RISR: RESIDUAL-INFORMED SCIENTIFIC EQUATION DISCOVERY WITH LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Symbolic regression combines structural search with numerical fitting, but aggregate fit scores do not describe how the remaining error varies across inputs. We introduce RISR, a residual-informed method that uses these error patterns to guide formula discovery and learn which corrections are worth fitting. A residual encoder compresses aligned inputs, targets, current predictions, and residuals into continuous tokens that condition a language model to propose formulas. For subsequent refinement, a dual-view relational encoder uses additive and regularized multiplicative residuals to predict the post-fit utility of candidate corrections. We evaluate RISR on scientific tasks from the LLM-SRBench. RISR achieves 63.57% and 38.50% ID accuracy at the 1% and 0.1% pointwise relative-error tolerances, respectively. The corresponding OOD accuracies are 56.07% and 38.24%. RISR outperforms the reported baselines using the same backbone. The results show that our residual-informed approach can improve numerical equation recovery.

1 INTRODUCTION

Symbolic regression (SR) discovers explicit mathematical relationships from observations, providing interpretable models for scientific analysis (Schmidt & Lipson, 2009; Udrescu & Tegmark, 2020; Cranmer, 2023). These expressions make dependencies between variables explicit and allow proposed relationships to be examined and compared. Discovering an equation requires both discrete search over expression structures and continuous estimation of numerical coefficients. The two are closely coupled: the usefulness of a structural term depends on its fitted coefficients and its combination with the rest of the equation. A discovery procedure must therefore explore candidate structures while accounting for their numerical behavior on the observations.

Existing methods address this coupled problem through different representations and search strategies. Sparse identification selects a small set of terms from a candidate function library (Brunton et al., 2016). Neural expression search learns a sampling policy through risk-seeking policy gradients (Petersen et al., 2021), while hybrid frameworks integrate neural search, genetic programming, and other complementary strategies (Landajuela et al., 2022). Another line of work trains expression generators on synthetic equation–observation pairs, learning to map numerical observations to symbolic expressions (Biggio et al., 2021; Kamienny et al., 2022). Together, these approaches provide mechanisms for proposing structures and organizing exploration of the expression space.

Recent work increasingly uses numerical evaluation to guide large language models (LLMs) during search. TPSR incorporates fitting accuracy and expression complexity into Transformer decoding through Monte Carlo tree search (Shojaee et al., 2023). For LLM-based program generation, FunSearch combines a pretrained language model with an evaluator in an evolutionary search procedure (Romera-Paredes et al., 2024). LLM-based equation-discovery systems use scientific knowledge and executable programs to construct and revise candidate relationships (Grayeli et al., 2024; Shojaee et al., 2025a; Xia et al., 2026). LLM-SR scores fitted candidates by their mean squared error and uses a score-organized experience buffer to construct subsequent prompts (Shojaee et al., 2025a). SR-Scientist further allows the model to analyze data and residual errors through code execution (Xia et al., 2026).

The form of numerical feedback determines what guidance is available for the next proposal. Aggregate fit scores rank candidate equations, but do not preserve the location or variable dependence

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

of their errors. Equations with comparable overall losses can therefore require different corrections. Pointwise residuals expose how the remaining error varies across inputs, including nonlinear patterns, variable interactions, and localized discrepancies. Turning this information into useful search guidance requires accounting for the current equation and the effects of numerical fitting. A correction suggested by a residual pattern may produce different improvements depending on its parameterization and how it is combined with the current equation. We study how to encode these numerical states for formula generation and how to learn which corrections are worth fitting.

We introduce **R**esidual-**I**nformed **S**ymbolic **R**egression (RISR), a method for scientific equation discovery. Its search encoder compresses aligned inputs, targets, predictions, and residuals into continuous tokens that condition LLM proposals (Li & Liang, 2021). These proposals are complemented by a learned structural-edit generator. Candidate formulas are fitted and evaluated to obtain an initial equation. For subsequent refinement, a dual-view encoder captures variable-specific patterns and interactions from additive and regularized multiplicative residuals. It models relationships between variables through attention, following the principle of attention-based set encoding (Lee et al., 2019). A scoring head uses these representations to predict the utility of candidate corrections after coefficient fitting. Corrections are fitted numerically and accepted when the training loss improves, after which the residuals and action ranking are refreshed.

The encoders are trained separately using synthetic data constructed from randomly transformed and composed elementary functions, independently of benchmark equations and observations. The search encoder and its projection are trained from scratch on residual-to-formula examples while the LLM remains frozen. The refinement encoder and scoring head learn from paired current equations with shared observations and candidate actions. Corrections are fitted on one subset of observations, and their outcomes on a disjoint subset provide utility supervision.

We evaluate RISR on scientific problems in the released LLM-SRBench data, covering materials science, chemistry, biology, and physics (Shojaee et al., 2025b). With Qwen3-Coder-30B-A3B-Instruct as the LLM backbone, RISR achieves 63.57% and 38.50% accuracy at the 1% and 0.1% relative-error tolerances. Its performance exceeds that of existing non-LLM or LLM-based methods with the same LLM as backbone and remains competitive with substantially larger backbones. Our contributions are summarized as follows:

- We introduce RISR, a residual-informed method that guides both formula search and subsequent refinement. A search encoder conditions LLM proposals on numerical error patterns, while a dual-view refinement encoder supports correction selection.
- We construct complementary synthetic datasets for training RISR. Random transformations and compositions generate residual-to-formula examples, while paired current equations with shared observations and candidate actions provide fitted-utility supervision on observations held out from coefficient fitting.
- We evaluate RISR on scientific problems. Our method improves overall accuracy over the reported 30B baselines and achieves the highest overall ID accuracy at the 1% tolerance among the baselines, including those using larger backbones.

2 Related work

Symbolic equation discovery Symbolic regression combines expression search with numerical evaluation to recover mathematical relationships from data (Schmidt & Lipson, 2009). To make this search more efficient, physics-inspired methods such as AI Feynman exploit properties including symmetry and separability to simplify the search (Udrescu & Tegmark, 2020). PySR uses a multi-population evolutionary procedure that alternates expression evolution, simplification, and coefficient optimization (Cranmer, 2023). Moving beyond explicit search procedures, neural approaches learn from generated equations and observations: NeSymReS pretrains a Transformer to predict symbolic expressions, while end-to-end symbolic regression predicts expressions and constants jointly before numerical refinement (Biggio et al., 2021; Kamienny et al., 2022). TPSR incorporates fitting accuracy and expression complexity into Transformer decoding through Monte Carlo tree search (Shojaee et al., 2023).

The score of a fitted equation does not determine which structural correction will improve it. The outcome also depends on the current residual pattern, the correction's parameterization, and its composition with the equation. RISR addresses this decision by learning correction utility from

fitted outcomes. Its dual-view encoder represents additive and regularized multiplicative correction signals, while a shared scorer uses these representations to rank corrections before fitting.

LLM-based Symbolic Regression LLM-based methods combine program generation with executable evaluation. FunSearch places a pretrained LLM and an evaluator within an evolutionary program-search loop (Romera-Paredes et al., 2024). LLM-SR combines language-model proposals, parameter fitting, and evolutionary search for SR (Shojaee et al., 2025a). LaSR learns a textual concept library from high-performing hypotheses and uses it to guide subsequent search (Grayeli et al., 2024). Alternatively, SR-Scientist leverages agents for data analysis, equation evaluation, and iterative optimization, including residual inspection through code execution (Xia et al., 2026).

Beyond making residuals available as agents' input, a central question is how to represent the current numerical state for subsequent formula proposals. RISR learns this representation from synthetic residual-to-formula examples. It jointly processes aligned inputs, targets, current predictions, and residuals, compressing them into fixed-size continuous tokens that condition a frozen LLM alongside the textual search context.

3 RISR

Given a set of training observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^d$ contains the input variables of observation i and $y_i \in \mathbb{R}$ is its observed response, RISR seeks an explicit equation that predicts the response from the inputs. As shown in Figure 1, RISR proceeds in two stages. In stage A, residual-informed formula search uses a search residual encoder to supply pointwise numerical feedback to a frozen LLM through continuous tokens. LLM proposals and learned structural edits are numerically fitted and evaluated to obtain an initial equation p_0. In stage B, residual-guided refinement then uses a dual-view refinement encoder and utility head to rank correction actions for the current equation. Selected corrections are fitted and accepted when they improve the training loss.

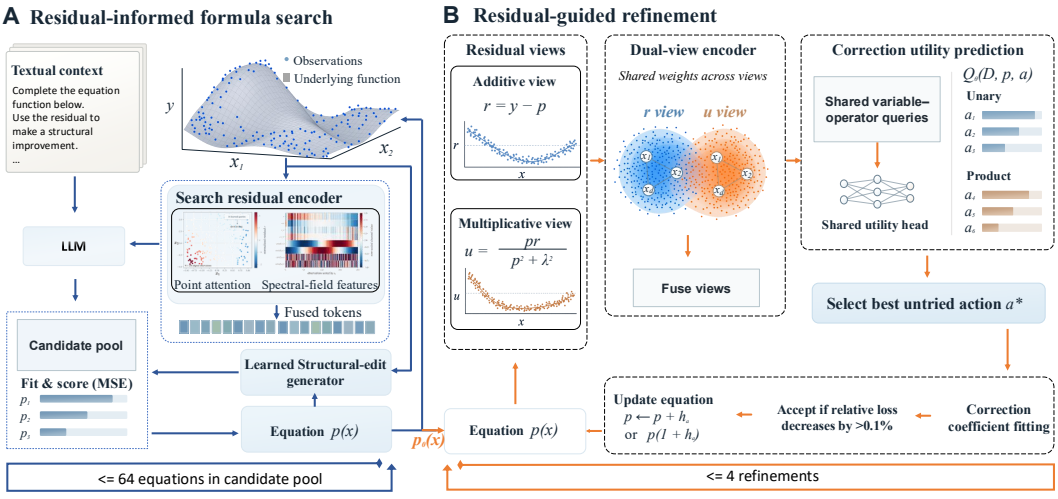


Figure 1: Overview of RISR. (A) Residual-informed formula search conditions a frozen LLM on textual context and numerical tokens from the search residual encoder. LLM proposals and learned structural edits are fitted and evaluated to select an initial equation p_0. (B) Residual-guided refinement encodes additive and regularized multiplicative residual views and predicts the fitted utility of unary and product corrections.

3.1 Residual-Informed Formula Search

Inspired by recent work on LLM-guided program search and symbolic regression (Romera-Paredes et al., 2024; Grayeli et al., 2024; Shojaee et al., 2025a), we iteratively generate and evaluate equation programs. The LLM receives two complementary inputs: textual context describing previously evaluated equations and numerical tokens representing a fitted parent's residual state.

Textual context. Each prompt contains equation programs sampled from the candidate pool. The highest-scoring equation included in the prompt is the parent equation used for numerical conditioning. The textual context provides the equation-completion instruction, variable descriptions, and sampled programs.

Numerical residual. The search residual encoder processes aligned tuples $(x_i, y_i, p(x_i), y_i - p(x_i))$. A point-attention path aggregates these observations, while a spectral-field path represents coordinate–residual interactions using frequency features and residual-region pooling. Learned gates fuse the two paths, and a projection maps their outputs to 16 continuous tokens in the LLM embedding space. These tokens accompany the textual context, conditioning executable formula proposals on the parent's input-dependent error pattern. A learnable structural-edit generator also supplies candidates. Given the observations and current formula, the generator combines its own residual encoder with an encoding of the formula's abstract syntax tree (AST) to predict an addition, replacement, deletion, or stop action. The resulting equation p_0, with the lowest mean squared error (MSE), initializes downstream refinement. Appendix A gives more details of the encoder architecture and proposal procedures.

3.2 Residual-Guided Refinement

Stage B constructs residual views, scores candidate actions, and fits the selected correction.

3.2.1 Dual-View Residual Encoding

For a current equation p, let $p_i = p(x_i)$ be its prediction and $r_i = y_i - p_i$ its residual. We consider two ways to correct its predictions. Let $h(x)$ denote a candidate correction. 1) An additive update, $p + h$, adds an output increment, so matching the observations requires $h(x_i) \approx r_i$. 2) A multiplicative update, $p(1 + h)$, applies a relative adjustment to the current prediction. Because its output increment is $p_i h(x_i)$, matching the observations requires $p_i h(x_i) \approx r_i$. The two update modes therefore motivate different correction signals for the same residual state.

The additive view uses r_i directly. For the multiplicative view, direct division by p_i would be unstable near zero, so we use the regularized signal:

$$u_i = \frac{p_i r_i}{p_i^2 + \lambda^2}. \tag{1}$$

Here λ is 5% of the prediction root mean square (RMS), with a positive floor. The dimensionless quantity u_i approximates the fractional adjustment r_i/p_i when $|p_i|$ is large relative to λ, and suppresses unstable ratios near zero. Appendix B gives more details.

The two residual views shown combine the same input coordinates, targets, and predictions with their respective correction signals, r and u. Each view summarizes correlations between transformed coordinates and observation signals. Pairwise statistics capture relationships between variables and the alignment of product corrections with each view's signal.

The two views share an embedding network and two relation-attention blocks that encode the variables. Pair features contribute to both attention biases and values. For each variable, we fuse its representations from the two views with a pooled global representation, producing $\mathbf{h}_j$ for variable j. This set-structured design shares parameters across variables and their relationships (Zaheer et al., 2017; Lee et al., 2019). Padding and masks support inputs with up to eight variables.

3.2.2 Correction Utility Prediction

Each correction action selects an additive or multiplicative update, one variable or two distinct variables, and an operator for each selected variable. We use eight operator families:

$$\Phi = \left\{ z,\ z^2,\ z^3,\ \sin z,\ \tanh z,\ \exp z,\ \log(1 + z^2),\ (1 + z^2)^{-1} \right\} \tag{2}$$

where z denotes a dimensionless scalar argument. A unary action applies one operator to one variable, while a product action combines two transformed variables. Masks exclude actions involving inactive variables.

A shared query network combines each variable representation $\mathbf{h}_j$ with an operator embedding $\mathbf{e}_k$ to form a variable–operator query. For product actions, the two queries are combined symmetrically and augmented with their product correlations from both residual views. A shared scoring

head combines this representation with the update mode and action type to predict the fitted utility $Q_\theta(\mathcal{D}, p, a)$. Appendix B specifies the action encoding and network architecture.

3.2.3 Coefficient Fitting and Adaptive Updates

Coefficient fitting. For the selected action, numerical fitting determines the coefficients of its correction function. For a subset S of observation indices, we define the relative squared-error objective

$$L_S(f) = \frac{1}{|S|} \sum_{i \in S} \left(\frac{f(x_i) - y_i}{s_{i,S}} \right)^2, \tag{3}$$

where $s_{i,S}$ is the target magnitude with a positive floor scaled to the target RMS on S. During refinement, fitting and acceptance use all training observations.

An action with selected variables J and operators ϕ_{k_j} defines the correction

$$h_a(x) = \alpha \prod_{j \in J} \phi_{k_j}(b_j z_j + c_j) + \beta, \tag{4}$$

where z_j is the standardized value of input variable j and $|J| \in \{1, 2\}$. The parameters b_j, c_j control each operator's input scale and shift, while α, β control the correction's amplitude and offset. The selected update mode gives either $p + h_a$ or $p(1 + h_a)$. We fit the correction by variable projection: for each setting of the nonlinear parameters b_j, c_j, weighted least squares determines the amplitude α and offset β (Golub & Pereyra, 1973; O'Leary & Rust, 2013).

Adaptive action selection. At each attempt, the frozen scorer selects the highest-scoring untried action. The fitted correction is accepted if it reduces the training relative squared error by more than 0.1%. After acceptance, the updated equation determines new residual views and a refreshed action ranking. Rejected actions leave the equation unchanged, and every attempted action is marked as tried. Refinement uses at most four action-fitting attempts, including rejected attempts, and makes no additional LLM calls. Appendix B gives the point-selection protocol, numerical fitting settings, and early-stopping criterion.

3.3 Synthetic Data and Model Training

We train the search residual encoder, structural-edit generator, and refinement model separately using synthetic data generated from elementary functions with sampled transformations, coefficients, and input coordinates, independently of the benchmark equations and observations. All three components are trained offline, while the pretrained LLM parameters remain frozen. Appendix C provides more details.

3.3.1 Search residual Encoder Data and Training

Target formulas combine randomly transformed elementary functions into compositional products, smooth regime transitions, or condition-gated responses. Current equations omit components or use affine approximations. Each example pairs a residual state from 256 observations over one to six variables with its target formula. Validation rejects nonfinite, nearly constant, or excessively large targets and checks numerical alignment and residual consistency. All target programs pass parsing and compilation. Audits confirm no cross-split overlap in example identities or exact normalized numerical inputs. The dataset contains 6,000 training and 768 validation examples. We train the search residual encoder and projector from scratch by minimizing the LLM loss on target-formula tokens, with the pretrained LLM parameters frozen.

3.3.2 Structural-Edit Data and Training

For structural-edit generator training, we generate 20,000 training and 2,000 validation examples, each with 256 observations over one to six variables. Current formulas omit a target component, contain an extra or incorrect component, or already match the target, providing supervision for addition, deletion, replacement, and stopping. Numerical evaluation of candidate edits supplies outcome regression and ranking targets. We train the program encoder and editing modules using supervised edit losses, parameter and outcome regression, and candidate ranking.

3.3.3 REFINEMENT-ENCODER DATA AND TRAINING

Refinement-encoder dataset. Targets combine three to six unary or product components. For each target, we construct paired current equations with different residual patterns, covering both coarse and near-target states. Each pair shares 256 observations and eight candidate actions selected by frozen policies and random sampling. Sharing observations and actions allows supervision to capture how an action's utility changes with the current equation.

The splits contain 8,192 training pairs, 1,024 validation pairs, and three test sets of 1,024 pairs covering new observation clouds, held-out product actions, and eight-variable inputs. Together, they provide 24,576 states and 196,608 action-utility labels. Validation checks action alignment, variable eligibility, and acceptance decisions.

Fitted-action utility. For each synthetic problem, we divide the 256 observations into 192 fitting rows F and 64 utility rows H. Rows in F supply the encoder inputs, coefficient fits, and acceptance decisions; rows in H measure the resulting utility. The losses L_F and L_H apply Eq. 3 to these respective subsets.

For each current equation and candidate action, we fit the correction on F and accept it if it reduces L_F by more than 0.1%. Let p'_a denote the accepted equation, or the unchanged parent when the action is rejected. Its supervision target is the clipped log reduction in error on H:

$$U(p, a) = \mathrm{clip}_{[-3,3]} \left[\log_{10} \frac{L_H(p) + \varepsilon}{L_H(p'_a) + \varepsilon} \right], \tag{5}$$

where $\varepsilon > 0$ stabilizes the logarithm. Rejected actions have zero utility, while accepted actions receive positive or negative utility according to their effect on H. The target therefore measures the outcome of fitting a correction beyond the rows used to estimate its coefficients.

Training objective. We jointly train the relational encoder and action scorer. The loss is:

$$\mathcal{L} = \mathcal{L}_{\text{value}} + 0.5\mathcal{L}_{\text{rank}} + 0.5\mathcal{L}_{\Delta}. \tag{6}$$

The value term $\mathcal{L}_{\text{value}}$ regresses measured utilities. The ranking term $\mathcal{L}_{\text{rank}}$ learns their ordering within each state (Burges et al., 2005). The paired-state term $\mathcal{L}_{\Delta}$ matches utility differences for the same action across the two current equations. We train with AdamW (Loshchilov & Hutter, 2019) for 16 epochs, corresponding to 2,048 updates, at a learning rate of 3×10^{-4}.

4 EVALUATION

4.1 EVALUATION SETTING

Dataset As many datasets may have been included in LLM pretraining corpora, LLM-SRBench (Shojaee et al., 2025b) combines known terms in the underlying equations with novel, synthetically generated terms to create problems that go beyond memorization. To ensure scientific rigor, two subject-matter experts verified all equations and visualizations of the corresponding generated data. We evaluate our approach on LSR-Synth, following Xia et al. (Xia et al., 2026). It contains 25 problems in materials science, 36 in chemistry, 24 in biology, and 44 in physics. Each problem contains a training set for SR methods, along with separate in-domain (ID) and out-of-domain (OOD) test sets.

Evaluation Metrics We report accuracy-to-tolerance following the released SR-Scientist (Xia et al., 2026; Kamienny et al., 2022; Biggio et al., 2021). Point i passes when $|\widehat{y}_i - y_i|/|y_i| \leq \tau$. Writing this pointwise indicator as $b_i(\tau)$, task success is

$$\mathrm{Pass}_\tau(f) = \mathbf{1}\left[\frac{1}{N} \sum_{i=1}^{N} b_i(\tau) \geq 0.95 \right]. \tag{7}$$

Accuracy is the percentage of tasks that pass, reported at $\tau = 0.01$ and 0.001.

Table 1: Scientific ID accuracy (%), measured by $\mathrm{Acc}_{1\%}$ and $\mathrm{Acc}_{0.1\%}$. The best and second-best results are highlighted in bold and underlined, respectively.

Method	Overall		Materials		Chemistry		Biology		Physics	
	$\mathrm{Acc}_{1\%}$	$\mathrm{Acc}_{0.1\%}$	$\mathrm{Acc}_{1\%}$	$\mathrm{Acc}_{0.1\%}$	$\mathrm{Acc}_{1\%}$	$\mathrm{Acc}_{0.1\%}$	$\mathrm{Acc}_{1\%}$	$\mathrm{Acc}_{0.1\%}$	$\mathrm{Acc}_{1\%}$	$\mathrm{Acc}_{0.1\%}$
Without LLMs										
E2E	0.26	0.00	1.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NeSymReS	3.10	0.78	8.00	4.00	2.78	0.00	4.17	0.00	0.00	0.00
DSR	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
uDSR	29.46	12.40	36.00	8.00	50.00	25.00	29.17	8.33	9.09	6.82
PySR	29.46	14.47	53.33	22.67	25.93	11.11	16.67	6.95	25.76	16.67
GLM-4.5-Air (Zeng et al., 2025)										
LaSR	14.21	8.53	18.67	12.00	16.67	10.18	11.11	5.56	11.36	6.82
LLM-SR	35.92	14.47	61.33	38.67	30.56	3.71	18.06	4.17	35.61	15.15
SR-Scientist	48.32	25.06	81.33	**70.67**	45.37	11.11	40.28	16.66	36.37	15.15
Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025)										
LaSR	11.89	7.49	13.33	6.67	16.67	9.26	6.95	6.95	9.85	6.82
LLM-SR	41.08	18.09	80.00	52.00	36.11	9.26	30.56	18.06	28.79	6.06
SR-Scientist	49.09	24.55	86.67	69.33	40.74	5.56	50.00	26.39	34.09	13.64
GPT-OSS-120B (Agarwal et al., 2025)										
LaSR	16.02	10.08	20.00	12.00	18.52	12.96	9.72	5.56	15.15	9.09
LLM-SR	28.16	11.37	64.00	38.67	22.22	3.71	11.11	2.78	21.97	6.82
SR-Scientist	**63.57**	**49.35**	74.67	60.00	**81.48**	**64.81**	**66.67**	**43.05**	40.91	34.09
Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025)										
LaSR	12.66	8.27	20.00	10.67	17.59	12.04	8.33	6.95	6.82	4.55
LLM-SR	24.55	7.24	29.33	6.67	30.55	10.18	15.28	5.55	21.97	6.06
SR-Scientist	32.30	16.02	81.33	52.00	22.22	5.56	22.22	8.33	18.18	8.33
SR-Scientist + RL	40.92	20.69	85.33	65.33	37.38	7.46	29.17	11.11	25.00	11.37
RISR	**63.57**	38.50	**89.33**	58.67	62.04	30.56	50.00	25.00	**57.58**	**40.91**

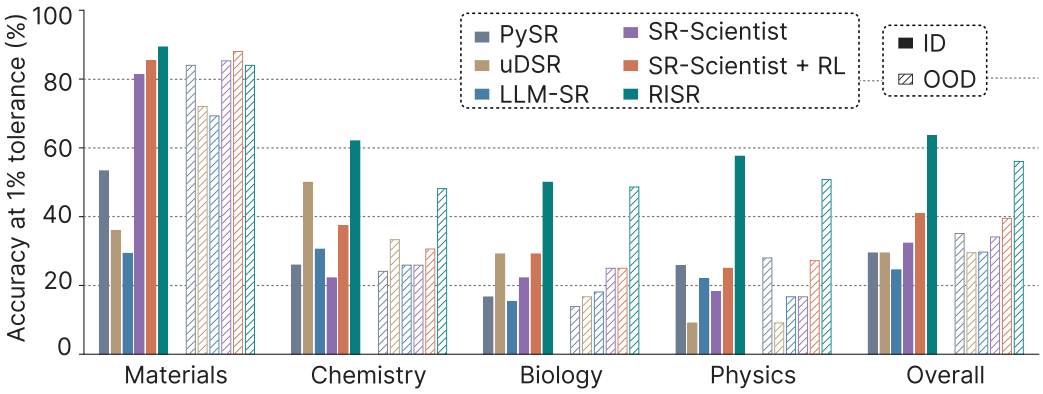


Figure 2: ID and OOD accuracy at a 1% relative-error tolerance. Colors denote methods. Filled and hatched bars represent ID and OOD results, respectively. All LLM-based methods use Qwen3-Coder-30B-A3B-Instruct.

Baselines We compare RISR against both non-LLM and LLM-based symbolic regression methods. The non-LLM baselines include E2E (Kamienny et al., 2022), NeSymReS (Biggio et al., 2021), DSR (Petersen et al., 2021), uDSR (Landajuela et al., 2022), and PySR (Cranmer, 2023), while the LLM-based baselines comprise LaSR (Grayeli et al., 2024), LLM-SR (Shojaee et al., 2025a), and SR-Scientist (Xia et al., 2026). We use the published results reported by (Xia et al., 2026).

Implementation and Protocol We use Qwen3-Coder-30B-A3B-Instruct as the search backbone. Its pretrained parameters remain frozen during encoder training and evaluation. Each search has 64 outer candidate slots, including initialization. LLM proposals and learned structural edits share

Table 2: The ablation results with one seed setting. Accuracy is reported at the 1% / 0.1% tolerances. For Stage B, all rows share the same Stage-A initial equations and numerical fitting protocol.

Configuration	ID accuracy ↑	OOD accuracy ↑
LLM-only formula search	7.75 / 0.00	9.30 / 3.10
+ residual-informed formula search (Stage A)	36.43 / 23.26	40.31 / 30.23
+ coefficient fitting	59.69 / 34.11	53.49 / 38.76
+ random corrections	60.47 / 34.88	52.71 / 36.43
+ learned-scored corrections (RISR)	63.57 / 37.21	55.04 / 38.76

this budget. Stage B first fits the selected equation's coefficients and then allows at most four new correction-fitting attempts, including rejections, without further LLM calls. Within-run equation selection and fitting use training observations, leaving the test set for evaluation.

4.2 Main Results

ID Performance Table 1 reports the ID performance of RISR, averaged across results of three seeds. More details can be found in Appendix D. Among methods using the same backbone, RISR achieves 63.57% and 38.50% accuracy at the 1% and 0.1% tolerances, respectively, exceeding SR-Scientist+RL by 22.65 and 17.81 percentage points. The gains span all four domains at the 1% tolerance, with particularly large improvements in chemistry and biology: 24.66 and 20.83 percentage points, respectively. This breadth indicates that the overall improvement is shared across scientific domains rather than concentrated in a single problem category.

The 30B model results are also competitive with systems built on substantially larger backbones. RISR exceeds the strongest reported Qwen3-Coder-480B result in seven of the ten dimensions. At the 1% tolerance, across all baselines, it matches the highest reported overall accuracy, 63.57% for SR-Scientist with GPT-OSS-120B. At the stricter 0.1% tolerance, its 38.50% accuracy ranks second overall, ahead of all reported GLM-4.5-Air and Qwen3-Coder-480B configurations. These results demonstrate the effectiveness of the complete RISR pipeline at the 30B scale and show that leading accuracy at the 1% tolerance can be achieved without adopting a larger LLM backbone.

As for domains, the strongest cross-backbone results occur in materials science and physics. RISR achieves the highest materials accuracy at the 1% tolerance, reaching 89.33%. In physics, it leads at both tolerances with 57.58% and 40.91%, exceeding the strongest competing results of 120B model by 16.67 and 6.82 percentage points, respectively. In chemistry, RISR ranks second at both tolerances. In biology, it ties for second place at the 1% tolerance.

OOD performance RISR achieves the highest overall OOD accuracy among the compared methods at the same 30B model scale (Figure 2). At the 1% tolerance, its accuracy reaches 56.07%, compared with 39.50% for SR-Scientist + RL, a gain of 16.57 percentage points. At 1%, OOD accuracy is 84.00% in materials science, 48.15% in chemistry, 48.61% in biology, and 50.76% in physics. Physics retains 40.91% accuracy at the stricter 0.1% tolerance. These results show that the advantage of RISR extends beyond ID fitting to prediction outside the training domain using the same discovered equations.

4.3 Ablation Study

Table 2 isolates the contributions of coefficient fitting and the learned correction-selection policy. The complete residual-informed search raises accuracy at the 1% tolerance from 7.75% to 36.43% on ID data and from 9.30% to 40.31% on OOD data compared with the LLM-only method. Residual-guided refinement provides additional gains, reaching 63.57% ID and 55.04% OOD accuracy at the 1% tolerance. Compared with random corrections, learned-scored corrections improve ID and OOD accuracy at the 1% tolerance by 3.10 and 2.33 percentage points. The results show a positive contribution from residual-informed formula search and learned action selection under the fixed correction budget.

4.4 Symbolic Accuracy

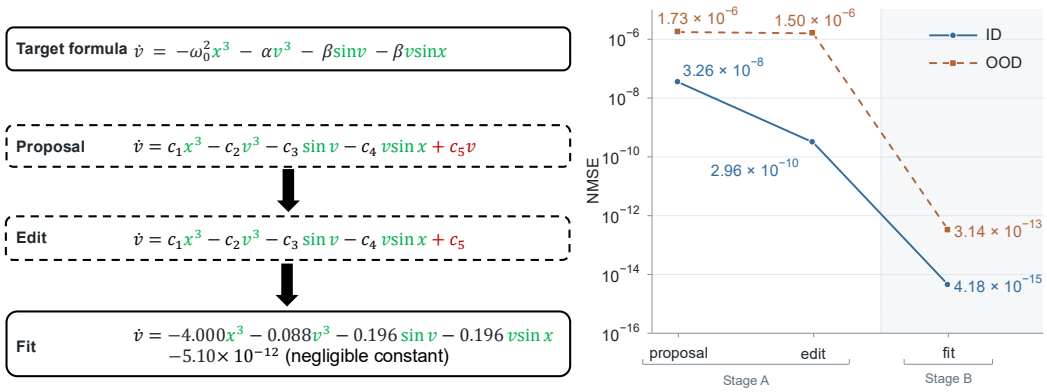


Figure 3: Case study on PO30. RISR retains the target components, removes an extraneous term through structural editing, and calibrates the coefficients during refinement.

Symbolic accuracy (SA) evaluates whether a discovered equation can represent the target relationship. We follow the symbolic-assessment criterion of SR-Scientist (Xia et al., 2026): A candidate is counted as positive when an assignment of its adjustable constants can represent the target expression. GPT-6-Astra serves as our judge, and non-unanimous cases are reserved for human review. 17 problems receive ten Yes votes, and 112 receive ten No votes, leaving no voting disagreements. We additionally manually reviewed all 17 cases classified as positive by the LLM judge. RISR obtains an SA of 13.18% $(17/129)$, exceeding existing methods (Table 3). SA and numerical accuracy are complementary: predictive agreement on held-out observations does not necessarily imply structural recovery under the constant-assignment criterion. Appendix E documents more details.

Table 3: SA (%) on one seed setting.

Method	LLM backbone	SA ↑
uDSR	—	0.77
PySR	—	4.65
LLM-SR	GLM-4.5-Air	5.43
LLM-SR	GPT-OSS-120B	4.65
LLM-SR	Qwen3-Coder-480B	5.43
SR-Scientist	GLM-4.5-Air	7.75
SR-Scientist	GPT-OSS-120B	7.00
SR-Scientist	Qwen3-Coder-480B	7.00
RISR	Qwen3-Coder-30B	**13.18**

4.5 Case Study

PO30 illustrates how RISR contributes to equation recovery (Figure 3). The target describes a nonlinear oscillator, with position x and velocity v. During Stage A, the formula proposal contains all four target components, but also includes an extra term $c_5 * v$. Then the learned structural-edit generator replaces this extra term with a constant and submits the edited formula for coefficient fitting and scoring. At Stage B, coefficient fitting calibrates the selected equation without changing its structure. The remaining constant becomes negligible, and the two independently fitted sinusoidal coefficients closely match the shared coefficient in the target.

We examine more cases (Appendix F), including those in which numerical error is substantially reduced despite incomplete recovery of the underlying formula. These cases may inform future research aimed not only at reducing numerical error but also at recovering the scientific formulas.

5 Conclusion

We present RISR, a two-stage method that uses the pointwise residual information of a current equation to guide scientific equation discovery. Residual-conditioned LLM proposals and structural-edit generators support formula search, while a dual-view encoder predicts which corrections are likely to improve the equation after fitting. For the evaluation, RISR reaches 63.57% and 38.50% ID accuracy at the 1% and 0.1% tolerances. The case study illustrates how structural editing and coefficient calibration contribute at different points in the discovery process. The evaluation results show the effectiveness of our method in the task of scientific equation discovery.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the writing of proofs, implement methods, assist with translation, and formulate mathematical claims. We have not used generative AI tools to generate synthetic data sets, propose or refine hypotheses, design or provide feedback on research methodology or experiments, clean and reformat datasets, support qualitative and thematic data analysis, or interpret results. Providing critical ingredients for proving mathematical claims and helping develop theoretical models or conceptual frameworks is not applicable to this work. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

RISR is developed and evaluated using publicly available scientific benchmark data and synthetic numerical datasets generated through transformations and compositions of elementary mathematical functions. This work focuses on methodological research in scientific equation discovery. We have not identified specific ethical concerns associated with the data and experiments presented in this study.

REPRODUCIBILITY STATEMENT

The main paper and appendices describe the model architectures, synthetic data generation procedures, training settings, and evaluation protocols used in our experiments. To facilitate reproducibility, we will publicly release the synthetic datasets and the code for data generation, model training, equation search and refinement, and evaluation.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

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

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113. URL https://www.pnas.org/doi/abs/10.1073/pnas.1517384113.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pp. 89–96, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363.

Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl. *arXiv preprint arXiv:2305.01582*, 2023. doi: 10.48550/arXiv.2305.01582.

G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on Numerical Analysis*, 10(2):413–432, 1973. doi: 10.1137/0710036. URL https://doi.org/10.1137/0710036.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. A unified framework for deep symbolic regression. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 33985–33998. Curran Associates, Inc., 2022. doi: 10.52202/068431-2463. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/dbca58f35bddc6e4003b2dd80e42f838-Paper-Conference.pdf.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Dianne P. O'Leary and Bert W. Rust. Variable projection for nonlinear least squares problems. *Comput. Optim. Appl.*, 54(3):579–593, April 2013. ISSN 0926-6003. doi: 10.1007/s10589-012-9492-9. URL https://doi.org/10.1007/s10589-012-9492-9.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Cláudio Prata Santiago, Sookyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=m5Qsh0kBQG.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: https://doi.org/10.1038/s41586-023-06924-6.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893. URL https://www.science.org/doi/abs/10.1126/science.1165893.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. Transformer-based planning for symbolic regression. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 45907–45919. Curran Associates, Inc., 2023. doi: 10.52202/075280-1990. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8ffb4e3118280a66b192b6f06e0e2596-Paper-Conference.pdf.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: scientific equation discovery via programming with large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=m2nmp8P5in.

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. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 55325–55359. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/shojaee25a.html.

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.

Shijie Xia, Yuhan Sun, and Pengfei Liu. Sr-scientist: Scientific equation discovery with agentic ai. In *International Conference on Learning Representations*, volume 2026, pp. 75787–75811, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 3394–3404, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

A Details of Residual-Informed Formula Search

This section describes the configuration used for RISR. Stage A combines residual-conditioned LLM proposals and learned structural edits.

A.1 Formula search and numerical inputs

The search extends the evaluator–buffer–sampler design of LLM-SR (Shojaee et al., 2025a). Executable candidates are fitted to the training observations and stored with their training MSE, predictions, and residuals. The buffer maintains ten islands and samples two equation programs from one island to form each prompt. The highest-scoring equation included in that prompt supplies the parent predictions for numerical conditioning. The textual input contains an equation-completion instruction, variable descriptions, and the sampled code context. Pointwise observations are supplied through continuous embeddings rather than a natural-language account of the residuals.

The residual encoder receives aligned coordinates x_i, targets y_i, parent predictions p_i, and residuals $r_i = y_i - p_i$. It has six coordinate slots. For a problem with more than six variables, each coordinate is scored by its largest absolute correlation with the residual, considering raw, absolute-value, and squared transformations on both sides. The six highest-scoring coordinates are retained in their original order, with coordinate-index tie breaking. Routing affects only this encoder: the equation signature and numerical fitting retain all original variables. The search configuration allows up to 4,000 encoding rows. If subsampling is required, rows are sorted by residual and one aligned row is sampled from each equal-count stratum.

Normalization statistics are computed on the retained rows. Coordinate j is centered by its median and divided by the larger of its interquartile range and standard deviation. A scale at most 10^{-8} is replaced by one. Coordinates are clipped to $[-12, 12]$ and padded to six slots. Let μ_y be the target mean and $\sigma_y = \max\{\mathrm{std}(y), 10^{-8}\}$. Targets and predictions are centered by μ_y and divided by σ_y. Residuals are divided by σ_y without centering. These three channels are clipped to $[-20, 20]$. The normalized residual's magnitude and signed $\log(1 + |r|)$ complete five observation channels. Each row therefore has eleven entries, with a separate mask for active coordinate slots.

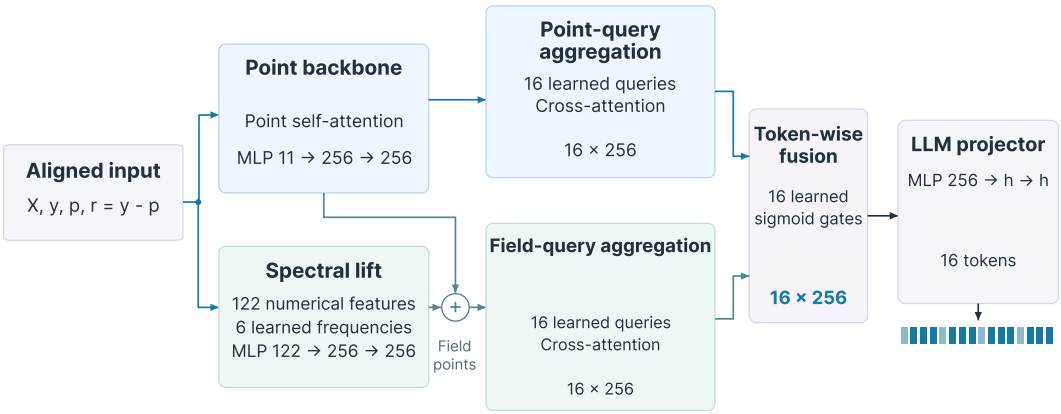


Figure 4: Residual encoder architecture.

A.2 Point-attention and spectral-field paths

Figure 4 shows how the residual encoder maps aligned numerical observations to continuous LLM tokens through point-attention and spectral-field paths.

Point-attention path. A two-layer $11 \to 256 \to 256$ MLP embeds each row. Two eight-head self-attention blocks mix point features, followed by layer normalization. For large observation sets, attention operates within chunks after sorting by normalized residual plus a small deterministic projection of the full row. Search-time chunks contain at most 256 points. Sixteen learned queries aggregate the valid points through two cross-attention blocks. Their shared context combines an MLP of feature means, standard deviations, and the variable mask with an MLP of the mask alone. A final layer normalization yields sixteen vectors of width 256.

Spectral-field path. The second path adds an explicit feature lift to the point representations. It contains 24 coordinate channels (each coordinate alone and multiplied by the normalized residual, its magnitude, or its square), 21 coordinate-pair products multiplied by the residual, 72 sine/cosine channels, and the five observation channels. Thus, each point supplies $24 + 21 + 72 + 5 = 122$ features. The coordinate-pair channels include squares. Six positive frequencies, shared across coordinates, are learned in log space and initialized on a logarithmic grid from one to eight. Their sinusoidal arguments include a factor of π. Features associated with padded coordinates are masked. An MLP maps the lift to width 256 and adds it to the point features. These are learned frequency features, not an FFT of the observations.

Seven learned residual centers, initialized uniformly from -3 to 3, define additional summaries. For each center, Gaussian-shaped weights depend on normalized residual distance and a shared learned positive precision. Weights sum to one over valid points. A weighted field-feature average and a learned region embedding form each summary. Sixteen field queries attend to both point features and the seven summaries. Two cross-attention/latent-self-attention pairs, followed by layer normalization, produce sixteen width-256 vectors.

Fusion and projection. One learned sigmoid gate per token interpolates between the two paths. The gates are shared across examples. A two-layer GELU projector with widths $256 \to 2048 \to 2048$, layer normalization, and a learned positive output scale maps the fused vectors to the Qwen embedding space. The scale starts at 0.02. One active marker embedding is replaced by the sixteen continuous vectors, and the sequence mask is updated. Inserted positions do not receive target-formula token labels. The LLM is frozen during encoder training, and both networks operate in evaluation mode during equation search.

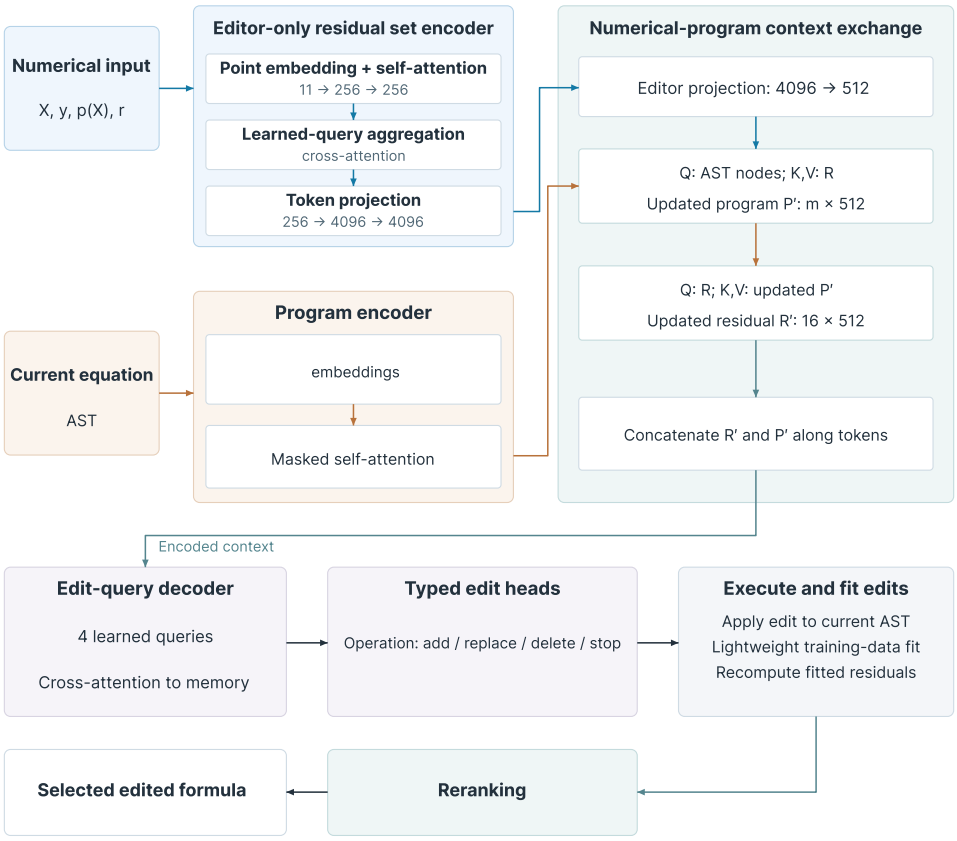


Figure 5: Numerical and program encoders within the structural edit generator.

A.3 Auxiliary learned structural-edit generator

Figure 5 shows the structural editor's numerical and program encoders. The editor takes the training observations and the current formula. Its own residual encoder summarizes coordinates, targets, predictions, and residuals into sixteen latent vectors, projected to the editor's hidden width. A program encoder represents the formula's abstract syntax tree, including node types, variables, constants, parent relations, and depths. Cross-attention combines the numerical and program representations to predict an edit operation, target node, function family, variable support, and continuous parameters. The available operations are addition, replacement, deletion, and stopping. The input program is limited to 96 nodes and numerical encoding uses at most 512 observations.

Predicted edits are converted into executable formulas and evaluated with lightweight training-data fitting. Their resulting residuals inform learned outcome scores used for reranking. A selected edit is then submitted to the common outer evaluator, which fits its coefficients and scores its training MSE. The editor's predicted extrapolation-related score is learned from synthetic data. It does not evaluate the benchmark OOD split during search. All editor weights are frozen at deployment. Its pretraining and edit supervision are described in Appendix C.2.

A.4 Stage-A LLM Calling

LLM decoding uses temperature 0.8, top-$k = 30$, top-$p = 0.9$, repetition penalty 1.0, and at most 256 new tokens per completion. The buffer cluster temperature is 0.1. Outer coefficient fitting uses BFGS with one configured restart and a 120-second evaluation timeout. Among executable logged candidates that reproduce their recorded training fit, the equation with the lowest training MSE is selected, retaining the earlier candidate on ties. This selected equation is the Stage-A output p_0.

B Details of Residual-Guided Refinement

B.1 Dual-view construction and relational statistics

Figure 6 details the dual-view refinement encoder and its relation-attention blocks. The encoder maps additive and multiplicative residual statistics to variable representations used by the correction utility scorer.

For predictions p_i and residuals $r_i = y_i - p_i$, the regularized multiplicative coordinate solves

$$u_i = \arg\min_{u\in\mathbb{R}}\{(r_i - p_i u)^2 + \lambda^2 u^2\} = \frac{p_i r_i}{p_i^2 + \lambda^2}, \qquad \lambda = 0.05\max\{\mathrm{RMS}(p), 10^{-8}\}. \tag{8}$$

Prediction RMS is computed on the encoding rows. The positive floor makes u_i defined even when all predictions vanish. The additive view uses r_i/σ_y; the multiplicative view uses $u_i/\max\{\mathrm{std}(u), 10^{-8}\}$, without centering the signal. Both signals are clipped to $[-20, 20]$.

Coordinates use median/IQR-or-standard-deviation normalization and clipping as in the residual encoder, but are padded to eight slots. Each view also contains target and prediction channels normalized by the target mean and standard deviation, its correction signal, the signal magnitude, and its signed logarithm. Squaring and cubing the signal yields seven observation features. For a normalized coordinate z, the fourteen coordinate transforms are

$$\begin{aligned}\Psi(z) = \big[&z, z^2, z^3, |z|, \log(1+z^2), e^{-z^2},\\ &\{\sin(\pi f z), \cos(\pi f z)\}_{f\in\{0.5,1,2,4\}}\big].\end{aligned} \tag{9}$$

Cross-correlations of the fourteen transforms with seven signals, plus their means and log standard deviations, provide $14\times 7 + 2\times 14 + 2\times 7 = 140$ features per variable. Moment standardization floors the variance at 10^{-8}.

For each directed variable pair (j, k), correlations of $z_j z_k$, $z_j^2 z_k$, and $z_j z_k^2$ with the correction signal, its magnitude, and its square give nine edge features. Separately, each unordered pair has 64 operator-product correlations in each view. These use the eight correction operators evaluated on normalized coordinates, standardize each operator response, and correlate their products with the view signal. Operator-response and signal standard deviations are floored at 10^{-4}; product variances are floored at 10^{-8}, and correlations are clipped to $[-1, 1]$. Exponentials in this statistical probe use arguments clipped to $[-5, 5]$. This probe clipping does not redefine the exponential operator in fitted correction formulas.

A shared $140 \to 128 \to 128$ embedding and two four-head relation-attention blocks encode each view. Linear maps of the nine edge features contribute attention biases and values. Each block has residual connections and a width-256 feed-forward hidden layer. Masked mean pooling of the two views supplies a global width-128 context. Concatenating this context with each variable's two representations gives a width-384 vector, which a $384\to128$ fusion layer maps to $\mathbf{h}_j$.

B.2 Action vocabulary and shared utility head

Each action specifies an additive or multiplicative update, one variable or two distinct variables, and one operator per variable. The operator order is

$$\Phi = \{z, z^2, z^3, \sin z, \tanh z, \exp z, \log(1+z^2), (1+z^2)^{-1}\}. \tag{10}$$

There are $16d + 128\binom{d}{2}$ valid actions at dimension d, or 3,712 for eight variables. Product factors are stored in canonical variable order, and actions involving padded variables are masked. A shared network maps the concatenation of $\mathbf{h}_j$ and a 16-dimensional operator embedding to a 64-dimensional query $\mathbf{q}_{j,k}$. The action representation is

$$\begin{aligned}\mathbf{v}_a = \big[&\mathbf{q}_L + \mathbf{q}_R,\ \mathbf{q}_L \odot \mathbf{q}_R,\ |\mathbf{q}_L - \mathbf{q}_R|,\\ &\mathbf{e}_m,\ I_{\mathrm{product}},\ c_a^{(+)},\ c_a^{(\times)}\big],\end{aligned} \tag{11}$$

where $\mathbf{e}_m$ is a 16-dimensional mode embedding and the last two scalars are the action's product correlations. Unary actions set $\mathbf{q}_R = 0$, $I_{\mathrm{product}} = 0$, and both correlations to zero. A shared $211\to128\to1$ head predicts fitted utility. The refinement encoder, queries, and utility head together contain 389,297 trainable parameters.

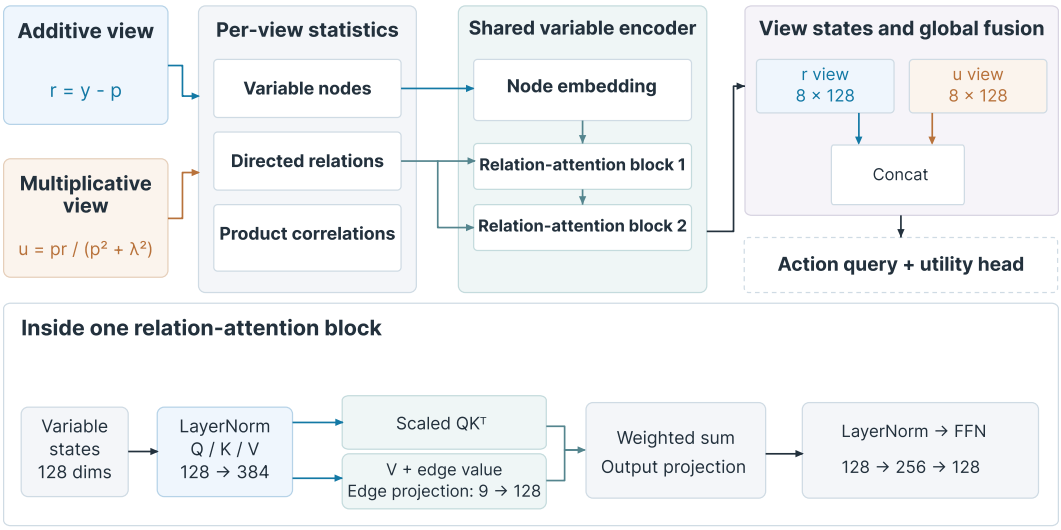


Figure 6: Dual-view refinement encoder. The two views share the embedding network and relation-attention weights. The lower panel shows how pair features modify attention logits and values.

B.3 Correction coefficient fitting

For an observation subset S, the fitting loss is

$$L_S(f) = \frac{1}{|S|} \sum_{i \in S} \left(\frac{f(x_i) - y_i}{s_{i,S}} \right)^2, \qquad s_{i,S} = \max\{|y_i|, 0.01\,\mathrm{RMS}(y_S), \epsilon_{64}\}, \tag{12}$$

where ϵ_{64} is double-precision machine epsilon. This weighted squared-error objective is different from the test-time accuracy-to-tolerance criterion.

At the start of Stage B, all adjustable coefficients of p_0 are fitted on the full training set with its structure fixed. The trust-region reflective least-squares solver starts from the saved coefficients, uses a two-point numerical Jacobian, at most 500 function evaluations, and 10^{-10} for each termination tolerance. Parameter scales are $\max\{|c_j|, 1\}$. A finite strict improvement in L_S is retained; otherwise the original coefficients are kept. This initial calibration is separate from the four-attempt correction budget and does not use the 0.1% minimum-improvement threshold required for accepting a new correction.

An action defines $h_a = \alpha g_a + \beta$, where $g_a(x) = \prod_{j \in J} \phi_{k_j}(b_j z_j + c_j)$ and $z_j = (x_j - \mu_j)/\max\{\sigma_j, 10^{-8}\}$ uses fitting-row means and standard deviations. For fixed nonlinear parameters, the two linear design columns are $(g_a, 1)$ for an additive update and (pg_a, p) for a multiplicative update, with response $y - p$. Dividing the design and response by $s_{i,S}$ gives the weighted least-squares problem for α, β. The current equation's existing coefficients remain fixed during every new correction fit.

The nonlinear parameters use three deterministic starts. Unary starts for (b, c) are $(1, 0)$, $(-1, 0)$, and $(2, 0.5)$; product starts for (b_1, c_1, b_2, c_2) are $(1, 0, 1, 0)$, $(-1, 0, 1, 0)$, and $(2, 0.5, 0.5, -0.5)$. Slopes are bounded by $[-20, 20]$ and shifts by $[-10, 10]$. Each start uses trust-region reflective least squares, a two-point Jacobian, and tolerances of 10^{-9}. Each action has a 15-second overall deadline across its starts. The best finite fit completed within the budget is retained. An attempt with no finite candidate leaves the equation unchanged; solver convergence alone is not the acceptance criterion.

B.4 Encoding rows, action selection, and stopping

The current refinement configuration encodes 256 training rows. When more rows are available, a deterministic sample without replacement is drawn using seed 20260831 and its indices are sorted. The same indices are used after accepted updates; their predictions, residuals, and normalization statistics are recomputed. This fixed encoding-row seed is distinct from the Stage-A search seed.

Every coefficient fit and acceptance decision uses the full training set. During synthetic utility training, encoding and fitting instead use the designated 192 fitting rows.

At each attempt, the frozen model ranks dimension-valid actions and selects the highest-scoring untried action, using a stable order for ties. For the fitted equation $\widetilde{p}_a$, acceptance requires finite training predictions and $L_S(\widetilde{p}_a) < 0.999 L_S(p)$. An accepted action updates p and its residual state; a rejection retains p. Every attempted action is marked as tried, and all attempts count toward the cap of four. Refinement stops before a new attempt when training NMSE is at most 10^{-12}. Thus, a task may require zero new correction fits even though its initial coefficients were refitted. No Stage-B operation calls the LLM or uses test observations for acceptance.

C Synthetic Data and Offline Training

The two main training corpora supervise residual-to-formula generation and fitted correction utility. The auxiliary structural editor has additional residual-pretraining and edit-supervision datasets. All are generated from elementary functions, sampled transformations, coefficients, and coordinates. They do not use benchmark equations or observation arrays as generation inputs. Table 4 separates their sizes and learning roles.

Table 4: Synthetic training collections. Counts are examples except for the paired utility corpus, whose counts are pairs. The three utility test sets are additional to its training and validation splits.

Collection	Train	Validation	Supervision
Search residual-to-formula	6,000	768	Target formula tokens
Editor residual pretraining	50,000	5,000	Families, variables, gains
Structural edits	20,000	2,000	Edits and fitted outcomes
Paired correction utility	8,192	1,024	Eight actions per state

C.1 Search encoder data and training

Three families contribute 2,000 training and 256 validation examples each. Compositional examples multiply two to six transformed factors with a global amplitude and at most ten target-program parameters; the current equation omits a factor or uses an affine approximation. Smooth-regime examples combine coordinate and condition responses through sigmoid or $\tanh$ gates; the current equation omits the late regime or uses an affine approximation. Factorized responses combine onset, hardening or softening, condition-dependent factors, and a smooth damage gate; current equations omit a condition factor or damage gate, or use an affine approximation.

Each example has 256 observations and one to six variables. Compositional coordinates are sampled uniformly from randomly chosen intervals among $[-3, 3]$, $[0.05, 5]$, $[0, 1]$, and $[0, 40]$. The other families use two randomly assigned active coordinates in $[0, 1]$, with additional coordinates in $[-1, 1]$. Elementary transformations include shifts, rescaling, powers, saturation, exponential decay or onset, squared sine, and hinges. Generation rejects nonfinite targets, target standard deviation at most 10^{-4}, and absolute outputs exceeding 10^5 for compositional examples or 10^4 for the other families. Validation checks row alignment, finite inputs, and $r = y - p$. All 6,768 target programs pass parsing and compilation; this is a syntactic validation and is not a proof of identifiability from observations.

The residual encoder and projector are randomly initialized and trained through target-formula token cross-entropy, with all LLM weights frozen and no LoRA adapter. Training uses AdamW, one epoch over 6,000 examples, an effective batch size of eight, and 750 updates. Learning rates are 2×10^{-5} for the encoder trunk and 10^{-4} for the projector, with weight decay 0.01.

C.2 Structural-editor data and training

The editor's residual encoder is separately pretrained on 50,000 training and 5,000 validation examples. A sampled sum of elementary terms forms a current equation; additional sampled terms form the clean target. The library includes constants, powers, trigonometric and saturating functions,

bounded exponentials, protected inverses and ratios, and variable products. Labels identify function families, active variables, and numerical fitting gains. This encoder is frozen during subsequent edit training.

Edit training uses 20,000 training and 2,000 validation examples, each with 256 observations and one to six variables. Current programs omit a target component, contain an extra or incorrect component, or already match the target. These constructions provide addition, deletion, replacement, and stop labels. Up to four valid candidate slots provide measured fitting gains, gains on separately sampled wider-range synthetic inputs, and changes in expression complexity. Gaussian observation noise has standard deviation equal to 0.5% of the clean-target standard deviation, with a positive floor. Structural-record hashes separate training and validation.

The program encoder and editing modules learn supervised operation, location, family, and variable predictions, continuous parameters, outcome regressions, and candidate rankings. AdamW training uses ten epochs, batch size sixteen, learning rate 2×10^{-4}, and weight decay 0.01. Selection uses validation loss. The synthetic wider-range observations provide offline supervision; they are not the benchmark OOD arrays.

C.3 Paired-state refinement data

Targets contain three to six unary or two-variable product components using the eight correction operators. Affine input slopes are sampled in $[0.6, 2.2]$ and shifts in $[-1, 1]$. Components are combined by addition, multiplication by one plus a component, saturation through $\tanh$, or division by one plus a component's square. An affine output transformation varies scale from 10^{-3} to 10^{3}. Each problem has 256 points drawn from uniform, clipped-normal, or standardized-lognormal distributions, followed by coordinate rescaling and translation. Training and validation use two to six variables; the dimensional test uses eight.

Two current equations share each observation cloud and noisy target. Coarse states are formed by removing an additive component, dividing out a multiplicative component, fitting a subset of components, or combining these operations. One quarter of pairs remain coarse. For the others, each prediction is rescaled toward the clean target,

$$p^{(s)} = f_{\text{clean}} + \alpha_s(p^{(s)}_{\text{coarse}} - f_{\text{clean}}), \qquad s \in \{0, 1\}, \tag{13}$$

placing its clean-error contribution, measured on the first 192 rows, into one of eight logarithmic bins spanning 10^{-8} to 1. Observation noise is unchanged. The encoder receives the numerical state, not the target expression.

Each pair shares eight dimension-valid actions. Frozen earlier residual-based policies propose top actions for both current states, and a frozen target-only policy supplies additional candidates. After deduplication, random valid actions fill the set to eight, retaining at least two random candidates.

Table 5: Paired-state utility splits. Each pair has two states and eight actions per state. Only the training split contributes labels to gradient updates.

Split	Pairs	States	Utility labels
Training	8,192	16,384	131,072
Validation	1,024	2,048	16,384
New observation clouds	1,024	2,048	16,384
Held-out product actions	1,024	2,048	16,384
Eight-variable inputs	1,024	2,048	16,384
Total	12,288	24,576	196,608

Splits use separate random streams and keep both states of a pair together. In the held-out-product test, operator-index pairs with $(o_1 + o_2) \bmod 4 = 0$ are reserved for candidate actions and excluded from candidates in the other splits. Target-expression construction can still use all operator combinations. This tests held-out correction actions rather than disjoint target-function families. Validation checks shared-action alignment, variable eligibility, acceptance decisions, and utility arithmetic. Positive, zero, and negative labels are retained.

C.4 UTILITY TARGETS, LOSSES, AND OPTIMIZATION

The first 192 rows form F and supply encoder inputs, coefficient fits, and acceptance decisions. The remaining 64 rows form H and measure fitted utility. Some target-generation normalizations use the full synthetic cloud before this assignment; H is therefore held out from correction fitting, not from every operation used to construct the synthetic problem. Neither subset contains benchmark test observations. After fitting on F, define

$$p'_a = \begin{cases} \widetilde{p}_a, & L_F(\widetilde{p}_a) < 0.999 L_F(p) \text{ and the fit is finite,} \\ p, & \text{otherwise.} \end{cases} \tag{14}$$

The supervision target is

$$U(p,a) = \mathrm{clip}_{[-3,3]}\left[\log_{10} \frac{L_H(p) + 10^{-8}}{L_H(p'_a) + 10^{-8}}\right]. \tag{15}$$

The same loss in Eq. 12 is evaluated separately on F and H. Rejection gives zero utility; nonfinite utility-row predictions receive -3. An accepted action may have negative utility if it worsens prediction on H.

Let $\widehat{U}_{s,a}$ be the predicted utility and ℓ the smooth-L_1 loss with transition at absolute error one. The value term averages $\ell(\widehat{U}_{s,a} - U_{s,a})$. For actions a, b in the same state, define $d_{s,ab} = U_{s,a} - U_{s,b}$, $\widehat{d}_{s,ab} = \widehat{U}_{s,a} - \widehat{U}_{s,b}$, and $w_{s,ab} = \min\{|d_{s,ab}|, 3\}\mathbf{1}[|d_{s,ab}| > 0.05]$. The ranking loss is

$$\mathcal{L}_{\mathrm{rank}} = \frac{\sum_{s,a,b} w_{s,ab} \log(1 + \exp[-\operatorname{sign}(d_{s,ab})\widehat{d}_{s,ab}])}{\max\{1, \sum_{s,a,b} w_{s,ab}\}}. \tag{16}$$

For each paired current state and shared action, the paired term averages

$$\ell\big((\widehat{U}_{0,a} - \widehat{U}_{1,a}) - (U_{0,a} - U_{1,a})\big). \tag{17}$$

The total is $\mathcal{L}_{\mathrm{value}} + 0.5\mathcal{L}_{\mathrm{rank}} + 0.5\mathcal{L}_{\Delta}$.

The refinement encoder, action-query network, and utility head are trained jointly from random initialization. AdamW uses learning rate 3×10^{-4}, weight decay 0.01, and gradient-norm clipping at one. A batch contains 64 pairs, or 128 states. Sixteen epochs over 8,192 pairs give 2,048 updates. The selected model remains frozen during refinement.

D RESULT STATISTICS

The main table and ID/OOD plot average three complete results with seeds 0, 42, and 20260831 (Table 6). We first compute the success rate for each seed, then take the mean of the three rates. Standard deviations are sample standard deviations with two degrees of freedom. Domain results use the corresponding domain's task count.

Table 6: Detailed results for the three seeds.

Result set	ID, 1%	ID, 0.1%	OOD, 1%	OOD, 0.1%
Seed 0	65.89 (85)	42.64 (55)	57.36 (74)	41.09 (53)
Seed 42	61.24 (79)	35.66 (46)	55.81 (72)	34.88 (45)
20260831	63.57 (82)	37.21 (48)	55.04 (71)	38.76 (50)
Mean ± SD	63.57 ± 2.33	38.50 ± 3.66	56.07 ± 1.18	38.24 ± 3.13

E SYMBOLIC ACCURACY: PROTOCOL AND DETAILED RESULTS

SA uses the constant-assignment criterion of SR-Scientist (Xia et al., 2026): whether some assignment of the candidate's adjustable constants makes it represent the target relationship. The original

program parameters and the parameters of accepted corrections remain adjustable for this assessment. Coordinate normalization, operator choices, powers, clipping guards, and fixed literals remain unchanged.

SA uses all 129 final programs and includes every accepted correction. With their saved coefficients, the exported programs reproduce stored predictions in all 387 training/ID/OOD replay checks.

GPT-6-Astra evaluates each candidate–target pair in ten separate requests. The recorded requests specify high reasoning effort and a limit of 8,192 completion tokens, without an explicit temperature or sampling seed. The prompt asks about the existence of suitable constant assignments and requests a structured Yes/No answer. Unanimous Yes and No outcomes are accepted; non-unanimous outcomes are reserved for human review. The completed assessment has 1,290 valid judgments: 17 tasks receive ten Yes votes and 112 receive ten No votes, with no unresolved cases (Table 7).

Table 7: Symbolic accuracy for the seed 20260831. All positive tasks receive ten Yes votes; all remaining tasks receive ten No votes.

Domain	Positive	Tasks	SA (%)
Materials	1	25	4.00
Chemistry	5	36	13.89
Biology	3	24	12.50
Physics	8	44	18.18
Overall	17	129	13.18

F More Case Studies

The examples below report saved formulas, selected actions, acceptance decisions, and replayed errors. The residual encoder supplies continuous numerical tokens, and the Stage-B model outputs utility scores rather than textual reasoning.

F.1 CRK22: a four-step numerical refinement

The chemistry reference has the structure

$$\dot{A} = -kA^2 - kA\exp(-k_s t) + \frac{k_z A^2}{1 + qA^4}, \tag{18}$$

where t is time and A is concentration.

Table 8: CRK22 action sequence. All four actions are additive and accepted by the training objective. Define $\mathrm{rq}(z) = (1 + z^2)^{-1}$ and $\mathrm{lsq}(z) = \log(1 + z^2)$.

Attempt	Mode	Correction family	OOD NMSE
0	—	Fitted parent	2.0917
1	Additive	$\mathrm{rq}(t)\ \tanh(A)$	4.0358×10^{-4}
2	Additive	$\mathrm{rq}(t)\ \mathrm{lsq}(A)$	2.9398×10^{-4}
3	Additive	$\mathrm{rq}(t)\ \sin(A)$	1.7143×10^{-4}
4	Additive	$\sin(t)\ \mathrm{rq}(A)$	6.4102×10^{-6}

ID NMSE changes from 1.6757×10^{-7} to 4.7072×10^{-8}. The OOD point fraction below each of the two main tolerances changes from zero to one. Each accepted correction changes the residual state used for the next ranking. The final formula is nevertheless not counted as symbolically equivalent by the ten-vote assessment.

F.2 MatSci9: additive and multiplicative corrections

MatSci9 (Table 9) alternates between multiplicative and additive actions on strain ϵ and temperature T. A multiplicative action means $p \leftarrow p(1 + h_a)$, not multiplication by the displayed operator product alone. Every listed factor again has fitted affine parameters, and all four attempts are accepted.

Table 9: MatSci9 refinement trajectory. Action names omit fitted affine parameters, amplitudes, and offsets.

Attempt	Mode	Correction family	OOD NMSE
0	—	Fitted parent	5.478×10^{-7}
1	Multiplicative	$\mathrm{cube}(\epsilon)\,\mathrm{square}(T)$	1.471×10^{-8}
2	Multiplicative	$\tanh(\epsilon)\,\mathrm{linear}(T)$	3.322×10^{-9}
3	Additive	$\mathrm{lsq}(\epsilon)\,\mathrm{lsq}(T)$	2.878×10^{-9}
4	Multiplicative	$\mathrm{cube}(T)$	7.566×10^{-9}

ID NMSE decreases from 1.490×10^{-9} to 2.580×10^{-12}. Final OOD NMSE is lower than the parent's, but higher than at the third prefix. The final OOD point fraction is one at both main tolerances. This nonmonotonic test error is consistent with training-only acceptance and is retained rather than replaced by the best test-scoring prefix.

G Utility-Ranking Experiments

Each synthetic example contains two current equations sharing the same observation cloud and the same eight candidate actions. Let s denote one state and $\bar{s}$ its paired state. For the paired-state swap diagnostic, we use the scorer outputs produced for $\bar{s}$ to select an action for s:

$$a_s^{\mathrm{swap}} = \arg\max_a \widehat{U}_{\bar{s},a}, \qquad U_s(a_s^{\mathrm{swap}}) \text{ is then evaluated using the utility labels of } s.$$

The same operation is performed in the opposite direction for $\bar{s}$. Thus, the action pool and fitting protocol are unchanged; only the predicted scores are taken from the other current equation.

To directly test whether the learned scorer identifies promising corrections, we evaluate it on three held-out synthetic splits with a fixed pool of eight dimension-valid actions per state. Each split contains 1,024 paired problems, or 2,048 current-equation states. Every state is associated with a fixed pool of eight dimension-valid correction actions.

The refinement model is frozen during this evaluation. For every state and candidate action, the correction is fitted on the designated fitting rows using the same variable-projection procedure as in the refinement stage. The fitted equation is accepted only when it reduces the fitting loss by more than 0.1%. Its utility is then measured on the disjoint utility rows using the clipped logarithmic error reduction defined in Eq. 5. The reported score is the mean utility of the action selected by each selector.

The results are shown in Table 10. The learned scorer achieves mean utilities of 0.4627, 0.4099, and 0.4532 on new observation clouds, unseen product actions, and eight-variable inputs, respectively. The learned scorer outperforms residual-correlation ranking and uniform random selection on all three held-out splits. The paired-state swap is substantially worse, which is consistent with the scorer using information about the current equation rather than only the shared action pool. This result provides direct evidence that the learned utility scorer can identify more promising corrections than simple selection rules in controlled candidate pools.

H Computational Accounting

Search budget. Stage A permits at most 64 outer candidate submissions, including the initial seed equation. Invalid submissions also consume an outer slot. After the outer search terminates, executable candidates that reproduce their recorded training fit are compared using the training objective, and the candidate with the lowest training MSE is selected as p_0. Stage B first fits the

Table 10: Held-out utility of the selected action in fixed eight-action candidate pools. Each column contains 2,048 states. Utilities are clipped base-10 log error reductions measured after coefficient fitting. Higher values are better.

Selector	New observation clouds	Unseen product actions	Eight-variable inputs
Learned utility scorer	0.46273	0.40994	0.45324
Residual-correlation ranking	0.39336	0.37877	0.39710
Uniform random selection	0.24101	0.22907	0.19557
Paired-state swap	0.27099	0.25677	0.22226

coefficients of p_0 under the relative-error objective. This calibration is separate from the subsequent budget of at most four new correction-fitting attempts.

Measured cost. The Stage-A configuration generates four continuations per LLM request. A complete 64-slot task search therefore uses at most 64 LLM generation requests.

For the seed 20260831, in Stage B, the refinement logs contain 424 new correction attempts, of which 400 are accepted. The projected fitting objective is evaluated 461,860 times, and the recorded correction-fit durations sum to 433.98 seconds.