, and Zhiguang Wang. Using genetic programming with prior formula knowledge to solve symbolic regression problem. *Computational Intelligence and Neuroscience*, 2016:1–17, 12 2015. doi: 10.1155/2016/1021378.

Georg Martius and Christoph H. Lampert. Extrapolation and learning equations, 2016. URL https://arxiv.org/abs/1610.02995.

David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, Summer 1995. ISSN 1063-6560. doi: 10.1162/evco.1995.3.2.199. URL http://vishnu.bbn.com/papers/stgp.pdf.

T. Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P. Santiago, Daniel M. Faissol, and Brenden K. Petersen. Symbolic regression via neural-guided genetic programming population seeding, 2021. URL https://arxiv.org/abs/2111.00053.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2021. URL https://arxiv.org/abs/1912.04871.

Subham S. Sahoo, Christoph H. Lampert, and Georg Martius. Learning equations for extrapolation and control, 2018. URL https://arxiv.org/abs/1806.07259.

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

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. LLM-SR: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models, 2025. URL https://arxiv.org/abs/2504.10415.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020. doi: 10.1126/sciadv.aay2631. URL https://www.science.org/doi/abs/10.1126/sciadv.aay2631.

Marco Virgolin and Solon P. Pissis. Symbolic regression is NP-hard, 2022. URL https://arxiv.org/abs/2207.01018.

Boxiao Wang, Kai Li, Tianyi Liu, Chen Li, Junzhe Wang, Yifan Zhang, and Jian Cheng. LLM-based scientific equation discovery via physics-informed token-regularized policy optimization, 2026. URL https://arxiv.org/abs/2602.10576.

Runxiang Wang, Boxiao Wang, Kai Li, Yifan Zhang, and Jian Cheng. DRSR: LLM-based scientific equation discovery with dual reasoning from data and experience, 2025. URL https://arxiv.org/abs/2506.04282.

P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 July 1995. URL http://divcom.otago.ac.nz/sirc/Peterw/Publications/ml95.zip.

Shijie Xia, Yuhan Sun, and Pengfei Liu. SR-Scientist: Scientific equation discovery with agentic AI, 2025. URL https://arxiv.org/abs/2510.11661.

A Code and Data Availability

We provide a public repository containing the FunctionEvolve implementation, experiment configuration files, prompt templates, scripts for running the main experiments and baselines, and raw result files used to produce the tables and figures in this paper. The repository also includes environment setup instructions, benchmark data access and preparation instructions, and command-line examples for reproducing the main LLM-SRBench experiments and ablations: https://github.com/Phoinikas03/FunctionEvolve.

The existing benchmark and baseline assets used in our experiments are credited in the main text and are publicly available from their official repositories. LLM-SRBench and its associated LLM-SR benchmark code are available under the MIT license at https://github.com/deep-symbolic-mathematics/llm-srbench; OpenEvolve is available under the Apache-2.0 license at https://github.com/algorithmicsuperintelligence/openevolve. The AI-Feynman benchmark is available from the official download page: https://space.mit.edu/home/tegmark/aifeynman.html.

The main LLM-SRBench evaluation uses the 129-task synthetic subset, covering four scientific domains: biology/population growth, chemistry/reaction kinetics, physics/oscillation, and materials science. For the AI-Feynman evaluation in Section 4.2, we use the full benchmark family consisting of the 100 original Feynman equations and the 20 bonus equations, for a total of 120 tasks. Symbolic matches are judged against the benchmark target expressions over the benchmark-specified variable domains. Thus, domain-restricted identities are counted as matches when the candidate and target agree throughout the stated domain; equivalence outside that domain is not required.

B Resource Usage

The main LLM-SRBench comparisons, ablations, and rerun baselines use the same 129-task synthetic subset. For the baseline reruns, we follow the public LLM-SR and OpenEvolve implementations and keep their released search and evaluator hyperparameters unchanged, rather than tuning them for FunctionEvolve. Concretely, FunctionEvolve runs for at most 30 evolution iterations per task, OpenEvolve runs for 200 iterations per task, and LLM-SR runs with a 1000-sample budget per task.

B1 LLM Usage

LLM call accounting. Both the *Claude Opus 4.6* and *GPT-5.2-medium* experiments are run through external LLM APIs. Unless otherwise stated, the accounting below uses the complete full-setting *Claude Opus 4.6* logs over the 129 LLM-SRBench tasks.

In each FunctionEvolve evolution iteration, the Selector chooses 5 candidate expressions as parents, and the LLM Mutator proposes 20 new candidate expressions for each selected parent. Search terminates early once 50 mature expressions are found, where maturity is defined by training NMSE falling below the task-specific threshold. Retries are used only when parsing or validation fails.

For the baselines, LLM-SR generates one new candidate expression per sample, and OpenEvolve generates one new child program per iteration. Table B1 reports the nominal per-task LLM-call budgets. Actual counts may be lower because of early stopping or higher because of retries and other implementation-specific calls. Figure 6 and Table B2 summarize the observed call counts.

The observed call counts show that FunctionEvolve uses substantially fewer LLM requests than the baseline search loops. Although its nominal per-task budget is 182 calls, early stopping reduces the average to 66.86 calls per task. Most of these calls are spent on the LLM Mutator, which averages 53.80 calls per task, while the Selector averages 11.06 calls and the Generator is fixed at 2 calls. In contrast, LLM-SR stays close to its 1000-iteration budget, and OpenEvolve stays close to its 200-iteration budget. Thus, FunctionEvolve reaches its main results with about 15x fewer LLM calls than LLM-SR and about 3x fewer calls than OpenEvolve on average.

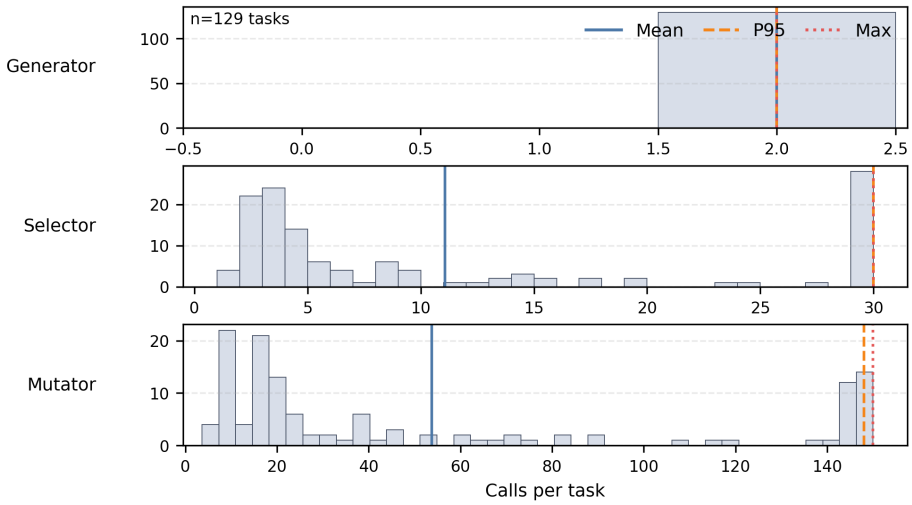


Figure 6: Component-wise distributions of LLM call counts over 129 complete task logs from the full *Claude Opus 4.6* FunctionEvolve experiment. Vertical lines mark the mean, 95th percentile, and maximum per-task call count for each main search component.

Table B1: Nominal per-task LLM-call budgets for FunctionEvolve and the LLM-driven baselines.

Method/component	Trigger	Nominal call budget per task
FunctionEvolve		
Generator	Domain-prior extraction and seed initialization	2 calls
Selector	Once per evolution round	≤ 30 calls
LLM Mutator	Once per selected parent	$\leq 30 \times 5 = 150$ calls
LLM-SR	One child hypothesis per iteration	1000 calls
OpenEvolve	One child hypothesis per iteration	200 calls

Table B2: LLM call usage on the 129-task LLM-SRBench subset under *Claude Opus 4.6*.

Method/component	Total calls	Mean/task	Std./task	Median	P95	Max
FunctionEvolve	8625	66.86	64.88	30	180	182
Generator	258	2.00	0.00	2	2	2
Selector	1427	11.06	11.08	5	30	30
LLM Mutator	6940	53.80	53.80	23	148	150
LLM-SR	129035	1000.27	3.07	1001	1001	1001
OpenEvolve	25994	201.50	2.11	200	205	206

LLM token accounting. We follow the same accounting protocol as the LLM call accounting. Table B3 reports the aggregate prompt, completion, and total token counts, and Figure 7 shows the per-task token distribution for the three FunctionEvolve LLM components, separating input and output tokens.

The token accounting shows a different tradeoff from the call accounting. FunctionEvolve uses fewer LLM calls than both LLM-SR and OpenEvolve, but each call is longer because the Selector observes the evolving tree summary and the LLM Mutator receives rich structural context. As a result, FunctionEvolve uses 340.36M total tokens, compared with 262.16M for LLM-SR and 244.79M for OpenEvolve. Within FunctionEvolve, token usage is highly concentrated in the Selector: it accounts for 290.44M tokens, or 85.33% of the FunctionEvolve total, while the LLM Mutator accounts for 49.07M tokens and the Generator for only 0.85M. Figure 7 shows that per-task token usage varies mainly in the Selector and LLM Mutator, while the Generator remains nearly constant across tasks because it is used only for initialization. Since prompt tokens are much cheaper than completion tokens in typical LLM pricing, we also report a price-weighted cost proxy in Table B3, computed

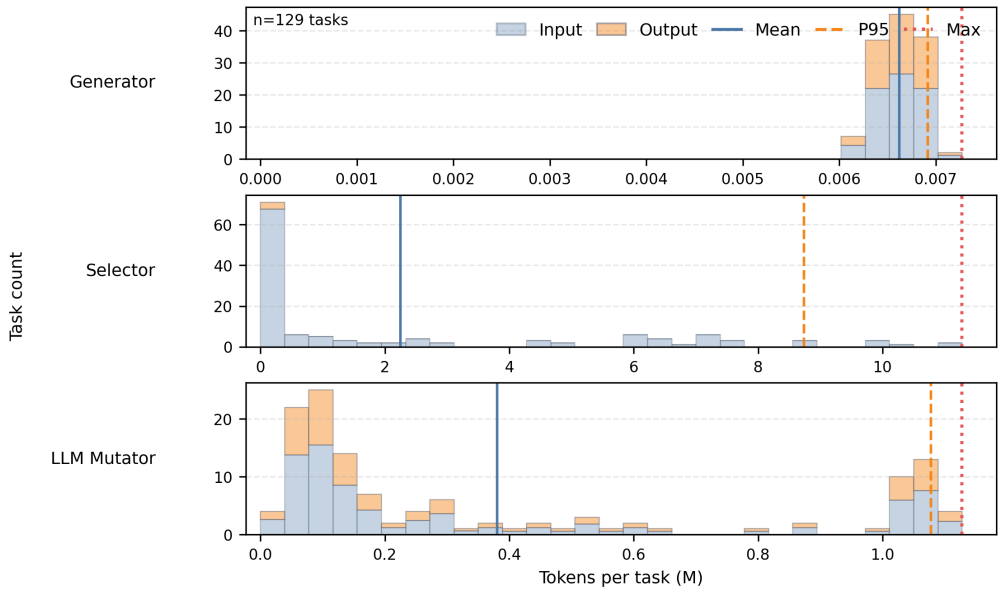


Figure 7: Per-task token-usage distributions for the three main FunctionEvolve LLM components under *Claude Opus 4.6*. Each row is a histogram over the 129 tasks; each bin is split into input-token and output-token contributions. Vertical lines mark the mean, 95th percentile, and maximum total tokens per task for the corresponding component.

as prompt tokens$/5$ plus completion tokens. Under this weighting, FunctionEvolve costs 85.56M output-token equivalents, lower than LLM-SR's 143.45M and comparable to OpenEvolve's 68.13M despite using more total tokens.

Table B3: Observed token usage on the 129-task LLM-SRBench subset under *Claude Opus 4.6*. Token counts and price-weighted costs are reported in millions. The weighted-cost column assumes one prompt token costs one fifth of one completion token.

Method/component	Prompt tokens	Completion tokens	Total tokens	Weighted cost
FunctionEvolve	318.50	21.86	340.36	85.56
Generator	0.50	0.35	0.85	0.45
Selector	288.86	1.58	290.44	59.35
LLM Mutator	29.14	19.93	49.07	25.76
LLM-SR	148.40	113.77	262.16	143.45
OpenEvolve	220.83	23.96	244.79	68.13

B2 CPU Usage

All local computation was performed on modern multicore CPU servers with heterogeneous configurations, each providing at least 64 logical CPUs. The local CPU workload consists primarily of candidate validation, program or expression execution, coefficient optimization, and post-fit screening. The existing logs do not record process-level user/system CPU time or instantaneous utilization; therefore, we report observed wall-clock time, evaluator parallelism, candidate-evaluation counts, and timeout behavior rather than exact CPU-hours.

This hardware lower bound is important for interpreting wall-clock cost. The original LLM-SR loop generates one offspring from one parent at each iteration, and OpenEvolve similarly evaluates one child program per iteration. FunctionEvolve instead selects multiple parents and expands them into a batch of candidate expressions. Although the full setting evaluates 126.09 offspring per round on average, these evaluations are independent and can be distributed across modern multicore CPUs; consequently, wall-clock time does not increase in proportion to the number of candidates, while

Table B4: CPU-evaluation protocol for FunctionEvolve and the LLM-driven baselines. Retry counts refer to local evaluator retries, not LLM API retries.

Method	Offspring per round	Evaluator	Timeout	Retries
FunctionEvolve	100 LLM proposals + rule mutations (mean: 126.09)	Structure-aware coefficient optimizer	120 s	0
LLM-SR	1 candidate per sample	BFGS optimizer	30 s	0
OpenEvolve	1 child program per iteration	BFGS optimizer	90 s	3*

*OpenEvolve retries evaluation exceptions up to three times; evaluator timeouts are returned directly as timeout results.

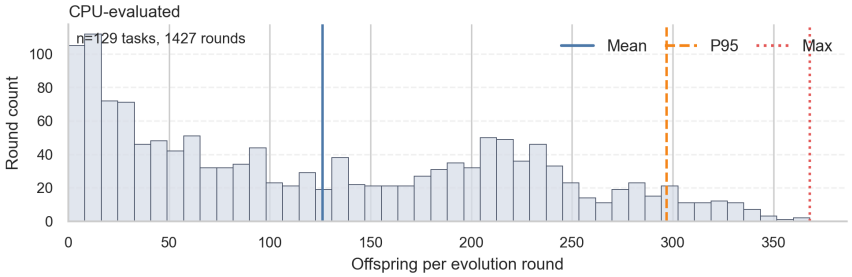


Figure 8: Distribution of CPU-evaluated offspring counts per FunctionEvolve round over 129 complete full-run task logs. Counts include candidates that remain after structural deduplication and parameter-count filtering and therefore enter coefficient optimization. Vertical lines mark the mean, 95th percentile, and maximum.

one-offspring baselines leave much of this parallelism unused. The same batch-parallel evaluation applies to the multi-offspring *w/o All* ablation.

Table B4 summarizes the local CPU-evaluation protocol. In FunctionEvolve, the 100 LLM-proposed offspring in each round come from 5 LLM Mutator calls over 5 selected parents, and deterministic rule mutations add further non-LLM offspring. The resulting batch-size distribution is shown in Figure 8. Using the most conservative 64-logical-CPU reference, the mean batch size corresponds to roughly $126.09/64 = 1.97$ candidate-equivalent parallel waves, making the local wall-clock evaluation load comparable to the one-offspring baseline steps up to scheduling and optimizer-runtime variation.

B3 GPU Usage

GPU resources were used only for local LLM inference, not for symbolic candidate execution, coefficient optimization, or post-fit screening. The locally served *Llama-3.1-8B*, *Qwen3.6-27B*, and *DeepSeek-V4-Pro* inference endpoints were deployed on 16 NVIDIA H20-3e data-center AI accelerators with CUDA 13.1 and 144 GiB HBM per card.

These GPU resources should therefore be interpreted as serving hardware for the local LLM backbones, which were served with vLLM 0.20.0 for inference. The CPU accounting in Appendix B2 remains the relevant local compute description for expression evaluation and fitting, while externally hosted LLMs such as *Claude Opus 4.6* are represented only through the call and token accounting in Appendix B1.

C Additional Details of FunctionEvolve

C1 Algorithmic Procedure

Operational subroutines RULEADDITION is a deterministic parent-level expansion. For a selected parent p, it attaches the elementary content library in Table 1 through $p + g$, $p \cdot g$, division templates, and unary wraps. There is no blanket preprocessing stage applied to all seeds or all offspring; each generated candidate corresponds to one explicit mutation step.

FunctionEvolve

Input LLM modules π_G, π_S, π_M, dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with task-level context T
Output Ranked symbolic expressions with fitted constants and scores

 # Initialization by Generator
$\mathcal{K} \leftarrow \pi_G.\textsc{Knowledge}(T, \mathcal{D})$
$\mathcal{S}_0 \leftarrow \pi_G.\textsc{Seeds}(T, \mathcal{D}, \mathcal{K})$
$\mathcal{E} \leftarrow \emptyset,\ \mathcal{H} \leftarrow \emptyset$
for $f \in \mathcal{S}_0$ **do**
  $f, \Theta \leftarrow \textsc{Normalize}(f)$
  $s, \hat{\mathbf{c}} \leftarrow \textsc{Optimizer}(f, \Theta, \mathcal{D})$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{(f, \Theta, \hat{\mathbf{c}}, s, \text{root})\}$
end for
 # Evolution over the expression tree
while stopping criterion is not met **do**
  $\mathcal{Z} \leftarrow \textsc{TreeSummary}(\mathcal{E}, \mathcal{H})$
  $\mathcal{P} \leftarrow \pi_S.\textsc{SelectParents}(\mathcal{Z}, T)$
  $\mathcal{C} \leftarrow \emptyset$
  for $p \in \mathcal{P}$ **do**
    $\mathcal{A} \leftarrow \textsc{RuleAddition}(p) \cup \textsc{RuleDeletion}(p)$
    $\mathcal{L} \leftarrow \pi_M.\textsc{Mutate}(p, \text{AST}(p), \mathcal{A}, \mathcal{K}, \mathcal{Z})$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(g, p) : g \in \textsc{NormalizeDedup}(\mathcal{A} \cup \mathcal{L})\}$
  end for
  for $(g, p) \in \mathcal{C}$ **do**
    $g, \Theta_g \leftarrow \textsc{Normalize}(g)$
    if $\textsc{Valid}(g, \Theta_g, \mathcal{E})$ **then**
      $s_g, \hat{\mathbf{c}}_g \leftarrow \textsc{Optimizer}(g, \Theta_g, \mathcal{D})$
      $\delta_g, \tilde{g}, \tilde{\Theta}_g \leftarrow \textsc{DegeneracyCheck}(g, \Theta_g, \hat{\mathbf{c}}_g)$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(g, \Theta_g, \hat{\mathbf{c}}_g, s_g, \text{parent} = p, \delta_g)\}$
      if $\delta_g = \textsc{simplified}$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\tilde{g}, g)\}$ **end if**
    end if
  end for
  $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{P}$; $\textsc{MarkMature}(\mathcal{E})$
end while
return $\textsc{TopExpressions}(\mathcal{E})$

DEGENERACYCHECK is a deterministic state-management routine that decides whether an evaluated candidate remains active. It returns $\delta_g \in \{\text{OK}, \text{SIMPLIFIED}, \text{OVERFIT}, \text{TIMEOUT}\}$. It marks deep expressions with abnormal fitted coefficients ($|c_j| > 10^3$) as OVERFIT, applies constant-induced simplifications such as zeroing small multiplicative terms, snapping near-rational exponents, simplifying $\exp(0)$ and $\log(1)$, reducing simple trigonometric degeneracies, and merging nearly identical nonlinear terms. If the simplification removes all feature variables or creates infinity, the candidate is marked OVERFIT; if it yields a different valid expression, the original evaluated candidate is marked SIMPLIFIED and the simplified expression is reinserted as its descendant for normal normalization, deduplication, and optimization. Timeouts are conservatively treated as OK.

These flags directly control the search tree. Nodes with $\delta_g \neq$ OK are kept only as evaluated audit records and are excluded from the Selector's tree summary, so they cannot become future parents. MARKMATURE marks an active evaluated node as mature once its training NMSE is below τ_{train}. Mature nodes may still be selected for a small annealing budget; after that budget is exhausted, they are removed from the Selector context. The run stops early after the configured number of mature nodes has been collected. TOPEXPRESSIONS returns mature nodes, supplemented if needed by non-mature evaluated nodes, ordered by training NMSE. Test and OOD NMSE are logged for post-hoc reporting only and are not exposed to the Selector or Mutator during search.

Full-run configuration Table C1 lists the command-line configuration used by `run.sh full`; unspecified arguments use the defaults in `main.py`.

Structured-output validation and fallback. In practice, Llama-3.1-8B also has weaker JSON-format following than newer models, so invalid structured outputs are replaced by the deterministic rule-based baseline in our implementation. When the LLM interface fails, the system falls back to the

Table C1: `run.sh full` configuration for the full FunctionEvolve setting.

Parameter	Meaning	Value
`-optimizer`	Constant-fitting backend	`Structure`
`-max-steps`	Maximum evolutionary iterations per task	30
`-n-seeds`	Number of Generator seed expressions	20
`-candidate-num`	Number of parents selected per iteration	5
`-selector-context-size`	Maximum number of active nodes shown to the Selector	200
`-max-mature-nodes`	Early-stop target number of training-NMSE mature nodes	50
`-overfit-min-depth`	Minimum SymPy AST depth for large-coefficient overfit screening	10
`-equation-workers`	Active equation-level workers in full mode	10
`-n-eval-workers`	Local child-evaluation workers when no shared global worker pool is used	16 (default)
`-timeout`	Per-task wall-clock timeout shared by normalization, evaluation, degeneracy check, and optimizer	120 s (default)
`-refine-output`	Final parent-selection sweep for mature/supplement nodes	disabled in `full`; enabled only by `refine_full`
`-data-dir`	LLM-SRBench data directory	optional; defaults to dataset loader path unless provided

baseline. Therefore, the measured gains provide a conservative estimate of the benefit of LLM-guided search.

C2 Generator

Procedure The Generator is used during the one-shot initialization stage. It has two LLM-facing subroutines. First, `generate_domain_knowledge` reads the task context and available variables, infers either the scientific domain or, for dimension-only tasks such as AI-Feynman entries without natural-language background, a dimension-aware structural prior. It returns formula priors and heuristic structural cues. These LLM-proposed priors are combined with a fixed catalog of generic scientific laws or dimension/variable-name templates, so the downstream search receives both broad priors and task-specific or metadata-specific priors.

Second, `initialize_seeds` uses the task context, variables, and extracted knowledge to generate the initial expression pool. The seed prompt asks for diverse structures spanning both compact monomial/product/ratio/composition forms and multi-term sums, with roughly half generic elementary-function structures and half knowledge-informed structures. Each response must be a JSON array of SymPy-parseable expressions together with the parameter placeholders used in the expression. The implementation validates the JSON schema, parses each expression with SymPy, checks that free symbols come only from dataset variables or listed `c0`, `c1`, ... constants, and requires the listed constants to match those actually used. If LLM generation yields too few valid seeds, a fixed fallback seed list supplements the pool.

Table C2: Generator configuration used by the *Claude Opus 4.6* full run. Component-specific overrides are read from `opus-4-6.yaml`.

Parameter	Meaning	Value
`n_seeds`	Number of initial seed expressions	20
`generator.temperature`	Default Generator decoding temperature	1.0
`domain_knowledge.temperature`	Domain-knowledge extraction temperature	0.7
`seed_generation.temperature`	Seed-expression generation temperature	0.9
`describe_batch.temperature`	Batch-description temperature	0.5
`max_tokens`	Maximum response length for Generator subroutines	128000

Generator Ablation For ablations, FunctionEvolve can replace the LLM Generator with a deterministic `MockGenerator`. In this mode, no LLM calls are made: domain knowledge is empty and initial expressions are drawn from a fixed fallback list. This isolates the contribution of LLM-based initialization and domain-prior extraction from the rest of the search pipeline.

Prompting interface The Generator uses separate prompt interfaces for domain knowledge extraction and seed expression initialization. The prompt boxes below show the interfaces used by these subroutines in abbreviated form. The domain knowledge prompt returns a JSON object, while the seed prompt returns a JSON array. In both cases, the implementation requests JSON-only output, optionally attaches a guided JSON schema when the backend supports it, extracts a JSON block if one is present, and validates the parsed result before it is inserted into the search state. The same seed-generation interface is used across LLM-SRBench and AI-Feynman; for AI-Feynman tasks without natural-language background, only the domain-knowledge stage switches to a dimension/variable/range-aware variant, and the seed prompt labels the knowledge-informed half as dimension/variable-informed formulas.

Generator domain-knowledge system prompt

```
You are a cross-disciplinary mathematical modeling expert, proficient in
↪  classical formulas and mathematical models from physics, chemistry,
↪  biology, engineering, economics, and other fields.

Given a background description of a symbolic regression task (including
↪  variable names and their physical meanings), please:
1. Analyze the most likely scientific domain/sub-domain for this task
2. List classical formula structures related to these variables in that
↪  domain

## Output Format
Return only a JSON object:
{"domain": "<domain name>", "analysis": "<1-2 sentence domain analysis>",
↪  "formulas": [...], "heuristics": [...]}

- domain: The scientific domain this task belongs to.
- analysis: Briefly explain why it belongs to this domain and the possible
↪  physical relationships between variables.
- formulas: List 5~15 common formula structural patterns in this domain. Use
↪  actual variable names when possible, arrange from simple to complex,
↪  include domain-specific function forms, and write each entry as "Name:
↪  formula form (applicable scenario)".
- heuristics: List 5~10 domain-specific formula-structure heuristic features
↪  describing what physical behavior corresponds to what mathematical
↪  structure.

Constant notation convention: constant parameters must appear in
↪  multiplicative form and must not be placed in the denominator. Use `c *
↪  var` rather than `var / c`.
```

Generator domain-knowledge user prompt

```
{context}

Available variables: {variables}

Please analyze the scientific domain of this task and list common formula
↪  structures related to these variables in that domain.
```

Generator seed system prompt

```
You are a symbolic-regression expert. Generate diverse candidate closed-form formulas for the dataset described below, to seed an evolutionary search.

## Structural Prior
The ground-truth formula may be EITHER:
- a single compact monomial / product / ratio / composition, such as `c0*x1*x2/x3` or `c0*exp(-c1*x**2)`; OR
- a sum of several terms, such as `c0*x + c1*x**2 + c2`.
Your seed set MUST cover both archetypes.

## Building Blocks
power `x**c`, trigonometric `sin`/`cos`, exponential `exp`, logarithmic `log(1+x)`, `sqrt`, and, where physically meaningful, `Abs(x)` and `Max(x, 0)`.

## Using {knowledge_basis}
{domain_knowledge}

## Constant Notation Convention
Constant parameters should always appear in multiplicative form, never alone in a denominator: write `c0*x` and `c0/(x + c1)`, not `x/c0`.

## Task Background
{context}

## Available Variables
{variables}

Please generate {n_seeds} candidate formulas with different structures at once, and return only a JSON array:
[{"expression":"...","params":["c0","c1"]}, ...]
```

Generator seed user prompt

```
Based on the above task metadata, generate {n_seeds} candidate formulas with diverse structures.

## Requirements
Formulas must cover both of the following categories, arranged from simple to complex:

A. Basic function structures (approximately half): Include BOTH compact single-term shapes (monomial/product/ratio/composition) AND multi-term sums. Ensure at least one variant of each basic type:
- Monomial / product: c0*x1*x2, c0*x1*x2/x3
- Power-law / composition: c0*x**c1, c0*exp(-c1*x**2), c0/sqrt(1 - x**2/c1**2)
- Linear: c0*x + c1
- Quadratic/polynomial: c0*x**2 + c1*x + c2
- Non-integer power law: c0*x**c1
- Exponential: c0*exp(c1*x)
- Logarithmic: c0*log(1 + c1*x)
- Rational: c0*x/(x + c1)
- Trigonometric: c0*sin(c1*x)
- Hyperbolic: c0*tanh(c1*x + c2)
- Square root: c0*sqrt(x**2 + c1)
- Simple combinations of the above basic types

B. Knowledge-informed formulas (approximately half): Based on the task background and the domain-knowledge or dimension/variable-name structures listed above, generate formula forms most likely to match the data.
```

Table C3: Main information maintained for each node in the evolutionary tree. The last column indicates whether the field itself, or a derived summary of it, is visible to the Selector.

Field	Type	Description	Selector visibility
`id`	string	Identifier for this node.	Yes
`formula`	string	Canonical symbolic expression string.	Yes
`parent_id`	string / None	Identifier of the parent node in the evolutionary tree.	Yes
`train_nmse`	float / None	Training NMSE after constant fitting.	Yes
`test_nmse`	float / None	Test NMSE maintained for evaluation.	No
`ood_test_nmse`	float / None	Out-of-distribution NMSE maintained for evaluation.	No
`is_evaluated`	bool	Whether the node has been evaluated after constant fitting.	No
`tree_depth`	integer	Depth of the parsed expression tree; exposed as `depth`.	Yes
`operator_counts`	dictionary	Counts of operator types in the expression tree; exposed to the Selector only through the aggregated field `n_operators`.	Yes
`ast_features`	structured features	Additional AST-derived structural features used internally.	No
`param_names`	list of strings	Names of tunable scalar parameters; exposed only through the aggregated field `n_params`.	Yes
`fitted_params`	list of floats	Optimized parameter values; exposed to the Selector as a formatted string when available.	Yes
`sympy_expr`	symbolic object	Parsed symbolic expression used for internal manipulation and analysis.	No

```
Note: All constants in formulas must be represented as c0, c1, c2..., and
↪  variables must use names from the available variables list.
```

C3 Evolutionary Tree Search

FunctionEvolve maintains the search state as an evolutionary tree rather than as an unstructured pool of candidate expressions. Each node in the tree corresponds to a symbolic expression together with its associated search state, while edges record parent-child relations induced by mutation. This representation preserves structural lineage throughout the search and allows the system to summarize the current search frontier for downstream selection.

At a high level, the tree stores two types of information: the current set of candidate expressions and the parent-child relations among them. Each node further maintains its own evaluation statistics, structural information, and parameter-related information. Table C3 summarizes the main information maintained for each node, together with whether it is visible to the Selector.

C4 Selector

Procedure At each search iteration, the Selector receives a compact summary of the current search tree, together with the task context and the record of previously chosen parents. Based on this information, it identifies a small set of parent expressions for further expansion and provides a brief rationale for each choice. These selected parents are then forwarded to the subsequent mutation stage. In this way, the Selector determines how the available search budget is allocated across competing search directions.

Table C4: Selector configuration used in our experiments.

Parameter	Meaning	Value
`candidate_num`	Number of parents selected per iteration	5
`selector_context_size`	Maximum number of summarized active nodes visible to the Selector in `run.sh full`	200
`temperature`	Sampling temperature for *Claude Opus 4.6* Selector decoding	0.3
`max_tokens`	Maximum response length for Selector decoding	128000

Visible node information Although each tree node stores a full internal state, the Selector does not access this state directly. Instead, it operates on the subset of per-node information marked as visible in Table C3, together with the aggregated statistic `n_children`. In practice, this summary provides information about predictive quality, structural complexity, local parameterization, and search history, without exposing the full internal node state. Importantly, the summary contains only training error. Test error and out-of-distribution error are maintained internally for evaluation, but are never revealed to the Selector. This design prevents evaluation-time information from leaking into the parent-selection process.

Selector Ablations For ablations, FunctionEvolve supports two Selector ablation variants. The first variant replaces the LLM Selector with a deterministic `MockSelector`. This fallback ranks evaluated nodes by numerical fitness and samples parents using a rank-based Boltzmann distribution, so no LLM call is made during parent selection. The second variant keeps the LLM-based parent-selection interface but removes AST-derived structural fields from the prompt, including parameter count, tree depth, and operator count. This preserves LLM decision making while hiding explicit AST structure, isolating the contribution of structural visibility in the Selector.

Prompting interface The exact prompt templates used by the Selector are shown below. The system prompt defines the selection criteria and the required JSON-only output format, while the user prompt template supplies the current tree summary and the historical selection records. The implementation also attaches the same guided JSON schema when the backend supports it. The Selector uses the same main template across datasets; AI-Feynman tasks append the dimension/variable/range task background and a short instruction to prioritize formula quality, structural diversity, variable-name cues, and dimensional metadata. For the ablation without AST-derived metadata, we use a variant with the corresponding fields removed.

Selector system prompt

```
You are a strategic planning expert for symbolic regression search.

You will see a summary table of all generated formulas in the current
↪  evolution tree, each record contains:
- id          : Numeric unique identifier of the formula
- formula     : Formula in SymPy format
- train_nmse  : Training set NMSE = MSE/Var(Y) (None means not yet evaluated;
↪  lower is better, <1 is better than mean prediction)
- n_children  : Number of offspring this node has produced
- n_params    : Number of tunable parameters
- depth       : AST tree depth (nesting level; higher means deeper function
↪  nesting)
- n_operators : Total number of operator nodes in AST tree (higher means
↪  longer expression)
- fitted_params : Fitted parameter values (scientific notation)

Your task is to select **{candidate_num} parent nodes** for the next
↪  evolution iteration.
The system will automatically perform AST-based structural mutations (delete
↪  subtrees, add new terms, etc.) on each parent to generate multiple
↪  candidate offspring.

Please consider the following when selecting:
1. **Structural simplicity first**: Do not simply select the nodes with the
↪  lowest NMSE. Prefer structurally simple nodes (fewer n_params, shallower
↪  depth, fewer n_operators) that already have relatively low NMSE (e.g.,
↪  ~1e-3 order) - simple good formulas are more likely to produce meaningful
↪  improvements through mutation, while low NMSE in complex formulas often
↪  comes from overfitting
```

```
2. **Algebraic structure diversity**: The selected {candidate_num} parents
↪  should cover different algebraic structures as much as possible (e.g.,
↪  polynomial, power law, exponential, trigonometric, rational, etc.),
↪  avoiding highly similar formulas. Diverse starting points enable more
↪  comprehensive exploration of the search space
3. **NMSE performance**: When structures are similar, prefer nodes with lower
↪  NMSE that still have room for improvement
4. **Number of offspring (n_children)**: If a node already has many offspring
↪  (e.g., n_children >= 5), it means that direction has been sufficiently
↪  explored; prefer nodes with fewer offspring
5. **Historical selection records**: Refer to the historical parent selection
↪  records. If certain formula structures have been repeatedly selected as
↪  parents in recent rounds but NMSE has not significantly decreased, they
↪  should be temporarily shelved in favor of other directions; if certain
↪  structures have not been explored recently, encourage selecting them as
↪  parents to expand the search space

Return only a JSON array with exactly {candidate_num} objects. Use the
↪  numeric `id` shown in the node summary as `parent_id`; do not copy the
↪  formula string into `parent_id`.

Required JSON schema:
{
  "type": "array",
  "minItems": {candidate_num},
  "maxItems": {candidate_num},
  "items": {
    "type": "object",
    "required": ["parent_id", "rationale"],
    "additionalProperties": false,
    "properties": {
      "parent_id": {"type": "integer"},
      "rationale": {"type": "string"}
    }
  }
}

Example item format (shape only; your actual output must contain exactly
↪  {candidate_num} objects):
[
  {"parent_id": 12, "rationale": "<selection rationale 1>"},
  {"parent_id": 7, "rationale": "<selection rationale 2>"}
]

- rationale: short final reason for selecting this parent.
```

Selector user prompt

```
Current evolution tree node summary ({n_nodes} nodes total, sorted by
↪  train_nmse ascending):

{tree_summary}

## Historical Parent Selection Records
{selection_history}

Please comprehensively analyze the above nodes and historical selection
↪  records, select {candidate_num} structurally diverse and concise parents,
↪  and output a JSON array.
```

C5 Mutator

AST representation Each symbolic expression is parsed into a SymPy abstract syntax tree (AST), where internal nodes correspond to operators such as `Add`, `Mul`, `Pow`, `exp`, `log`, and `sin`, and leaf nodes correspond to variables or scalar parameters. For each expression f, we extract depth-aware structural features of the form

$$\mathcal{F}(f) = \{\texttt{Op}(\text{Depth} : k) \mid \texttt{Op} \text{ is the operator type of a node and } k \text{ is its depth}\}.$$

For example, the expression $c_0 x^{c_1} + c_2 \exp(c_3 x)$ contains features such as `Add(Depth:0)`, `Mul(Depth:1)`, `Pow(Depth:2)`, and `exp(Depth:2)`.

Rule-based mutations The AST-rule Mutator applies deterministic edits directly on the expression tree. We use two classes of rule-based mutations. The first class performs deletion and simplification operations, including removing one term from an `Add` node, removing one non-numeric factor from a `Mul` node, unwrapping elementary functions such as $\exp(g) \mapsto g$, $\log(g) \mapsto g$, and $\sin(g) \mapsto g$, and reducing powers such as $x^n \mapsto x$. These operations are recursively applied to valid subtrees and remove candidates that do not contain feature variables.

The second class performs broad template-based addition. Given a parent expression f, the Mutator first defines an elementary content library g using the families in Table 1. It then enumerates $f + g$ additive terms, $f \cdot g$ multiplicative factors, division templates, and unary wraps $\phi(cf)$ or f^c. In implementation, division uses the numerically stable subset $f/(1 + g_s)$ with g_s instantiated from linear, power, exponential, and logarithmic content, so the edit remains local to the parent and avoids bare learned constants in denominators. We intentionally keep this rule library generic and domain-agnostic to reduce dataset-specific overfitting, leaving richer structural refinements to the LLM-guided channel. For multivariate problems, the pair rational and power-law coupling families are instantiated over up to six ordered variable permutations, while single-variable families are instantiated over each feature variable.

Normalization and deduplication To reduce redundant candidates, we define a parameter-name-invariant structural fingerprint $\kappa(f)$. The fingerprint is constructed by recursively replacing all scalar parameter symbols in the AST with a shared placeholder, sorting the children of commutative operators such as `Add` and `Mul`, and serializing the resulting normalized tree. Expressions with the same fingerprint are treated as structurally equivalent up to parameter renaming.

After mutation, candidates are further normalized before evaluation. The normalization procedure applies algebraic simplifications, merges redundant scalar parameters, completes elementary-function parameterizations when necessary, and renumbers parameters in order of appearance as $c_0, c_1, c_2, \ldots$. Examples include rewriting $\exp(A + c)$ as $c' \exp(A)$, rewriting $\exp(c \log g)$ as g^c, simplifying $\log(\exp(g))$ to g, and merging products or sums of scalar parameters into a single parameter.

LLM-guided mutations In addition to rule-based mutations, the LLM Mutator proposes non-template structural refinements for each parent expression. Its input includes the task context, the parent formula, an annotated AST representation, fitted parameter values, summaries of already generated AST-rule mutations, extracted domain knowledge, and top-ranked historical expressions by training NMSE. This information allows the LLM to avoid duplicating existing programmatic mutations while proposing non-template local refinements informed by task context and domain knowledge.

The LLM Mutator uses two edit categories. ADD preserves the full parent and attaches out-of-library or domain-semantic material as a term, factor, or outer wrap. SUBST uses the annotated AST to replace a structurally mismatched subtree, so the full parent need not be preserved. In our implementation, the LLM proposes 20 candidates per parent with no fixed category quota; the prompt asks it to choose the mix according to the parent structure and fitted behavior.

Degeneracy and overfitting screening After constant fitting, each evaluated candidate is screened before it can re-enter the active search frontier. The screening is intentionally coefficient-aware: it uses the fitted constants rather than only the symbolic skeleton. Large-coefficient overfitting is checked only for sufficiently deep expressions, because shallow expressions often require large physical scale factors. Specifically, if the SymPy AST depth of the candidate is at least `overfit_min_depth`, any

Table C5: Categories of LLM-guided mutations.

Category	Description
ADD	Preserve f and attach non-template or domain-semantic content as $f + g$, $f \cdot g$, or $\phi(f)$.
SUBST	Replace a structurally wrong AST subtree with a different expression; the replaced subtree is not preserved.

fitted coefficient with magnitude above 10^3 marks the node as OVERFIT. Independently of depth, the Mutator applies the simplification rules described after Algorithm 1, including small-coefficient term collapse, rational snapping of exponent parameters, elementary identities such as $\exp(0)$ and $\log(1)$, sinusoidal degeneracies, and merging of like terms.

The result is not only diagnostic. A node flagged as OVERFIT is excluded from all future parent-selection prompts. A node flagged as SIMPLIFIED is also excluded in its original form, while the simplified expression is treated as a new candidate and evaluated through the same optimizer and screening path. Thus the flag δ_g restricts search by preventing degenerate high-scoring expressions from consuming Selector and mutation budget, while still preserving useful simplified descendants when the fitted constants reveal a simpler active formula.

Configuration Table C6 summarizes the Mutator configuration used in our symbolic-regression experiments. The deterministic channel is always enabled in the full FunctionEvolve setting, and the LLM channel is configured by the per-model YAML files in `FunctionEvolve`.

Table C6: Mutator configuration used in our experiments.

Parameter	Meaning	Value
`mutation_channels`	Candidate-generation channels in the full setting	AST deletion, AST addition, LLM mutation
`llm_candidates_per_parent`	Number of LLM proposals requested for each selected parent	20
`mutator_seen_topk`	Number of historical best expressions shown to the LLM Mutator	100
`auto_mutation_summary`	Maximum number of rule-generated mutation descriptions shown to the LLM	20
`max_params`	Maximum number of scalar parameters allowed in an evaluated candidate	10
`overfit_min_depth`	Minimum SymPy AST depth for degeneracy and overfitting screening	10
`temperature`	Sampling temperature for *Claude Opus 4.6* Mutator decoding	0.7
`reasoning_effort`	Reasoning setting for *GPT-5.2* Mutator decoding	medium
`max_tokens`	Maximum response length for Mutator decoding	128000

Prompting interface The prompt templates used by the LLM Mutator in the full FunctionEvolve setting are shown below. The system prompt defines the ADD/SUBST edit categories, domain-knowledge constraints, constant-notation convention, output schema, and allowed variables. The user prompt supplies the selected parent, its annotated AST, fitted parameter values, summaries of deterministic rule-generated candidates, and the current historical best formulas. This Mutator template is shared across LLM-SRBench and AI-Feynman; AI-Feynman changes the task background and the knowledge-section heading to dimension/variable-informed structures, but does not use a separate mutation prompt. The no-AST ablation uses the same JSON output contract but switches to a no-AST system prompt and removes the annotated AST block and AST-local substitution guidance from the user prompt.

LLM Mutator system prompt

```
You are a symbolic-regression expert. Propose **{target_num}** diverse
↪  structural **edits** to the parent formula so that each resulting
↪  candidate may fit the data better. Every edit takes the parent expression
↪  and transforms it into one new candidate expression.

## Structural Prior
Do **NOT** assume the ground-truth formula is a sum of terms. It may be a
↪  compact monomial / product / ratio / composition, or a sum of several
↪  mechanistic terms. Let the parent's current fit guide you: if the parent
↪  already captures the trend, prefer small local refinements; if the parent
↪  looks structurally wrong, prefer a rewrite. Do **not** blindly keep
↪  growing the formula additively.

## Two Kinds of Edit
The deterministic engine already covers simple template grafts (linear /
↪  power / exp / log / sin terms and factors, rational fractions, safe
↪  saturating divides, and single wraps) and all prunes / deletions /
↪  simplifications. See "Auto-Generated Candidates (do not repeat)" in the
↪  task. Your job is the part it cannot template:

- **ADDITION** (keep all of `f`; attach new material): `f + g`, `f * g`, or
↪  wrap `f` inside an outer function. Propose **out-of-library /
↪  domain-semantic** content, such as Arrhenius terms, Lorentz factors,
↪  cross-variable couplings, or nested compositions. The whole parent is
↪  preserved.
- **SUBSTITUTION** (discard a subtree, replace it): use the annotated AST to
↪  locate a structurally wrong subtree and replace it with a different
↪  structure (`t -> s`, where `t` no longer appears). Use this when the
↪  parent's shape is mismatched, not when you merely want to add a
↪  correction.

Do **not** propose pure simplifications / deletions -- the program handles
↪  those. Weight your {target_num} proposals toward whichever kind the
↪  parent's fit and structure call for; there is no fixed quota.

Elementary building blocks: power `x**c`, trigonometric `sin`/`cos`,
↪  exponential `exp`, logarithmic `log(1+x)`, and `sqrt`.

## Non-Elementary Function Usage
In addition to the elementary functions above, the following non-elementary
↪  functions are very common in scientific modeling and may be used as
↪  appropriate in either edit category:
- **Abs(x)** (absolute value): suitable for modulus operations in physical
↪  systems, amplitude extraction of sign-alternating variables in
↪  oscillatory systems. E.g. Abs(sin(x)) extracts oscillation amplitude,
↪  Abs(x)**c constructs V-shaped power law, log(1 + Abs(x)) symmetrizes
↪  logarithmic growth. Prefer Abs when a variable can be positive or
↪  negative but only the magnitude matters physically.
- **Max(x, 0)** (positive part / ReLU): suitable for describing piecewise
↪  behavior such as threshold activation, one-sided response. E.g. Max(x -
↪  c, 0) means response only above threshold c, Max(f(x), 0) truncates the
↪  negative part. Suitable for systems with critical points / activation
↪  thresholds.
- Syntax: use `Abs(x)` and `Max(x, 0)` in SymPy.

## Using Domain Knowledge
Design structures informed by the knowledge below. You may introduce a
↪  structure through either kind of edit: as an **ADDITION** (a new factor /
↪  term / wrap that preserves `f`) or as a **SUBSTITUTION** (rewrite a
↪  mismatched subtree). When an edit is motivated by this knowledge, state
↪  the basis in the `mutation` field.
{domain_knowledge}
```

```
## Constant Notation Convention
Constant parameters should always appear in **multiplicative form**, never
↪  alone in a denominator:
- Correct: `c0 * x`, `c0 * sin(c1 * x)`, `c0 / (x + c1)`
- Incorrect: `x / c0`, `sin(x / c0)` -- a constant in the denominator is
↪  numerically unstable. To express a "scaled variable", write `c * var`,
↪  not `var / c`.

## Parameter Naming
Any **newly introduced** constant must use a **fresh** `cK` that does not
↪  appear in the parent formula or its fitted parameters; keep the names of
↪  parent constants you retain.

## Task Background
{context}

## Output Format (strict compliance required)
Return only a JSON array of exactly {target_num} items.

Example item format:
[{{"expression": "<SymPy-parseable formula>", "params": ["c0", "c1"],
↪  "mutation": "ADD: <what changed and why>"}}]

- expression: use c0, c1, c2... for constants to be optimized; variables must
↪  come from the available variable list.
- params: list all parameter placeholders; if no constants, use [].
- mutation: **must** start with `ADD:` or `SUBST:`, followed by a brief
↪  rationale. When the edit is knowledge-driven, name the formula / law /
↪  structure it references.
- All functions must use SymPy/Python syntax: exp(x), log(x), sqrt(x),
↪  sin(x), cos(x), Abs(x), Max(x, 0).

## Available Variables
{variables}
```

LLM Mutator user prompt

```
## Parent Formula
{parent_formula}

## AST Structure
{labeled_ast}

## Fitted Parameters
{parent_fitted_params}

## Auto-Generated Candidates (do not repeat)
{auto_mutations_summary}

## Historical Best Formulas (top {topk})
{top_exprs}

Following the system instructions, propose {target_num} edits of two kinds:
↪  **ADDITION** (out-of-library / domain-semantic terms, factors, or wraps
↪  that preserve the parent) and **SUBSTITUTION** (rewrite a structurally
↪  wrong subtree located via the AST). Do not repeat the auto-generated
↪  template candidates above, and do not propose pure simplifications. There
↪  is no fixed quota -- weight the mix toward what the parent's AST and
↪  current fit suggest.
```

Mutator Ablations For ablations, FunctionEvolve supports three Mutator ablation variants. The first variant disables the LLM mutation channel and keeps only programmatic AST-rule mutations,

namely rule deletion and template-based rule addition. The second variant disables the programmatic rule channel and keeps only LLM-generated ADD/SUBST edits, while still providing the LLM with the standard parent-level structural context. The third variant also uses only LLM-generated ADD/SUBST edits, but further removes the annotated AST representation and replaces AST-local mutation guidance with parent-formula subexpression guidance in the LLM prompt. The second and third variants do not carry a hidden blanket preprocessing stage. This set of variants separates the effects of deterministic AST-rule expansion, LLM-guided local structural refinement, and explicit AST visibility in the mutation interface.

C6 Optimizer

For each candidate expression, the structure-aware coefficient optimizer fits scalar coefficients under a fixed symbolic skeleton and returns the numerical score used by the evolutionary search. Since inaccurate coefficient fitting can make a correct skeleton appear unpromising, this appendix details how FunctionEvolve uses expression structure during coefficient fitting, following the same sequence as the main text.

Expression analysis before optimization Before numerical search, the optimizer parses the candidate into a symbolic expression and extracts parameter-specific constraints from the expression structure. Parameters that appear as exponents over feature-dependent bases are treated carefully, especially when the base can be negative, because arbitrary real exponents can leave the real domain. In this case the implementation evaluates such powers through a real-valued safe-power wrapper and snaps the exponent to the nearest multiple of $1/3$ when needed. The optimizer also identifies parameters that should be positive or searched over a smaller range, such as coefficients inside logarithms, exponentials, trigonometric arguments, or power exponents. The default full-parameter box is $[-100, 100]$; rational-sensitive exponents use $[-10, 10]$, positive parameters use $[10^{-6}, 100]$, small parameters use $[-5, 5]$, and parameters that are both positive and small use $[10^{-6}, 5]$. In addition, data-dependent bounds are computed for parameters controlling exponential growth and feature offsets. For terms of the form $\exp(c_i g(\mathbf{x}))$, the coefficient bound is chosen so that the absolute exponential argument is at most 10 over the training data. For offset parameters in $\exp(g(x - c_i))$ or $(x - c_i)^p$, the search interval is the observed feature range enlarged by $\max(0.2\,\mathrm{range}(x), 10)$.

Variable projection The main optimization path first attempts to separate linear and nonlinear parameters. Many candidates can be written as

$$f(\mathbf{x}; \mathbf{c}) = f(\mathbf{x}; \mathbf{c}_L, \mathbf{c}_N) = \psi(\mathbf{x}; \mathbf{c}_N) + \sum_{j=1}^{m} c_{L,j} \phi_j(\mathbf{x}; \mathbf{c}_N),$$

where $\mathbf{c}_L$ are linear coefficients and $\mathbf{c}_N$ are nonlinear parameters. For any fixed $\mathbf{c}_N$, the optimal $\mathbf{c}_L$ is obtained by ordinary least squares, so the explicit search only needs to explore the nonlinear subspace:

$$\min_{\mathbf{c}_N} \min_{\mathbf{c}_L} \left\| \mathbf{y} - \psi(\mathbf{X}; \mathbf{c}_N) - \sum_{j=1}^{m} c_{L,j} \phi_j(\mathbf{X}; \mathbf{c}_N) \right\|_2^2 .$$

In practice, the optimizer differentiates the expression with respect to candidate parameters, selects a maximal subset that is jointly affine, and constructs the OLS basis functions from the corresponding partial derivatives after setting the linear parameters to zero. The remaining fixed part ψ is subtracted from the target before solving the linear coefficients analytically at each nonlinear parameter assignment.

Structured pre-search for difficult nonlinear parameters Some nonlinear parameters are particularly difficult for continuous optimization. For compound power structures such as $(x + c_1)^{c_2}$, where the power exponent is itself a single parameter and the base contains another parameter, the optimizer first enumerates

$$\{1, 2, 1/2, 3, 1/3, -1, 3/2, 4, 5, -1/2, -2, -3\}$$

for the exponent and runs variable projection on the remaining parameters under each fixed value. This pre-search is executed before the main variable-projection stage and can consume up to 70%

of the variable-projection time budget. Inside variable projection, any remaining power-exponent parameters are handled by a second candidate grid,

$$\{1/3, 1/2, 1, 3/2, 2, 3, 4, 5, -1/3, -1/2, -1, -2, -3\},$$

using a sequential greedy strategy when multiple exponents are present. For each fixed exponent value, the remaining nonlinear parameters are searched by trust-region least squares with five random restarts, and linear coefficients are still solved by OLS. For center or offset parameters appearing in Gaussian-like, exponential-shift, or shifted-power forms, the optimizer detects $(x - c_i)$-style subexpressions and scans 100 equally spaced candidate centers between the observed minimum and maximum of the corresponding feature. These structured pre-search steps provide strong initial basins before the general nonlinear search begins.

Global search over nonlinear parameters After variable projection, the remaining objective over $\mathbf{c}_N$ is lower-dimensional but can still be highly non-convex. FunctionEvolve therefore searches the nonlinear subspace with trust-region least squares, CMA-ES, and differential evolution, each using an independent initialization drawn from the same constraint-aware sampler. The variable-projection path uses the first 45% of the optimizer timeout, with the remaining time in that window divided approximately equally across these three continuous search stages after structured pre-search. This design focuses global exploration on the parameters that determine the nonlinear shape of the expression, while avoiding unnecessary search over linear coefficients that can be solved exactly by OLS.

The fallback condition is explicit in the implementation. Let $\sigma_y^2 = \mathrm{Var}(\mathbf{y}_{\mathrm{train}})$, with a unit fallback for nearly constant targets, and define the early-success threshold $\tau = 10^{-10}\sigma_y^2$. Variable projection is treated as unavailable when no valid decomposition is produced, for example because the expression has no jointly affine coefficient, no basis functions can be constructed, or OLS/basis/fixed-part evaluation becomes non-finite. It is treated as insufficient when the best MSE found so far remains above τ. In either case, the optimizer falls back to full-parameter numerical search until 90% of the total timeout, allocating the remaining fallback budget equally to full-parameter trust-region least squares, CMA-ES, and differential evolution. Thus "insufficient" is not a hidden symbolic criterion; it is the same MSE threshold used for early termination of the optimizer.

Local refinement The best solutions obtained from the preceding stages are then refined with gradient-based local optimization, implemented as L-BFGS-B over the same parameter bounds. This polishing stage uses the final 10% of the total timeout. If parent parameters are available from the evolutionary tree, the optimizer also performs an additional warm-started L-BFGS-B pass after copying the overlapping parent parameter prefix into the current skeleton.

Snap and refit Finally, the optimizer performs optional snapping for exponent-like or rational-sensitive parameters. Parameters that must remain stable over negative bases are snapped to nearby rational values whose reduced denominators are odd. Power exponents are then tested against the same nearby integer/rational grid used above, restricted to candidates within distance 3 of the current value. Crucially, snapping is followed by an L-BFGS-B refit with the snapped exponent fixed; the snapped form is accepted only if the refit preserves or improves the numerical objective. This makes the final expression more stable and interpretable without sacrificing the fitness estimate used by the search.

C7 Optimizer Benchmark Variants

The optimizer benchmark in Section 4.6 is built from ground-truth LLM-SRBench skeletons so that failure can be attributed to coefficient fitting rather than structural search. Numeric constants are replaced by symbolic parameters, parameter couplings are decoupled when possible, and the resulting exact skeleton forms the original-expression group. We then create controlled variants that preserve the same target function under a known parameter assignment. The composite zero-term variants add zero-amplitude nonlinear terms, either $c_a \sin(\log(1 + c_b x) + c_c)$ or $c_a \exp(c_b \cos(c_c x))$ with $c_a = 0$, testing whether an optimizer can ignore unnecessary nonlinear additive structure. The power-augmentation variants introduce identity power transformations by replacing a feature with x^{c_a} or raising the whole expression to c_a, with $c_a = 1$, testing difficult exponent-like parameters. The rational zero-term variants add terms of the form $(c_a x + c_b)/(c_c x + c_d)$, or the analogous numerator

using a second feature, with $c_a = c_b = 0$ and $c_c = c_d = 1$. These four groups are the categories reported in Figure 5.

D Additional Experimental Results

D1 Search-stage Tool-use Audit

We audited the complete search-stage inference logs for the reported *Claude Opus 4.6* run on all 129 LLM-SRBench tasks. Requests contained model configuration and system/user messages, with no tool definitions or tool-choice fields. Responses contained text only, with no tool-use or web-search-result blocks. We also found no URLs or references to public benchmark repositories in the search prompts or outputs. These observations exclude a live web/tool retrieval channel in the audited run, but do not establish that benchmark expressions or related scientific knowledge were absent from pretraining. This audit concerns the search calls, rather than the separate symbolic-equivalence judging stage.

The accompanying code release includes complete LLM call records for the blank-description *w/o AST Structure* setting under `logs/llm-srbench/degen_mutator3_selector2_description-blank`. Other inspected runs were not all uploaded because of repository storage limits. The LangLaw inference logs were also checked for web-search use, and no such channel was used.

D2 LangLaw Evaluation Protocol and Predictive Baselines

We evaluate the two LangLaw tasks with more than 20 samples: bulk modulus (504 samples, 22 inputs) and band gap (745 samples, 27 inputs). Full, *w/o AST Structure*, and No LLM use the same shuffled 80/20 splits with `random_state=42`; all search and candidate selection use training data only. The LLM-based variants use *Claude Opus 4.6*. No LLM uses programmatic search with the structure-aware optimizer. This evaluation does not include LangLaw's separate OOD samples.

For a comparison on identical held-out observations, we evaluate the formulas printed in the LangLaw paper using their published rounded coefficients, without refitting. Their errors on our splits are 0.0837 eV/Å^3 MAE for bulk modulus and 0.673 eV RMSE for band gap. The paper-reported values, 0.0739 and 0.672 respectively, come from a different protocol and are not treated as same-split results. We retain MAE and RMSE in their original units rather than converting published errors using the variance of our test targets. Formula rounding and protocol differences should be considered when interpreting the comparison.

To contextualize band-gap prediction error, we also evaluate XGBoost and an MLP using all 27 inputs on the same 596/149 split. Each model is selected from 64 configurations using shuffled five-fold cross-validation within the training partition; test data are evaluated after model selection. The selected XGBoost uses 1,500 trees of depth 5, learning rate 0.02, and row/feature subsampling of 0.65. The MLP standardizes inputs and targets and uses two ReLU layers of widths 128 and 64, Adam with learning rate 0.003, batch size 32, and early stopping.

Table D1: Band-gap prediction on the same held-out split. Feature counts for symbolic models refer to inputs appearing in their selected expressions.

Model	Features	RMSE (eV)	MAE (eV)	R^2
Published LangLaw formula	5/27	0.673	0.531	0.827
FunctionEvolve	10/27	0.579	0.412	0.872
XGBoost	27/27	0.396	0.196	0.940
MLP	27/27	0.417	0.224	0.934

The black-box models achieve lower prediction error under this protocol; their scores are empirical reference points, not an established error floor. FunctionEvolve produces an inspectable expression involving fewer input variables, but using fewer variables does not imply a simpler formula or a physically correct law. The two tasks provide limited evidence for handling larger candidate-variable sets, rather than a comprehensive high-dimensional generalization study.

LangLaw was first publicly submitted on February 26, 2026 [Guan et al., 2026], after the February 5 release of Claude Opus 4.6 [Anthropic, 2026]. This chronology reduces concern about direct memorization of the newly published benchmark and formulas. It does not prove that the underlying materials data or related domain knowledge were absent from pretraining.

D3 Sensitivity to Seed Initialization

We evaluate initialization sensitivity on all 24 biological population-growth (BPG) tasks using *Claude Opus 4.6*. The controlled variant replaces LLM-generated initial expressions with the fixed programmatic fallback templates. Extracted domain knowledge, the Selector, both AST and LLM Mutators, and the structure-aware optimizer are retained. This differs from *w/o Generator*, which also removes extracted domain knowledge, and from No LLM, which removes LLM guidance throughout search.

The fallback templates use only the first input variable. For BPG, the inputs are time t and population P, so these seeds contain elementary functions of t, including polynomial, trigonometric, and rational forms, but do not initially include P. Later search components retain access to both variables. This deliberately weak initialization tests whether subsequent evolution can recover useful structures despite an uninformative initial pool.

Table D2: Seed initialization on the 24 BPG tasks. Candidates are ranked by training NMSE, and SA@k counts tasks with a symbolic match among the top k.

Initialization	SA@1	SA@5	SA@10	SA@50
LLM-generated seeds	16/24	20/24	21/24	22/24
Programmatic fallback seeds	13/24	16/24	17/24	23/24

LLM-generated seeds improve recovery at $k = 1, 5, 10$, while the two settings reach comparable recovery at $k = 50$. Subsequent evolutionary search can therefore compensate for part of the weak initialization on this subset, but informative seeds help place correct expressions within smaller reported candidate sets. These single-run results on BPG do not establish insensitivity to seed quality across other domains.

D4 Complexity-aware Final Selection Details

The complexity-aware selectors in Section 4.7 are applied only after the evolutionary search has finished. They do not choose parents, generate mutations, fit coefficients, or alter the trajectory. For each task, we collect the full candidate trajectory, preserve the training-NMSE ranking used by the search, and compute simple structural features for each expression. The selectors then choose a fixed reporting shortlist from this already generated pool. Test and OOD NMSE are never used by these selectors.

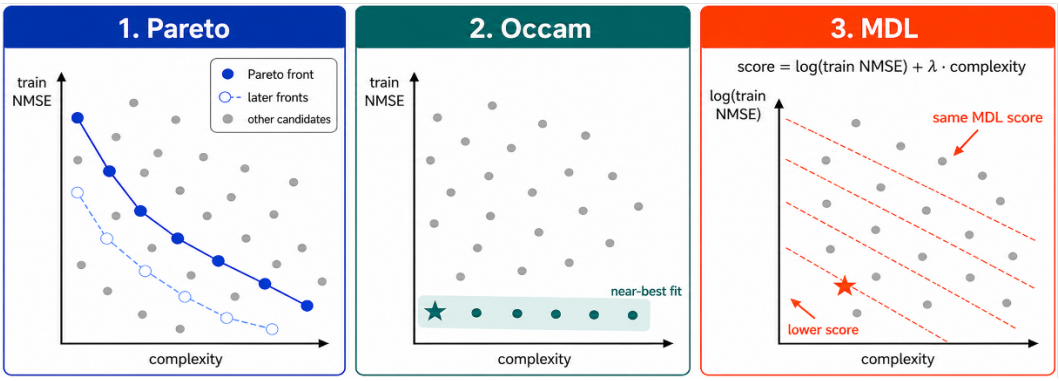


Figure 9: Complexity-aware final-candidate selection rules. Pareto treats each candidate as a point in training-NMSE–complexity space and selects from successive non-dominated fronts; Occam first keeps candidates with near-best training fit and then prefers simpler expressions; MDL combines training loss and complexity into a single scalar score. Details are provided in Appendix D4.

Let candidate i have training error e_i, original training-NMSE rank r_i, and structural features: tree size s_i, operator count o_i, fitted-parameter count p_i, and special-function count h_i. We use a weighted complexity score

$$q_i = s_i + w_o o_i + w_p p_i + w_h h_i,$$

where the weights are fixed before evaluation. Pareto selection performs non-dominated sorting over $(\log e_i, q_i)$: a candidate is dominated if another candidate has no larger value in both objectives and a strictly smaller value in at least one. Candidates from the first front are selected first, ordered by $(\log e_i, q_i, r_i)$; if the shortlist is not full, the selected front is removed and the next front is added in the same way.

Occam selection first restricts the pool to candidates with near-best training fit. Specifically, if $e_\star$ is the best finite training NMSE, the near-best set contains candidates satisfying

$$\log_{10}(\max(e_i, 10^{-300})) - \log_{10}(\max(e_\star, 10^{-300})) \le \Delta.$$

Within this near-best set, candidates are ranked by $(q_i, \log e_i, r_i)$, so simpler expressions are preferred only after the training error is already close to the best observed fit. If the near-best set is too small to fill the shortlist, the rule falls back to the full pool with the same complexity-first ordering.

MDL selection uses a single scalar objective,

$$m_i = \log e_i + \alpha p_i + \beta s_i + \beta_o o_i,$$

and returns the candidates with the smallest m_i, with the original rank r_i used only as a deterministic tie-breaker. This rule approximates a description-length tradeoff: the log training loss rewards fit, while parameter, tree-size, and operator penalties discourage unnecessarily complex formulas. All three selectors therefore test whether post-hoc complexity-aware reporting can recover exact symbolic forms that were generated by the search but ranked below numerically strong surrogate expressions by training NMSE alone.

D5 Non-MatSci Failure Correlation Audit

Table D3 reports the pairwise linear-dependence audit for the twelve FunctionEvolve failure cases (tasks with $\mathrm{SA@50} = 0$) outside the MatSci subset. Unlike MatSci, these domains do not exhibit a dataset-wide input collapse, but several individual failures still contain highly correlated variable pairs. Four of these tasks—CRK0, CRK6, CRK22, and PO17—do eventually recover a symbolically correct expression, but only among candidates ranked beyond the top-50 budget (at ranks 693, 229, 313, and 224, respectively); they therefore count as $\mathrm{SA@50}$ failures here even though they are not failures under the full-trajectory exact-match audit.

Table D3: Pairwise linear dependence in non-MatSci failures.

Case	Variable pair	R^2
BPG4	t, P	0.723803
BPG20	t, P	0.674447
CRK0	t, A	0.208901
CRK6	t, A	0.951128
CRK22	t, A	0.517110
CRK28	t, A	0.999234
CRK29	t, A	0.732757
PO0	x, t	0.001431
PO16	x, t	0.004350
PO17	x, v	0.028476
PO23	x, v	0.964941
PO43	x, v	0.368555

D6 Baseline NMSE Summary

Table D4 summarizes the symbolic and numerical performance of the non-FunctionEvolve baselines evaluated with *Claude Opus 4.6*. Here symbolic accuracy is reported as $\mathrm{SA@50}$ ($\mathrm{SA@1}$), Train/Test/OOD NMSE are split-wise medians over 129 tasks, and Test Acc_τ counts tasks with test maximum relative error at most τ.

The corresponding per-task records are available in the GitHub raw data files described above.

Table D4: Baseline NMSE summary for methods run with *Claude Opus 4.6*. Symbolic accuracy is reported as $\mathrm{SA@50}$ ($\mathrm{SA@1}$). NMSE entries are medians over 129 tasks, and accuracy entries are raw test-set task counts.

Method	$\mathrm{SA@50}$ ($\mathrm{@1}$) ↑	Train NMSE ↓	Test NMSE ↓	OOD NMSE ↓	Test $\mathrm{Acc}_{0.1}$ ↑	Test $\mathrm{Acc}_{0.01}$ ↑	Test $\mathrm{Acc}_{0.001}$ ↑
Direct Prompting	2 (1)	1.24e-3	1.36e-3	7.62e-2	18	7	4
OpenEvolve [Sharma, 2025]	**24** (5)	4.50e-6	4.30e-6	6.52e-3	27	8	1
LLM-SR [Shojaee et al., 2024]	**24** (**11**)	**2.62e-6**	**9.80e-7**	**4.45e-4**	**40**	**13**	**5**

Table D5: Extended ablation results on 129 LLM-SRBench tasks. Entries report $\mathrm{SA@50}$ ($\mathrm{SA@1}$).

Method	Model	Chemistry (36)	Biology (24)	Physics (44)	Materials Science (25)	Total (129)
Full	*GPT-5.2-medium*	**30** (**13**)	**20** (**15**)	**37** (**36**)	**16** (5)	**103** (**69**)
w/o All	*GPT-5.2-medium*	12 (4)	18 (1)	2 (1)	3 (0)	35 (6)
w/o Generator	*GPT-5.2-medium*	21 (9)	**20** (13)	29 (24)	9 (3)	79 (49)
w/o Selector	*GPT-5.2-medium*	20 (**13**)	17 (12)	14 (12)	11 (**7**)	62 (44)
w/o LLM Mutator	*GPT-5.2-medium*	12 (7)	14 (11)	12 (8)	7 (1)	45 (27)
w/o AST Mutator	*GPT-5.2-medium*	27 (12)	18 (14)	14 (13)	11 (3)	70 (42)
w/o Structure-aware Optimizer	*GPT-5.2-medium*	18 (3)	14 (7)	12 (5)	2 (1)	46 (16)
Full	*Claude Opus 4.6*	**31** (**16**)	22 (16)	**39** (**34**)	**15** (**6**)	**107** (**72**)
w/o All	*Claude Opus 4.6*	12 (4)	16 (5)	4 (1)	2 (0)	34 (10)
w/o Generator	*Claude Opus 4.6*	27 (12)	22 (14)	33 (29)	9 (3)	91 (58)
w/o Selector	*Claude Opus 4.6*	25 (13)	20 (11)	20 (18)	9 (4)	74 (46)
w/o LLM Mutator	*Claude Opus 4.6*	11 (8)	15 (11)	13 (10)	7 (2)	46 (31)
w/o AST Mutator	*Claude Opus 4.6*	30 (**16**)	22 (11)	23 (20)	9 (3)	84 (50)
w/o AST Structure	*Claude Opus 4.6*	17 (10)	**23** (**17**)	20 (13)	10 (3)	70 (43)
w/o Structure-aware Optimizer	*Claude Opus 4.6*	17 (6)	17 (4)	16 (12)	3 (0)	53 (22)

D7 Complete Ablation Results

Table D5 reports the domain-wise symbolic-accuracy ablation results for both *GPT-5.2-medium* and *Claude Opus 4.6*. Each entry gives $\mathrm{SA@50}$ ($\mathrm{SA@1}$). Tables D6 and D7 complement this view with aggregate NMSE and numerical-accuracy summaries for the same ablation settings. The two model backbones show the same qualitative pattern: removing the LLM Mutator or replacing the structure-aware coefficient optimizer with L-BFGS produces the largest $\mathrm{SA@50}$ drops, while removing the Generator, Selector, or AST Mutator also consistently reduces symbolic recovery.

NMSE and numerical accuracy summaries The following tables are computed from the NMSE result records and report all ablation settings used in our experiments. The records contain five aggregate rows per run (`BPG_all`, `CRK_all`, `MatSci_all`, `PO_all`, and `Total_all`); these tables use only the 129 individual benchmark tasks. In each table, *Method* names the same setting used in the ablation study, *SA* is the number of exact symbolic matches, Train/Test/OOD NMSE report the median normalized mean squared error on the corresponding split, and Test Acc_τ reports the number of tasks with test maximum relative error at most τ. To keep the appendix compact, we omit the full per-task detail tables; detailed raw data are available at `https://github.com/Phoinikas03/FunctionEvolve`.

GPT-5.2-medium Table D6 reports the *GPT-5.2-medium* backbone. The *w/o Generator*, *w/o Selector*, *w/o LLM Mutator*, *w/o AST Mutator*, *w/o Structure-aware Optimizer*, and *w/o All* rows follow the ablation definitions in Section 4.3: they respectively remove LLM-based initialization, LLM-based parent selection, LLM mutation proposals, AST-rule mutations, the structure-aware coefficient optimizer, and the combined Generator/Selector/AST-Mutator/Structure-aware-Optimizer stack.

Table D6: Complete ablation NMSE summary for *GPT-5.2-medium*. NMSE entries are medians over 129 tasks, and accuracy entries are raw test-set task counts.

Method	SA ↑	Train NMSE ↓	Test NMSE ↓	OOD NMSE ↓	Test $\mathrm{Acc}_{0.1}$ ↑	Test $\mathrm{Acc}_{0.01}$ ↑	Test $\mathrm{Acc}_{0.001}$ ↑
Full	**103**	**1.82e-13**	**2.08e-13**	**2.21e-10**	**111**	**88**	**73**
w/o All	35	4.20e-7	4.27e-7	3.98e-4	49	19	11
w/o Generator	79	2.88e-13	2.42e-13	2.38e-9	107	81	71
w/o Selector	62	3.77e-13	4.63e-13	3.72e-9	99	72	63
w/o LLM Mutator	45	1.24e-12	1.16e-12	1.13e-8	91	67	55
w/o AST Mutator	70	3.77e-13	3.47e-13	4.14e-9	99	74	66
w/o Structure-aware Optimizer	46	4.01e-8	4.33e-8	1.80e-5	69	42	22

The corresponding per-task records are available in the GitHub raw data files described above.

Claude Opus 4.6 Table D7 uses the same column definitions for the *Claude Opus 4.6* backbone. In addition to the main ablation rows, *w/o AST Structure* keeps the LLM Selector and LLM Mutator but removes AST-derived information from their prompts and disables the AST Mutator, while *Rule-only + Structure-aware Optimizer* disables the Generator, the LLM Selector, and the LLM Mutator while retaining the default structure-aware coefficient optimizer.

Table D7: Complete ablation NMSE summary for *Claude Opus 4.6*. NMSE entries are medians over 129 tasks, and accuracy entries are raw test-set task counts.

Method	SA ↑	Train NMSE ↓	Test NMSE ↓	OOD NMSE ↓	Test $\mathrm{Acc}_{0.1}$ ↑	Test $\mathrm{Acc}_{0.01}$ ↑	Test $\mathrm{Acc}_{0.001}$ ↑
Full	**107**	**1.34e-13**	**1.38e-13**	**1.78e-10**	**113**	**94**	**78**
w/o All	34	2.31e-7	1.93e-7	2.15e-4	53	27	15
w/o Generator	91	2.43e-13	2.06e-13	4.23e-10	110	87	75
w/o Selector	74	4.14e-13	4.65e-13	4.71e-9	100	78	63
w/o LLM Mutator	46	1.04e-12	1.01e-12	1.09e-8	89	67	55
w/o AST Mutator	84	2.43e-13	2.06e-13	2.52e-9	108	83	71
w/o AST Structure	70	4.25e-13	5.09e-13	6.73e-9	96	71	59
w/o Structure-aware Optimizer	53	5.37e-9	8.19e-9	5.68e-6	66	43	27
Rule-only + Structure-aware Optimizer	29	1.42e-10	1.47e-10	3.65e-7	78	53	39

The corresponding per-task records are available in the GitHub raw data files described above.

D8 Numerical Accuracy at Stricter Tolerances

We distinguish two tolerance-based task-count metrics. Our Acc_τ follows the convention used in LLM-SRBench reporting and in the main text: a task is counted only when the top-1 prediction has relative error at most τ at every evaluation point, equivalently when its maximum relative error is at most τ. For a single test task, this indicator can be written as

$$\mathrm{Acc}_\tau = \mathbf{1}\left(\max_{1\le i\le N_{\mathrm{test}}} \left|\frac{\hat{y}_i - y_i}{y_i}\right| \le \tau\right).$$

A relaxed variant, denoted $95\%\mathrm{Acc}_\tau$, counts a task when at least 95% of its evaluation points have relative error at most τ.

SR-Scientist [Xia et al., 2025] reports this relaxed $95\%\mathrm{Acc}_\tau$ metric for $\tau = 0.01$ and $\tau = 0.001$, which is not directly aligned with the reporting convention in LLM-SRBench-style papers or with our main-text Acc_τ numbers. Table D8 therefore reports test-set entries as Acc_τ ($95\%\mathrm{Acc}_\tau$): values cited from SR-Scientist appear only in parentheses, while results from our executable runs report strict all-point counts with the corresponding relaxed 95% counts in parentheses. Table D9 uses the same convention for OOD results.

D9 Reliability Audit for LLM-as-a-Judge Symbolic Accuracy

Symbolic accuracy at k counts a task as successful if any of the top-k candidates is mathematically equivalent to the ground-truth expression. In this equivalence check, ground-truth constants are fixed, whereas candidate constants may be reassigned to establish equivalence. Physics-oscillator (PO) variables are allowed to take negative values, so expressions such as `v**c` are not treated as equivalent to `Abs(v)**c`. To reduce dependence on a single model judgment, each verification case is judged independently by *GPT-5.2* and *Claude Opus 4.6*. If their valid verdicts disagree, or if either run returns `verify_error`, the case is manually adjudicated.

We audit 500 verification cases sampled randomly. For *Opus 4.6*, repeated rows caused by retries are deduplicated by `sample_id`; when available, the final non-`verify_error` result is used. Table D10

Table D8: Test-set numerical accuracy at stricter tolerances on the 129-task synthetic subset of LLM-SRBench. Entries report raw task counts as Acc_τ ($95\%\mathrm{Acc}_\tau$), with $\tau = 0.01$ or 0.001; parenthesized-only entries are the relaxed $95\%\mathrm{Acc}_\tau$ results cited from Xia et al. [2025]. Best all-point Acc_τ values are shown in bold, while the ground-truth row is a reference row. Entries marked with * are converted from percentages reported in Xia et al. [2025] using the corresponding domain sizes, with counts rounded independently.

	Chemistry (36)		Biology (24)		Physics (44)		Materials Science (25)		Total (129)	
Model	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$
Direct Prompting										
Claude Opus 4.6	0 (1)	0 (0)	2 (4)	1 (2)	2 (3)	2 (2)	3 (10)	1 (5)	7 (18)	4 (9)
OpenEvolve [Sharma, 2025]										
Claude Opus 4.6	5 (8)	0 (2)	1 (2)	0 (0)	0 (9)	0 (2)	2 (11)	1 (1)	8 (30)	1 (5)
LLM-SR [Shojaee et al., 2024]										
Claude Opus 4.6	5 (9)	1 (1)	1 (6)	1 (1)	0 (7)	0 (2)	7 (19)	3 (10)	13 (41)	5 (14)
SR-Scientist [Xia et al., 2025]										
*Qwen3-Coder-480B-A35B-Instruct**	(15)	(2)	(12)	(6)	(15)	(6)	(22)	(17)	(63)	(32)
GLM-4.5-Air*	(16)	(4)	(10)	(4)	(16)	(7)	(20)	(18)	(62)	(32)
*GPT-OSS-120B**	(29)	(23)	(16)	(10)	(18)	(15)	(19)	(15)	(82)	(64)
*GPT-OSS-20B**	(18)	(8)	(8)	(5)	(13)	(7)	(16)	(10)	(55)	(30)
*Qwen3-Coder-30B-A3B-Instruct**	(8)	(2)	(5)	(2)	(8)	(4)	(20)	(13)	(42)	(21)
Qwen3-Coder-30B-A3B-Instruct + RL*	(13)	(3)	(7)	(3)	(11)	(5)	(21)	(16)	(53)	(27)
FunctionEvolve (Ours)										
GPT-5.2-medium	26 (33)	22 (32)	14 (23)	**9** (23)	28 (36)	24 (32)	20 (25)	18 (24)	88 (117)	73 (111)
Claude Opus 4.6	**27** (35)	**23** (34)	**15** (23)	**9** (23)	**31** (34)	**27** (34)	**21** (25)	**19** (22)	**94** (117)	**78** (113)
Reference: GT equations	27 (35)	24 (35)	14 (23)	10 (23)	36 (39)	33 (39)	25 (25)	25 (25)	102 (122)	92 (122)

Table D9: OOD numerical accuracy at stricter tolerances on the 129-task synthetic subset of LLM-SRBench. Entries report raw task counts as Acc_τ ($95\%\mathrm{Acc}_\tau$), with $\tau = 0.01$ or 0.001; best all-point Acc_τ values are shown in bold.

	Chemistry (36)		Biology (24)		Physics (44)		Materials Science (25)		Total (129)	
Model	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$	$\mathrm{Acc}_{0.01}\uparrow$	$\mathrm{Acc}_{0.001}\uparrow$
Direct Prompting										
Claude Opus 4.6	2 (2)	0 (1)	3 (4)	1 (2)	3 (3)	2 (2)	13 (13)	11 (11)	21 (22)	14 (16)
OpenEvolve [Sharma, 2025]										
Claude Opus 4.6	6 (7)	1 (1)	3 (3)	1 (2)	2 (9)	0 (2)	19 (21)	14 (14)	30 (40)	16 (19)
LLM-SR [Shojaee et al., 2024]										
Claude Opus 4.6	7 (8)	2 (2)	4 (6)	1 (2)	2 (7)	0 (2)	20 (20)	17 (17)	33 (41)	20 (23)
FunctionEvolve (Ours)										
GPT-5.2-medium	31 (34)	26 (31)	**16** (23)	**10** (20)	29 (35)	21 (30)	**25** (25)	**25** (25)	101 (117)	82 (106)
Claude Opus 4.6	**32** (35)	**29** (34)	**16** (23)	**10** (21)	**32** (34)	**24** (33)	**25** (25)	**25** (25)	**105** (117)	**88** (113)

summarizes agreement. The judges agree on 476 valid `match`/`no_match` decisions. There are 21 valid verdict disagreements and three cases involving `verify_error`, so only 24/500 cases require manual intervention. Table D11 reports the resulting judge accuracies after manual adjudication, with and without counting the single unscorable case in the denominator.

Table D10: Reliability audit of the LLM-as-a-judge protocol for symbolic accuracy.

Category	Count	Rate
Total audited samples	500	100.0%
Same valid verdict	476	95.2%
Valid verdict disagreement	21	4.2%
Cases involving `verify_error`	3	0.6%
Manual intervention cases	24	4.8%

Table D11: Judge accuracy after manual adjudication. The single unscorable case had no extractable final candidate formulas.

Judge	Correct / 499	Accuracy / 499	Correct / 500	Strict accuracy / 500
GPT-5.2	482	96.59%	482	96.40%
Claude Opus 4.6	493	98.80%	493	98.60%

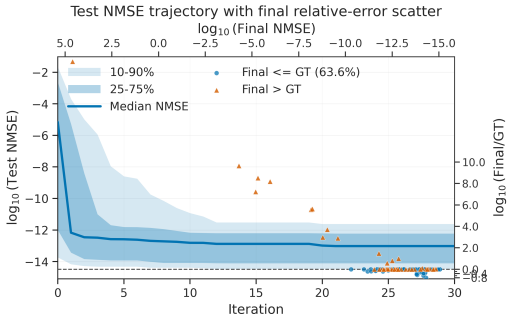
Table D12 lists the 24 cases requiring manual intervention. Among the 23 scorable cases in this set, *GPT-5.2* is correct on 6 and *Opus 4.6* is correct on 17. The only unscorable case is `sample_id=294`, where both judges fail because the log contains no extractable final candidate formulas.

Table D12: Manual adjudication for cases where the two judges disagreed or at least one returned `verify_error`.

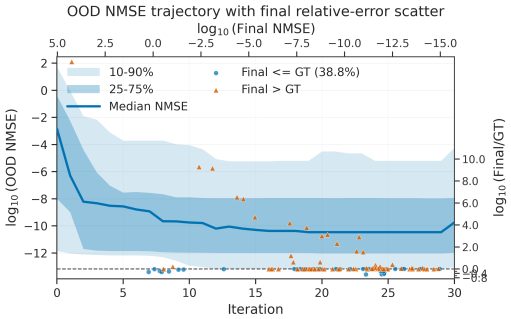
ID	Case	*GPT-5.2*	*Opus 4.6*	Final	Main reason
002	BPG5	`no_match`	`match`	`match`	Rational common-denominator rewriting makes a candidate equivalent.
038	BPG1	`no_match`	`match`	`match`	Candidate has both $P\exp(ct)$ and a rational term that can match after rewriting.
044	BPG23	`match`	`no_match`	`match`	Candidate can represent the polynomial plus $P\exp(cP)$ structure.
080	PO20	`no_match`	`match`	`no_match`	PO domain requires `Abs(v)`; `v**c*x` is not equivalent.
082	PO34	`no_match`	`match`	`no_match`	Same PO-domain absolute-value issue.
112	CRK1	`match`	`no_match`	`match`	Candidate can reduce to polynomial terms plus a phase-shifted log-trigonometric term.
145	MatSci1	`match`	`no_match`	`no_match`	Required Arrhenius term $\exp(-c/T)$ is absent.
169	CRK29	`no_match`	`match`	`match`	Denominator form can represent the rational term and the linear/exponential terms.
173	MatSci14	`no_match`	`match`	`match`	Candidate can represent the required linear-in-T and ϵ^p cross terms.
231	CRK19	`match`	`no_match`	`no_match`	Requires $\sin(\log(A+1))$; candidates have no-phase cosine or wrong log arguments.
257	CRK22	`no_match`	`match`	`match`	Common-denominator rewriting matches the ground truth.
294	MatSci12	`verify_error`	`verify_error`	unscorable	No final formulas or code snippets were extractable from the log.
304	BPG3	`no_match`	`match`	`match`	Candidate has the needed P^3, P^2, and $P\exp(cP)$ basis functions.
315	MatSci8	`match`	`no_match`	`match`	Candidate can represent $T\epsilon^p$, $T\epsilon$, ϵ^p, and ϵ.
362	CRK3	`no_match`	`match`	`match`	Candidate has $A\exp(ct)$, A^2, and a phase-shifted $\sin(\log(A+1))$ term.
363	MatSci2	`verify_error`	`no_match`	`no_match`	Candidates lack the required combination of ϵ^3, $T\exp(-\epsilon)$, and $\exp(-\epsilon)$.
368	MatSci18	`verify_error`	`match`	`match`	Candidate has $\epsilon^2+\epsilon(T+c)^2$.
371	PO12	`no_match`	`match`	`no_match`	`sqrt(v**c)` is not a valid substitute for `Abs(v)**(1/3)` on the PO domain.
374	CRK22	`no_match`	`match`	`match`	Common-denominator rewriting matches the ground truth.
383	MatSci28	`no_match`	`match`	`match`	Candidate contains the required $T\epsilon$, $T\log(\epsilon+1)$, and ϵ^2 terms.
386	PO22	`no_match`	`match`	`match`	Candidate can express $\exp(-\lvert x\rvert)$ via $\exp(c/\sqrt{x^{-2}})$.
391	BPG11	`no_match`	`match`	`match`	Candidate can represent P, P^2, $P^{1/3}$, and $P\exp(ct)$.
415	BPG16	`no_match`	`match`	`match`	Candidate can reduce to $P^3+P^2+P^{1/3}$.
439	BPG1	`no_match`	`match`	`match`	Candidate can match with $P\exp(ct)$ plus a reciprocal rational term.

This audit does not assume that LLM judges are infallible. Instead, it shows that single-judge errors are measurable and rare under our prompt, and that the reported SA@k values use a two-judge-plus-manual-adjudication protocol rather than unchecked single-model judgments. The main *GPT-5.2* error mode is rejecting valid rational-function equivalences that become clear after common-denominator rewriting. The main *Opus 4.6* error mode is over-relaxing domain constraints, especially treating `v**c` as if it could stand for `Abs(v)**c` in PO tasks where negative velocities are allowed.

D10 Test/OOD Trajectories and Ground-truth Numerical Error



(a) Test NMSE trajectory and comparison with ground-truth NMSE



(b) OOD NMSE trajectory and comparison with ground-truth NMSE

Figure 10: Global NMSE trajectories and comparisons between final and ground-truth NMSE. Curves show median $\log_{10}(\mathrm{NMSE})$, and shaded bands show 25–75% and 10–90% ranges using bottom and left axes. Overlaid points use top and right axes: horizontal coordinate is $\log_{10}(\mathrm{NMSE}_{\mathrm{final}})$ and vertical coordinate is $\Delta = \log_{10}(\mathrm{NMSE}_{\mathrm{final}}/\mathrm{NMSE}_{\mathrm{GT}})$. Blue circles indicate $\Delta \leq 0$, and orange triangles indicate $\Delta > 0$.

To further inspect the numerical behavior behind the aggregate NMSE values, Figure 10 shows the test and OOD NMSE search trajectories for the strongest *Claude Opus 4.6* setting, with overlaid scatter points comparing final NMSE against the ground-truth NMSE for each task. On the test split, the global-best median NMSE drops to the numerical floor within the first few iterations and then remains close to the ground-truth NMSE distribution, indicating that the optimizer often fits recovered skeletons to near ground-truth numerical precision. This is also reflected in the paired comparison: the final expression is no worse than the ground-truth expression on 82/129 tasks (63.6%).

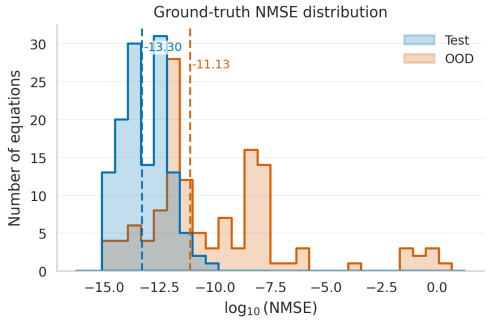


Figure 11: Ground-truth NMSE distributions on the test and OOD splits. OOD has a larger and more dispersed error baseline.

Out-of-distribution evaluation shows the same qualitative error reduction, but the paired comparison is weaker, with 50/129 tasks (38.8%) no worse than ground truth. This gap should be read against Figure 11: even the ground-truth expressions have larger and more dispersed NMSE on held-out OOD samples, with a median about two orders of magnitude higher than on the test split. Thus, this setting has a higher numerical error floor even for the exact laws, and the weaker paired comparison largely reflects the intrinsic difficulty and lower numerical stability of these samples, although task-specific extrapolation effects can still amplify expression-level differences.

D11 Robustness under Noisy Observations

We further evaluate FunctionEvolve on noisy versions of LLM-SRBench using the same *Claude Opus 4.6* setting. As shown in Table D13, exact symbolic recovery is sensitive to observation noise: total SA@50 (SA@1) drops from 107 (72) in the clean setting to 54 (24) under 1% noise and 40 (13) under 5% noise. The numerical criterion also degrades sharply, with test $\mathrm{Acc}_{0.1}$ decreasing from 113 tasks to 39 and 23 tasks, respectively. One possible explanation is that the search ranks candidates by noisy training fit while symbolic accuracy is judged against the clean ground-truth expression. Under noise, structurally different expressions can become numerically plausible explanations of the perturbed samples, so the evolutionary search may favor smooth or overfit alternatives that fit the observations but no longer match the exact law. These results suggest

Table D13: Robustness under noisy observations with *Claude Opus 4.6*. Entries report counts over 129 tasks.

Setting	SA@50 (SA@1)	$\mathrm{Acc}_{0.1}$	Test NMSE
Clean	**107** (**72**)	**113**	**1.38e-13**
1% noise	54 (24)	39	1.32e-6
5% noise	40 (13)	23	3.05e-5

that noisy scientific equation discovery requires uncertainty-aware scoring and validation in addition to stronger symbolic search.

D12 Materials-science Identifiability Audit

To diagnose weaker MatSci results in Section 4, we identify two severe dataset identifiability issues. First, all 25 tasks use strain ϵ and temperature T, but the samples collapse to one dimension: across train, test, and OOD splits, $T \approx 273{+}500\epsilon$. Fitting $T = a{+}b\epsilon$ for each task and split gives correlations within 2.7×10^{-12} of one and a maximum absolute residual of 4.43×10^{-5}. This collinearity makes explicit temperature coupling unidentifiable, so structurally different expressions can achieve identical NMSE by replacing temperature-coupled terms with functions of ϵ alone. Figure 12 shows the residuals from this near-linear geometry. Second, Arrhenius-like factors vary too little to be reliably identified. Eight MatSci equations contain $\exp(-c/T)$ terms, but these factors change by 0.11%–2.06% on train/test and 0.006%–0.108% on OOD. In MatSci5, $\exp(-5.9485/T)$ changes from 0.978446 to 0.989105 on train and from 0.989106 to 0.989672 on OOD. Hence $\epsilon^p \exp(-c/T)$ behaves like a rescaled pure-strain term $C\epsilon^p$, making the Arrhenius structure difficult to distinguish from a constant multiplier.

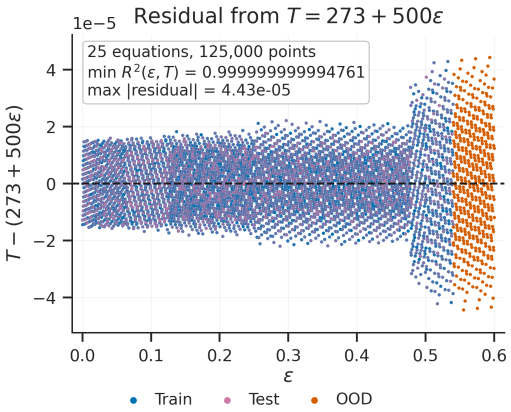


Figure 12: Residuals from the MatSci. Across train, test, and OOD samples, the maximum absolute residual is 4.43×10^{-5}.

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 state the core methodological contributions and the empirical claims on LLM-SRBench; these are supported by the method description, main experimental results, ablation studies, and detailed component configurations in the appendix.

   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 Discussion and Conclusion discuss limitations related to structural identifiability in LLM-SRBench, including collinearity between MatSci inputs and strong pairwise dependence in some non-MatSci failures. The Conclusion also states open directions for extending the method to dynamical systems without closed forms and to problems with larger sets of candidate input variables.

   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: [N/A].

   Justification: The paper does not include theoretical results such as theorems, lemmas, or formal propositions requiring assumptions and proofs; its claims are methodological and empirical, supported by algorithm descriptions, experiments, ablations, and benchmark analysis.

   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 paper describes the benchmark, evaluation metrics, model backbones, search protocol, and symbolic-equivalence audit used for the main experiments. It also provides the complete algorithmic procedure, module configurations, prompt templates, optimizer details, ablation definitions, and raw data links in the appendix, giving a path to reproduce or verify the main experimental results.

   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: Appendix A provides a public repository with the FunctionEvolve implementation, experiment configurations, prompt templates, scripts for running the main experiments and baselines, environment setup instructions, benchmark data access and preparation instructions, and raw result files used to produce the tables and figures.

   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 paper specifies the benchmark, evaluation metrics, model backbones, search protocol, and train/test/OOD reporting used in the experiments. The appendix provides the full algorithmic procedure, module configurations, prompt templates, optimizer details, ablation definitions, and code/data availability information needed to understand and reproduce the experimental settings.

   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: [No].

   Justification: The paper reports raw task counts, median NMSE values, per-domain breakdowns, paired comparisons, and distributional summaries such as 25–75% and 10–90% ranges over tasks, but it does not report repeated-run error bars, confidence intervals, or statistical significance tests for the main experimental comparisons. Repeating the full symbolic-search evaluation multiple times is expensive, with a single full-dataset run costing approximately $2,000.

   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 describes CPU resources for expression evaluation and coefficient optimization, GPU resources for local LLM inference, and call and token usage for externally hosted LLMs.

   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 evaluates a symbolic-regression algorithm on existing scientific benchmark tasks and synthetic equation-discovery datasets, without human subjects, personal or sensitive data, deception, or high-risk deployment. The paper also provides code and data access information for reproducibility in Appendix A.

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: [N/A].

Justification: The introduction discusses the positive impact of symbolic regression for interpretable scientific discovery, extrapolation, and knowledge transfer. The work is foundational algorithmic research on symbolic regression benchmarks, with no human-subject data, personal or sensitive data, or direct deployment setting considered in the paper; we therefore do not identify direct negative societal impacts.

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 a high-risk model or dataset. It provides algorithm code, experiment configurations, prompt templates, and benchmark-result artifacts for reproducing symbolic-regression experiments on existing scientific benchmark tasks, so special safeguards for high-risk release are not applicable.

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 paper credits the creators of the existing benchmarks and baseline methods through citations, including LLM-SRBench and prior LLM-guided symbolic-regression systems. Appendix A also lists the official URLs and licenses for the existing benchmark and baseline assets used in the experiments, including LLM-SRBench/LLM-SR under the MIT license and OpenEvolve under the Apache-2.0 license.

    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: Appendix A documents the new assets released with the paper, including the FunctionEvolve implementation, experiment configurations, prompt templates, reproduction scripts, benchmark data preparation instructions, command-line examples, and raw result files used to produce the tables and figures.

    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 paper does not involve crowdsourcing, human-subject experiments, participant instructions, compensation, or human data collection. All experiments are algorithmic evaluations on existing symbolic-regression benchmark tasks.

    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: The paper does not involve crowdsourcing, human-subject experiments, study participants, or human data collection. Therefore participant-risk disclosure and IRB or equivalent approval are 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: [Yes].

    Justification: LLMs are core components of FunctionEvolve: they generate initial expressions, select parents, and propose mutations, as described in Section 3. Appendices C1, C2, C4, and C5 provide the corresponding procedures, prompt templates, and configurations.

    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.