SR-Scientist: Scientific Equation Discovery With Agentic AI

Shijie Xia [*1 2 3] **Yuhan Sun** [*1 3] **Pengfei Liu** [1 2 3]

Abstract

Recently, Large Language Models (LLMs) have been applied to scientific equation discovery, leveraging their embedded scientific knowledge for hypothesis generation. However, current methods typically confine LLMs to the role of an equation proposer within search algorithms like genetic programming. In this paper, we present SR-SCIENTIST, a framework that elevates the LLM from a simple equation proposer to an autonomous AI scientist that writes code to analyze data, implements the equation as code, submits it for evaluation, and optimizes the equation based on experimental feedback. Specifically, we wrap the code interpreter into a set of tools for data analysis and equation evaluation. The agent is instructed to optimize the equation by utilizing these tools over a long horizon with minimal human-defined pipelines. Empirical results show that SR-SCIENTIST outperforms baseline methods by an absolute margin of 6% to 35% on datasets covering four science disciplines. Additionally, we demonstrate our method's robustness to noise, the generalization of the discovered equations to out-of-domain data, and their symbolic accuracy. Furthermore, we develop an end-to-end reinforcement learning framework to enhance the agent's capabilities.

1. Introduction

In the era of Agentic AI, Large Language Models (LLMs) have evolved from simple knowledge retrievers to agentic models capable of completing complex tasks by interacting with their environments (Schneider, 2025), as exemplified by products like Claude Code (Anthropic, 2025) and Gemini CLI (Google, 2025). These agents exhibit many characteristics of human scientists, such as engaging in long-horizon interaction with environmental feedback, operating with autonomy, and relying less on predefined pipelines (Newell et al., 1972). However, in most current work using LLMs for scientific discovery (Shojaee et al., 2025a; Novikov et al., 2025; Romera-Paredes et al., 2024), LLMs serve as static components within human-crafted pipelines, lacking the autonomy to generate and refine hypotheses through active environmental interaction. Building a scientific discovery framework around agentic models could therefore shift the paradigm from using LLMs as passive tools to empowering them as autonomous agents that drive the entire discovery lifecycle. In this paper, we focus on equation discovery, a fundamental task in science.

Mathematical equations are central to scientific progress, serving as concise and interpretable models of physical phenomena (Lemos et al., 2022; Batra et al., 2021; Hernandez et al., 2019). The task of data-driven equation discovery, also known as symbolic regression (SR), is an NP-hard problem due to its vast search space (Virgolin & Pissis, 2022). Traditional methods have relied on Genetic Programming (GP) for combinatorial search (Cranmer, 2023) or on deep neural networks trained on large-scale synthetic data for direct prediction (Kamienny et al., 2022; Biggio et al., 2021), which often suffer from efficiency and scalability issues. Recently, the research community has begun to embed LLMs into GP algorithms as equation proposers, leveraging their scientific prior knowledge to generate more effective hypotheses (Shojaee et al., 2025a; Grayeli et al., 2024). While this can enhance search efficiency, LLMs primarily serve as equation generators in a fixed pipeline, which lacks the autonomy to gain insight for equation design through tools.

To address these limitations, we introduce SR-SCIENTIST, a framework designed to enable LLMs to find scientific equations through long-horizon optimization driven by experimental feedback and analysis. To achieve this, we wrap the code interpreter as tools to analyze data and evaluate equations. As shown in Figure 1, the LLM agent is instructed to solve problems by utilizing these tools, including writing code to analyze data, implementing the equation as code, submitting it for evaluation, and optimizing the equation based on experimental feedback. In the process, we emphasize long-horizon optimization, allowing the agent to interact with data through tools over multiple turns (e.g., more than twenty) to gather sufficient information for equation design. To overcome the context length limitations of current LLMs, we implement an experience buffer to

[1]Shanghai Jiao Tong University [2]Shanghai Innovation Institute [3]GAIR. Correspondence to: Shijie Xia <shijiexia@sjtu.edu.cn>.

Accepted at ICML 2026 AI for Science Workshop.

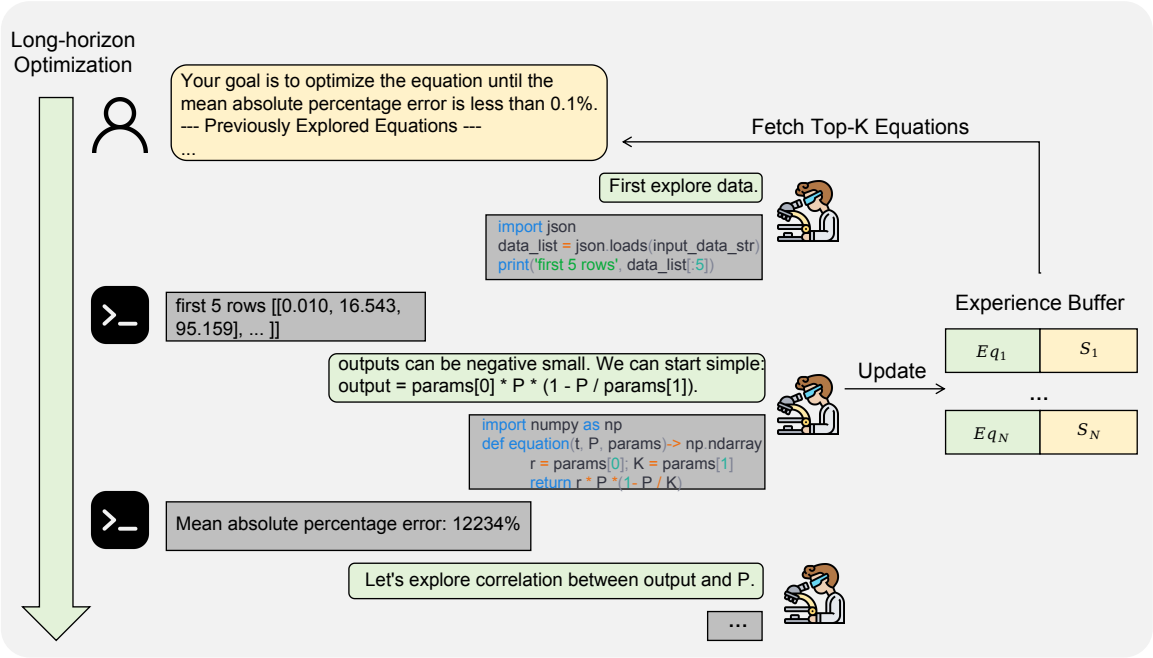

Figure 1. The inference framework of SR-SCIENTIST. At each iteration, the LLM agent autonomously conducts long-horizon optimization using code interpreters for data analysis and equation evaluation. To overcome the context length limitations of current LLMs, we implement an experience buffer to fetch the best-performing equations for subsequent iterations. 'Eq' denotes the equation and 'S' denotes the equation score.

store the explored equations and fetch the best-performing equations for subsequent iterations. Moreover, we adhere to the principle of minimal human-defined pipelines (Sutton, 2019), in which the agent is free to determine its own workflow for a given problem. Within the flexible framework, we develop a reinforcement learning (RL) pipeline from training data construction to reward design to help the LLM agent evolve its capabilities.

To validate the effectiveness of our framework, we evaluate SR-SCIENTIST using 5 different models as backbone LLMs on datasets covering 4 science disciplines carefully designed to prevent the LLMs from relying on memorized equations from their training data (Shojaee et al., 2025b). Results demonstrate that SR-SCIENTIST consistently outperforms state-of-the-art SR methods, with four of the models achieving an absolute margin of 6% to 35% over baselines. We also observe significant improvement after applying RL training. Additionally, we demonstrate our method's robustness to noise, the generalization of the discovered equations to out-of-domain data, and their symbolic accuracy. Furthermore, our ablation studies highlight the importance of data analysis, the experience buffer, and long-horizon optimization.

Overall, our contributions are as follows:

- We develop SR-SCIENTIST, a framework where an autonomous agent discovers scientific equations through long-horizon, tool-driven data analysis and equation evaluation.
- We show that SR-SCIENTIST significantly outperforms baseline methods in precision, generalization, robustness to noise, and symbolic accuracy.
- We develop a corresponding end-to-end reinforcement learning pipeline to enhance the agent's capabilities.

2. Related Work

Symbolic Regression SR aims to identify an interpretable mathematical expression that accurately describes the relationship between variables in observed data (Makke & Chawla, 2024; Cava et al., 2021). Current popular methods can be categorized as follows: 1) *GP-based Methods*: This line of methods uses constrained representations, such as expression trees, to define a search space composed of symbolic operators, constants, and variables (Cranmer, 2023). Evolutionary or genetic programming is then applied to search for solutions. 2) *Deep learning Methods*: These methods leverage deep neural networks to learn a direct mapping from numerical observations to their underlying mathematical expressions through online (Petersen et al., 2021; Landajuela et al., 2022) or offline learning (Kamienny et al., 2022; Biggio et al., 2021). 3) *LLM-based methods*:

Due to the efficiency and scalability issues of the above methods, the research community has integrated LLMs with traditional techniques like GP, leveraging the models' extensive knowledge to generate more informed hypotheses (Shojaee et al., 2025a; Grayeli et al., 2024; Ma et al., 2024; Wang et al., 2025b). While promising, current hybrid methods still lack the direct analysis of observed data through tools to gain insights. Furthermore, the LLM's behavior is often fixed, lacking the autonomy to decide its actions in a goal-oriented manner. Moreover, most work focuses on using LLMs for inference, without exploring how these models evolve their abilities through methods like RL.

Agentic AI Agentic AI can autonomously execute complex tasks in dynamic environments requiring adaptation, interaction, and reasoning in a goal-oriented manner (Schneider, 2025). The open-source community has built powerful agentic models for tasks like software engineering (Team et al., 2025; Zeng et al., 2025a), search (Gao et al., 2025), and GUI operations (Qin et al., 2025; Wang et al., 2025a), enabling them to solve tasks in an end-to-end manner with minimal human intervention. However, there has been limited focus on scientific discovery. In this paper, we demonstrate that with careful design, these agentic models can be powerful for equation discovery and can also improve their abilities through RL.

3. SR-Scientist

3.1. Problem Formulation

Symbolic regression aims to find a symbolic expression that describes the underlying relationship in a dataset. Formally, given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ is an input vector and $y_i \in \mathbb{R}$ is a scalar output, an SR method seeks to find a concise symbolic function f such that $f(\mathbf{x}_i) \approx y_i$. The goal of SR is not only to discover a function that minimizes the error between the predicted values and the true values but also one that generalizes well to unseen inputs while maintaining interpretability.

3.2. Inference Framework

We outline the overall inference framework in Algorithm 1. At each iteration, we set a goal G_i for a desired optimization precision and the agent is instructed to find an equation that satisfies the precision. We choose the mean absolute percentage error[1] (MAPE) as the goal: MAPE $= \frac{100\%}{n}\sum_{i=1}^n \left|\frac{y_i - f(\mathbf{x}_i)}{y_i}\right|$. Then, the LLM agent solves the problem by interleaving internal reasoning with external environment feedback. Formally, the agent's trajectory is structured as follows, typically within a ReAct (Yao et al., 2023) framework:

$$(r_1, \mathcal{T}_1, o_1), (r_2, \mathcal{T}_2, o_2), ..., (r_k, \mathcal{T}_k, o_k) \tag{1}$$

where r_i denotes the model's natural language reasoning at step i, $\mathcal{T}_i \subseteq \mathcal{T}$ is the set of tools invoked at step i, and o_i is the observation received after executing the tools in $\mathcal{T}_i$. We also set a maximum number of interaction turns M for the trajectory to avoid excessively long inference times without improved performance.

Tool Design We provide a code interpreter as the primary tool, enabling the agent to interact with the data and validate hypotheses. Specifically, we wrap the code interpreter into two common tools for the tasks, `data analyzer` and `equation evaluator`, denoted as T_1 and T_2, respectively. For T_1, we link it to the observed data and include a code example in the prompt demonstrating how to access the data. Through this tool, the agent can write and execute diverse code to analyze the data, such as inspecting data samples, conducting statistical analysis, or analyzing the residuals between the predicted value and the true value[2]. Notably, we do not constrain the code to these examples, allowing for diverse and emergent analysis patterns. For T_2, following the approach of Shojaee et al. (2025a), we design the tool to accept an equation skeleton with placeholders for constants in code format (See Figure 1 or 4 for the equation example). During the execution phase, the BFGS algorithm is implemented to optimize these constants and then report the performance of the equation. It also accepts equations with constants decided through other methods like data analysis. This tool prevents the agent from writing repetitive code for equation evaluation during the long-horizon exploration process.

Experience Buffer While the agent can continue the optimization process over multiple turns, the model's limited context length presents a challenge. Moreover, the equations can still perform poorly after a long-horizon optimization. To overcome this, we maintain an experience buffer $E = \{(e_i, s_i)\}_{i=1}^N$ to record the equations the agent has explored, where e_i denotes the equation and s_i is its corresponding MAPE. At the beginning of each iteration, we fetch the best K equations from the buffer and provide them to the agent as in-context examples. The optimization goal

[1] This is slightly different from other SR methods that usually use the mean squared error (MSE) or normalized mean squared error (NMSE) as the score, as we find MAPE provides a more uniform precision target across different data than the aforementioned metrics. To ensure a fair comparison with other SR methods, we select the equation with the lowest NMSE in practical evaluation.

[2] Since we primarily use a text-only model that cannot accept figures as input, we explicitly instruct the model via prompts to avoid generating plots. However, the model backbone is orthogonal to our framework, which intrinsically supports such plot analysis patterns.

Algorithm 1 SR-SCIENTIST inference framework.

Input: Iterations N; maximum turns M; number of fetched equations K; the initial goal G_1

```
# Store ranked equations by heap
H ← heap()
for i = 1, ..., N do
  # Generate candidate equations from LLM
  if i == 1 then
    E_i ← LLM(M, G_i)
  else
    E_i ← LLM(H.topk(K), M, G_i)
  end if
  H.append(E_i)
  # Check stopping condition and update goal
  if stopping_condition(H) then
    break
  else
    G_{i+1} ← UpdateGoal(H, G_i)
  end if
end for
```

Output: `H.topk(1) # Return the best`

is also updated if it was reached in the previous iteration. This mechanism effectively bypasses the context length limitation. Additionally, we explored using GP algorithms for experience management but observed no significant improvement.

Stopping Condition and Submission We stop the exploration process when the maximum number of iterations is reached or the equation's error is sufficiently small. Then, we select the equation with the best performance on the observed data and submit it for final evaluation.

3.3. Training Framework

Training Data Construction Following the approach of Shojaee et al. (2025b), we employ a mixed strategy of rule-based and model-based data synthesis. For each scientific scenario covering multiple variables, we instruct an LLM to synthesize potential relationships between the variables, typically in the form of an equation skeleton. For each skeleton, we use the model to determine the values for the constants based on their physical significance and thereby construct the full equation. Once the complete equation is formulated, we define an appropriate range of values for its variables and synthesize the observed data accordingly. This dataset is then split into training and evaluation sets. The training data is accessible to the agent during rollouts, while the evaluation data is used to measure performance and calculate the reward.

Reward Design In the rollout process, the LLM agent is instructed to find an equation that satisfies the precision specified by a MAPE goal s_{goal}. Then, the LLM agent interacts with the environment and generates multiple equations. To simplify the RL infrastructure, the agent conducts the optimization process for one iteration. Unlike math or code tasks that typically assign binary rewards based on the outcome (DeepSeek-AI et al., 2025; Zeng et al., 2025b), the performance of equations can be measured by continuous metrics such as MAPE. This makes it possible to assign continuous rewards to avoid reward sparsity. Since we only submit the best equation at inference (Petersen et al., 2021), we select the best equation from those explored and use its score s for the calculation. We employ a log-linear reward function that maps s to the range $[0, 1]$ as follows:

$$\mathcal{R} = \text{clip}\left(\frac{\lg s_{\text{max}} - \lg s}{\lg s_{\text{max}} - \lg s_{\text{goal}}}, 0, 1\right) \tag{2}$$

where s_{max} represents the maximum MAPE for which a non-zero reward can be gained. Additionally, we explore other reward functions and present the discussion in Appendix C.2.

Training Algorithm We apply the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm for optimization. To encourage exploration, we omit the KL penalty term against a reference model and observe that this leads to faster convergence and comparable performance.

4. Experiments

4.1. Setup

Dataset Due to the vast corpus that LLMs have been pretrained on, the evaluation dataset must be carefully curated to prevent LLMs from having memorized the equations. We evaluate our methods on the synthetic part of the LLM-SRBench (Shojaee et al., 2025b), denoted as LSR-Synth[3]. It combines known terms in the underlying equation with synthetic, novel terms to create problems that go beyond memorization. Furthermore, all equations and visualizations of their generated data points were verified by two subject matter experts to ensure scientific rigor. It contains 129 problems, spanning four scientific domains: chemistry (36), biology (24), physics (44), and material science (25). For each problem, it has three types of datasets: a training set accessible to the SR method, an in-domain (ID) test set, and an out-of-domain (OOD) test set.

Evaluation Metrics We use accuracy-to-tolerance as our main metric. We find it serves as a more robust metric when aggregating the overall results covering multiple problems

[3]We also considered LSR-Transform (Shojaee et al., 2025b) as a candidate for the evaluation dataset. However, we found that it might still have memorization issues, as nearly 50% of the problems achieved a sufficiently small error after the first iteration.

Method	Overall		Material Science		Chemistry		Biology		Physics	
	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
Without LLMs										
GPLearn	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
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
Qwen3-Coder-480B-A35B-Instruct										
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
GLM-4.5-Air										
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
GPT-OSS-120B										
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**
GPT-OSS-20B										
LaSR	12.66	8.53	14.67	8.00	15.74	12.96	11.11	6.95	9.85	6.06
LLM-SR	33.33	12.92	70.67	46.67	28.71	5.56	22.22	2.78	21.97	5.30
SR-SCIENTIST	42.64	23.00	62.67	40.00	49.07	23.15	34.72	19.44	30.30	15.15
Qwen3-Coder-30B-A3B-Instruct										
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
+ RL	40.92	20.69	85.33	65.33	37.38	7.46	29.17	11.11	25.00	11.37

Table 1. Comparison of SR-SCIENTIST and SR baseline models on different scientific benchmark problems measured by $Acc_{0.01}$ and $Acc_{0.001}$.

than others, like Normalized Mean Squared Error. It is also more challenging than other metrics like R^2. Please refer to Appendix D.1 for a detailed analysis. Given a desired tolerance threshold τ, the metric calculates whether the predicted values and the ground truth values satisfy it as follows:

$$\text{Acc}_\tau = \mathbb{1}\left(\max_{1\le i\le N_{\text{test}}} \left|\frac{\hat{y}_i - y_i}{y_i}\right| \le \tau\right) \tag{3}$$

We discard the 5% worst predictions to prevent outliers due to the presence of the max operator, following Biggio et al. (2021); Kamienny et al. (2022). Additionally, we present symbolic accuracy, which checks if the discovered equations are identical to the ground truth equations. We present the details for the symbolic accuracy calculation in Appendix D.6.

Baselines We comprehensively compare our methods against several baseline methods. For methods without LLMs, we include GPLearn, E2E (Kamienny et al., 2022), NeSymReS (Biggio et al., 2021), DSR (Petersen et al., 2021), uDSR (Landajuela et al., 2022), and PySR (Cranmer, 2023). For the LLM-based methods, we include LLM-SR and LaSR for comparison. For methods without LLMs, we constrain the number of equation candidates for each problem to 100,000. For methods with LLMs, we constrain the LLM calls to 1,000, with at most 1,000 equation candidates from each problem. For SR-SCIENTIST, we set the maximum number of turns per iteration to 25, with a total of 40 iterations. We present the configuration details of the baseline methods in Appendix A and the configuration of SR-SCIENTIST in Appendix B. We evaluate our method with different models as backbone LLMs, including Qwen3-Coder-480B-A35B (Qwen, 2025), GLM-4.5-Air (Zeng et al., 2025a), GPT-OSS-120B (OpenAI, 2025), GPT-OSS-20B, and Qwen3-Coder-30B-A3B. Due to computation costs and training infrastructure limitations, we

Table 2. Symbolic accuracy (SA) of different methods on LSR-Synth. GLM, GPT, and Qwen represent GLM-4.5-Air, GPT-OSS-120B, and Qwen3-Coder-480B, respectively.

Method	SA
uDSR	0.77
PySR	4.65
LLM-SR (GLM)	5.43
LLM-SR (GPT)	4.65
LLM-SR (Qwen)	5.43
SR-SCIENTIST (GLM)	7.75
SR-SCIENTIST (GPT)	7.00
SR-SCIENTIST (Qwen)	7.00

use only Qwen3-Coder-30B-A3B for RL training, while the other models are used for inference only. However, we also verify the effectiveness of RL training on other models and present the results in Appendix C.4. For each experiment, we repeat it three times and report the average value to reduce noise.

Training Details For training data construction, we synthesize training data for the scientific scenarios in LSR-Synth. To prevent data contamination, two authors of this paper independently and carefully checked the equation skeletons to prevent duplicates and confirmed their agreement. Finally, we obtained 1024 problems for RL training. Please refer to Appendix C.1 for details. During rollout, we set the maximum number of turns to 20 and train for 60 steps. We set s_{max} to 100% and s_{goal} to 0.1% in the reward design. The detailed training configure can be found in the Appendix C.3.

4.2. Main Results

Precision Table 1 presents the accuracy results. In terms of overall accuracy, SR-SCIENTIST consistently outperforms the baseline methods, with four of the models achieving an absolute performance margin of 6% to 35% over the baselines. Notably, when using GPT-OSS-120B as a backbone, SR-SCIENTIST achieves the highest overall performance, with an $Acc_{0.01}$ of 63.57% and an $Acc_{0.001}$ of 49.35%. In terms of performance across different subjects, with backbones such as Qwen3-Coder-480B, GLM-4.5-Air, and GPT-OSS-120B, SR-SCIENTIST surpasses other LLM-based methods in all subjects. For Qwen3-Coder-30B, end-to-end RL training significantly improves its performance compared to the original model in all subjects, highlighting that the agent can evolve its abilities through its own experience. We provide further analysis of the LLM backbone for RL training in Appendix C.4 and the computation cost discussion in Appendix D.2.

OOD Performance Figure 2 shows the performance of the discovered equations on OOD data. For material science, the performance of all methods improves when shifting from ID to OOD data, while for other subjects, the methods exhibit varying trends. Among them, SR-SCIENTIST still achieves the best performance on OOD data, demonstrating its strong generalization capabilities. The results of other models are presented in Appendix D.3 and show similar trends.

Robustness to Noise To test the robustness to noise in the observed data, we add Gaussian noise with different standard deviations ($\sigma = \{0.01, 0.05, 0.1\}$) to each training data point the model has access to and report the overall performance. As shown in Figure 3, while the performance of all methods drops as the noise level increases, SR-SCIENTIST consistently performs better than other methods, especially with Qwen3-Coder-480B and GLM-4.5-Air as LLM backbones.

Symbolic Accuracy Table 2 shows the symbolic accuracy of representative methods, evaluated on all problems in the dataset. Overall, it is challenging to identify equations identical to the ground truth from observed data. Among these methods, SR-SCIENTIST achieves the best performance, correctly identifying the most equations. For a more qualitative assessment, Figure 4 presents the discovered equations for the PO10 and PO37 physics problems, which are related to nonlinear oscillators. For both problems, SR-SCIENTIST identifies the structure of the equation and its constants, while the equations from other methods are usually more complex and less accurate. Additionally, SR-SCIENTIST produces trajectories that illustrate its derivation process, providing information that can inspire human scientists to design better equations.

4.3. Analysis

Ablation studies Table 3 presents the results of our ablation studies on the data analysis and experience buffer components. The findings indicate that allowing the agent to perform data analysis is crucial for performance, particularly for the GPT-OSS-120B and Qwen3-Coder-480B models. Moreover, the experience buffer plays an important role in the long-term memory of the agent, allowing it to utilize the experience of top-performing equations from previous iterations. Either directly removing the experience buffer or randomly sampling equations from it leads to a significant performance drop.

The Effect of the Turn Number We fixed the LLM calls at around 1,000 and traded off between maximum turns and iterations to investigate the effect of different exploration lengths at each iteration. Increasing the maximum turns

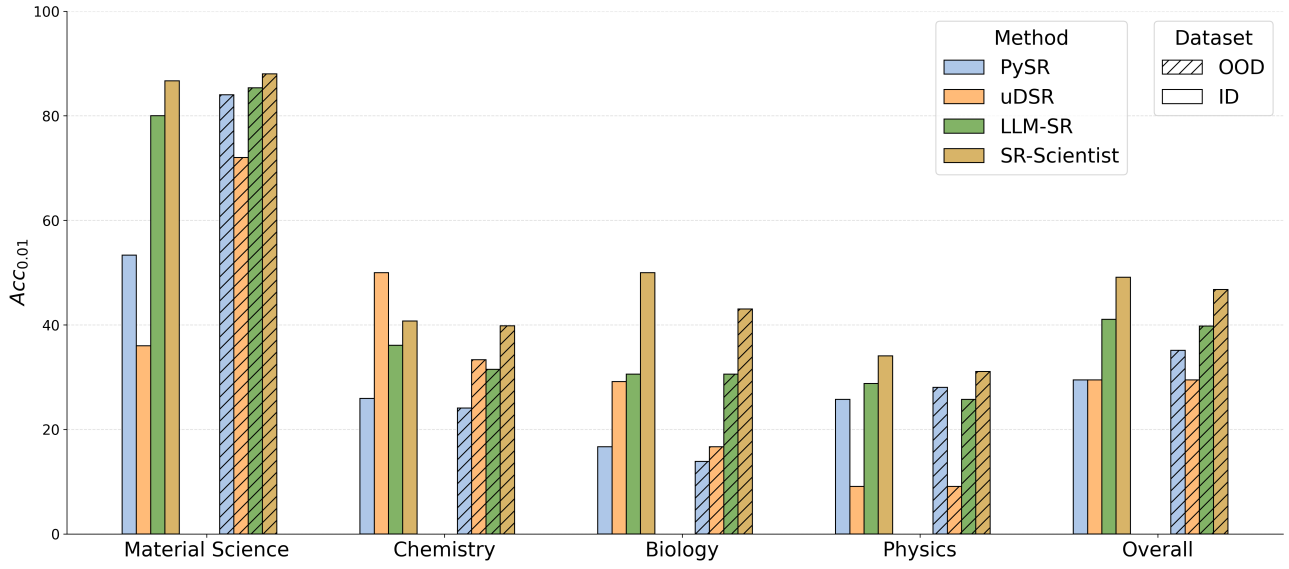

Figure 2. Detailed results of in-domain (ID) and out-of-domain (OOD) performance using $\text{Acc}_{0.01}$ across various LSR-Synth scientific domains. (with Qwen3-Coder-480B as LLM backbone)

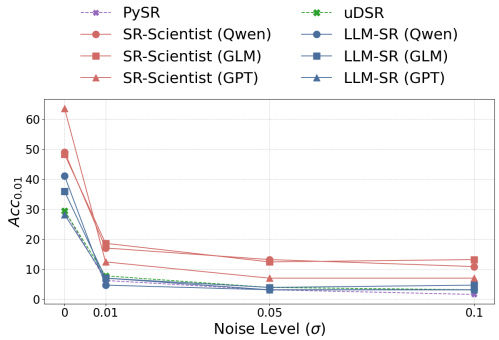

Figure 3. Noise robustness analysis on LSR-Synth. Qwen, GLM, and GPT represent Qwen3-Coder-480B, GLM-4.5-Air, and GPT-OSS-120B, respectively.

from 10 to 25, as shown in Figure 5, significantly improves performance, highlighting the value of long-horizon optimization. However, performance stagnates or slightly drops beyond 25 turns, indicating that excessively long exploration does not bring additional performance and that the inference budget should be allocated to initiating new iterations.

Table 3. Ablation studies. In 'w/o T_1', the agent can not utilize the `data analyzer` tool. In 'w/o experience', the agent can not utilize the experience buffer and optimize from scratch for each iteration. In 'w/o top-k', we randomly sample equations from the experience buffer instead of using top-k sampling. GPT, Qwen, and GLM represent GPT-OSS-120B, Qwen3-Coder-480B, and GLM-4.5-Air, respectively.

Method	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
SR-SCIENTIST (GPT)	63.57	49.35
w/o T_1	35.66	16.28
w/o experience	57.36	41.86
w/o top-k	58.14	41.86
SR-SCIENTIST (Qwen)	49.09	24.55
w/o T_1	41.08	14.73
w/o experience	35.66	16.28
w/o top-k	26.36	13.70
SR-SCIENTIST (GLM)	48.32	25.06
w/o T_1	46.51	22.48
w/o experience	37.21	18.61
w/o top-k	41.96	21.71

Tool Call Analysis We analyzed the distribution of tool call types and instructed LLMs to classify the `data analyzer` behaviors, presenting the results in Figure 6. Overall, the Qwen and GLM families exhibit similar patterns, with around 80% of their calls dedicated to equation evaluation and 20% to data analysis. Within the data analysis calls, data statistics generally account for a higher percentage, showing the importance of calculating relevant values such as correlations and averages. In contrast, GPT-OSS-20B and GPT-OSS-120B tend to directly write their own code to perform residual error analysis for more fine-grained information. Further case studies show that GPT-OSS-120B also tends to directly define equation constants through data analysis, demonstrating greater flexibility. Moreover, after RL training, the Qwen3-Coder-30B also increases its use of data statistics for more advanced data analysis.

PO10

Ground Truth

$$\dot{v}(t) = -3.98x(t) - 2x(t)^3 - 0.05x(t)t + 0.8sin(t)$$

PySR

$$\dot{v}(t) = -(0.049441915x(t)t + \frac{x(t)}{0.12852237}) + 0.78994745\sin(t) + \sin(\frac{x(t)}{0.49664396}) + \sin(\frac{x(t)}{0.5089395})$$

SR-Scientist (Qwen3-Coder-480B)

```
def equation(x, t, params):
    output = - params[0] * x - params[1] * x**3
             - params[2] * x * t
             + params[3] * np.sin(params[4] * t
                                  + params[5])
    return output
```

$$\dot{v}(t) = -3.98x(t) - 2x(t)^3 - 0.05x(t)t + 0.8sin(t)$$

LLM-SR (Qwen3-Coder-480B)

```
def equation(x, t, params):
    output = - params[0]*x - params[1]*x**3
     + params[2]*np.cos(params[3]*t)
     + params[4]*np.exp(-params[5]*x**2)*np.sin(t)
    return output
```

$$\dot{v}(t) = -4.96x(t) - 1.85x(t)^3 - 0.02cos(1.59t) + 0.79sin(t)e^{-0.01t}$$

PO37

Ground Truth

$$\dot{v}(t) = -1.73x(t) - 1.92v(t) + 0.72sin(t) - 0.14x(t)t - 0.086v^3$$

PySR

$$\dot{v}(t) = -1.069x(t) - 0.052v(t) + 1.069\sin(3.883\exp(-t))\cos(1.136\sqrt{t} - 1.136t + 0.454)$$

SR-Scientist (GPT-OSS-120B)

```
def equation(x, t, v, params):
    return (-1.73 * x
            -1.92 * v
            +0.72 * np.sin(t)
            -0.14 * x * t
            -0.086 * v**3)
```

$$\dot{v}(t) = -1.73x(t) - 1.92v(t) + 0.72sin(t) - 0.14x(t)t - 0.086v^3$$

LLM-SR (GPT-OSS-120B)

```
def equation(x, t, v, params):
    k, beta, c, A, omega = params[:5]
    phi, d, kappa, gamma = params[5:9]
    k_eff = k + kappa * t
    nonlinear = -beta * x**3 - gamma * x**5
    drive = A * np.sin(omega * t + phi)
    return -k_eff * x + nonlinear - c * v + drive + d
```

$$\dot{v}(t) = -(1.725 + 0.139*t)*x(t) + 0.047*x(t)^3 - 0.054*x(t)^5 - 1.918*v(t) + 0.716*sin(t - 6.281) + 0.00000117$$

Figure 4. Equations discovered for the PO10 and PO37 physics problems. The variables $\dot{v}(t)$, t, x, and v represent acceleration in a non-linear harmonic oscillator, time, position, and velocity, respectively. Terms highlighted in green are common to both the predicted and ground truth equations.

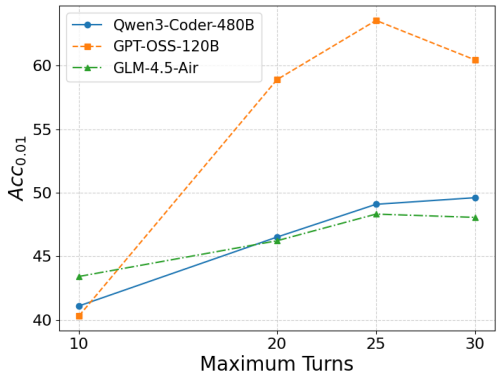

Figure 5. Overall performance under different maximum turns. We keep the total number of LLM calls at around 1,000 and trade off between maximum turns and iterations.

Figure 6. The tool call behaviors of different models.

5. Limitations and Future Directions

First, in our main experiments, we only used text-only models. Expanding the model to incorporate multi-modal inputs and include plots in the analysis phase is a promising direction. Second, in noisy scenarios, although SR-SCIENTIST performs best, it still suffers from a significant performance drop. Future work could explore incorporating common noise-handling strategies into the agent's prompt to guide its data analysis. Third, since we only provide the top-performing equations to the agent in each iteration, the agent may still re-explore equations that previously performed poorly. Optimizing the agent's memory system is an interesting future direction.

6. Conclusion

In this paper, we introduce SR-SCIENTIST, a framework that transforms the LLM from a passive equation proposer into an autonomous scientist for symbolic regression. Our experiments show that SR-SCIENTIST significantly outperforms existing methods in precision, generalization, robustness to noise, and symbolic accuracy. Furthermore, we develop a complete reinforcement learning pipeline that allows the agent to self-evolve and enhance its capabilities.

Acknowledgments

We would like to express our great gratitude to Yan Ma for assisting with the writing and to Xuefeng Li for providing advice on RL Infra. This work was partially funded by the National Natural Science Foundation of China (62476168), the National High Technology Research and Development Program of China (2015AA015408), the Shanghai Science and Technology Development Funds (14ZR1403200), and the AI for Science Program of the Shanghai Municipal Commission of Economy and Informatization (2025-GZL-RGZN-BTBX-01013).

References

Anthropic. claude-code. https://github.com/anthropics/claude-code, 2025.

Batra, R., Song, L., and Ramprasad, R. Emerging materials intelligence ecosystems propelled by machine learning. *Nature Reviews Materials*, 6(8):655–678, 2021.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales, 2021. URL https://arxiv.org/abs/2106.06427.

Cava, W. L., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance, 2021. URL https://arxiv.org/abs/2107.14351.

Cranmer, M. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023. URL https://arxiv.org/abs/2305.01582.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Gao, J., Fu, W., Xie, M., Xu, S., He, C., Mei, Z., Zhu, B., and Wu, Y. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl, 2025. URL https://arxiv.org/abs/2508.07976.

Google. Gemini cli. https://github.com/google-gemini/gemini-cli, 2025.

Grayeli, A., Sehgal, A., Costilla-Reyes, O., Cranmer, M., and Chaudhuri, S. Symbolic regression with a learned concept library, 2024. URL https://arxiv.org/abs/2409.09359.

Hernandez, A., Balasubramanian, A., Yuan, F., Mason, S., and Mueller, T. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression, 2019. URL https://arxiv.org/abs/1904.01095.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers, 2022. URL https://arxiv.org/abs/2204.10532.

Landajuela, M., Lee, C. S., Yang, J., Glatt, R., Santiago, C. P., Aravena, I., Mundhenk, T., Mulcahy, G., and Petersen, B. K. A unified framework for deep symbolic regression. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 33985–33998. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/dbca58f35bddc6e4003b2dd80e42f838-Paper-Conference.pdf.

Lemos, P., Jeffrey, N., Cranmer, M., Ho, S., and Battaglia, P. Rediscovering orbital mechanics with machine learning, 2022. URL https://arxiv.org/abs/2202.02306.

Ma, P., Wang, T.-H., Guo, M., Sun, Z., Tenenbaum, J. B., Rus, D., Gan, C., and Matusik, W. Llm and simulation as

bilevel optimizers: A new paradigm to advance physical scientific discovery, 2024. URL https://arxiv.org/abs/2405.09783.

Makke, N. and Chawla, S. Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, 57(1):2, 2024.

Newell, A., Simon, H. A., et al. *Human problem solving*, volume 104. Prentice-hall Englewood Cliffs, NJ, 1972.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2021. URL https://arxiv.org/abs/1912.04871.

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., Zhong, W., Li, K., Yang, J., Miao, Y., Lin, W., Liu, L., Jiang, X., Ma, Q., Li, J., Xiao, X., Cai, K., Li, C., Zheng, Y., Jin, C., Li, C., Zhou, X., Wang, M., Chen, H., Li, Z., Yang, H., Liu, H., Lin, F., Peng, T., Liu, X., and Shi, G. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

Qwen. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Schneider, J. Generative to agentic ai: Survey, conceptualization, and challenges, 2025. URL https://arxiv.org/abs/2504.18875.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Shojaee, P., Meidani, K., Gupta, S., Farimani, A. B., and Reddy, C. K. Llm-sr: Scientific equation discovery via programming with large language models, 2025a. URL https://arxiv.org/abs/2404.18400.

Shojaee, P., Nguyen, N.-H., Meidani, K., Farimani, A. B., Doan, K. D., and Reddy, C. K. Llm-srbench: A new benchmark for scientific equation discovery with large language models, 2025b. URL https://arxiv.org/abs/2504.10415.

Sutton, R. The bitter lesson. *Incomplete Ideas (blog)*, 13(1): 38, 2019.

Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., Chen, Z., Cui, J., Ding, H., Dong, M., Du, A., Du, C., Du, D., Du, Y., Fan, Y., Feng, Y., Fu, K., Gao, B., Gao, H., Gao, P., Gao, T., Gu, X., Guan, L., Guo, H., Guo, J., Hu, H., Hao, X., He, T., He, W., He, W., Hong, C., Hu, Y., Hu, Z., Huang, W., Huang, Z., Huang, Z., Jiang, T., Jiang, Z., Jin, X., Kang, Y., Lai, G., Li, C., Li, F., Li, H., Li, M., Li, W., Li, Y., Li, Y., Li, Z., Li, Z., Lin, H., Lin, X., Lin, Z., Liu, C., Liu, C., Liu, H., Liu, J., Liu, J., Liu, L., Liu, S., Liu, T. Y., Liu, T., Liu, W., Liu, Y., Liu, Y., Liu, Y., Liu, Y., Liu, Z., Lu, E., Lu, L., Ma, S., Ma, X., Ma, Y., Mao, S., Mei, J., Men, X., Miao, Y., Pan, S., Peng, Y., Qin, R., Qu, B., Shang, Z., Shi, L., Shi, S., Song, F., Su, J., Su, Z., Sun, X., Sung, F., Tang, H., Tao, J., Teng, Q., Wang, C., Wang, D., Wang, F., Wang, H., Wang, J., Wang, J., Wang, J., Wang, S., Wang, S., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Z., Wang, Z., Wang, Z., Wei, C., Wei, Q., Wu, W., Wu, X., Wu, Y., Xiao, C., Xie, X., Xiong, W., Xu, B., Xu, J., Xu, J., Xu, L. H., Xu, L., Xu, S., Xu, W., Xu, X., Xu, Y., Xu, Z., Yan, J., Yan, Y., Yang, X., Yang, Y., Yang, Z., Yang, Z., Yang, Z., Yao, H., Yao, X., Ye, W., Ye, Z., Yin, B., Yu, L., Yuan, E., Yuan, H., Yuan, M., Zhan, H., Zhang, D., Zhang, H., Zhang, W., Zhang, X., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Z., Zhao, H., Zhao, Y., Zheng, H., Zheng, S., Zhou, J., Zhou, X., Zhou, Z., Zhu, Z., Zhuang, W., and Zu, X. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Virgolin, M. and Pissis, S. P. Symbolic regression is np-hard, 2022. URL https://arxiv.org/abs/2207.01018.

Wang, H., Zou, H., Song, H., Feng, J., Fang, J., Lu, J., Liu, L., Luo, Q., Liang, S., Huang, S., Zhong, W., Ye, Y., Qin, Y., Xiong, Y., Song, Y., Wu, Z., Li, A., Li, B., Dun, C., Liu, C., Zan, D., Leng, F., Wang, H., Yu, H., Chen, H., Guo, H., Su, J., Huang, J., Shen, K., Shi, K., Yan, L., Zhao, P., Liu, P., Ye, Q., Zheng, R., Xin, S., Zhao, W. X., Heng, W., Huang, W., Wang, W., Qin, X., Lin, Y., Wu, Y., Chen, Z., Wang, Z., Zhong, B., Zhang, X., Li, X., Li, Y., Zhao, Z., Jiang, C., Wu, F., Zhou, H., Pang, J., Han, L., Liu, Q., Ma, Q., Liu, S., Cai, S., Fu, W., Liu, X., Wang,

Y., Zhang, Z., Zhou, B., Li, G., Shi, J., Yang, J., Tang, J., Li, L., Han, Q., Lu, T., Lin, W., Tong, X., Li, X., Zhang, Y., Miao, Y., Jiang, Z., Li, Z., Zhao, Z., Li, C., Ma, D., Lin, F., Zhang, G., Yang, H., Guo, H., Zhu, H., Liu, J., Du, J., Cai, K., Li, K., Yuan, L., Han, M., Wang, M., Guo, S., Cheng, T., Ma, X., Xiao, X., Huang, X., Chen, X., Du, Y., Chen, Y., Wang, Y., Li, Z., Yang, Z., Zeng, Z., Jin, C., Li, C., Chen, H., Chen, H., Chen, J., Zhao, Q., and Shi, G. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning, 2025a. URL https://arxiv.org/abs/2509.02544.

Wang, R., Wang, B., Li, K., Zhang, Y., and Cheng, J. Drsr: Llm based scientific equation discovery with dual reasoning from data and experience, 2025b. URL https://arxiv.org/abs/2506.04282.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Zeng, A., Lv, X., Zheng, Q., Hou, Z., Chen, B., Xie, C., Wang, C., Yin, D., Zeng, H., Zhang, J., Wang, K., Zhong, L., Liu, M., Lu, R., Cao, S., Zhang, X., Huang, X., Wei, Y., Cheng, Y., An, Y., Niu, Y., Wen, Y., Bai, Y., Du, Z., Wang, Z., Zhu, Z., Zhang, B., Wen, B., Wu, B., Xu, B., Huang, C., Zhao, C., Cai, C., Yu, C., Li, C., Ge, C., Huang, C., Zhang, C., Xu, C., Zhu, C., Li, C., Yin, C., Lin, D., Yang, D., Jiang, D., Ai, D., Zhu, E., Wang, F., Pan, G., Wang, G., Sun, H., Li, H., Li, H., Hu, H., Zhang, H., Peng, H., Tai, H., Zhang, H., Wang, H., Yang, H., Liu, H., Zhao, H., Liu, H., Yan, H., Liu, H., Chen, H., Li, J., Zhao, J., Ren, J., Jiao, J., Zhao, J., Yan, J., Wang, J., Gui, J., Zhao, J., Liu, J., Li, J., Li, J., Lu, J., Wang, J., Yuan, J., Li, J., Du, J., Du, J., Liu, J., Zhi, J., Gao, J., Wang, K., Yang, L., Xu, L., Fan, L., Wu, L., Ding, L., Wang, L., Zhang, M., Li, M., Xu, M., Zhao, M., Zhai, M., Du, P., Dong, Q., Lei, S., Tu, S., Yang, S., Lu, S., Li, S., Li, S., Shuang-Li, Yang, S., Yi, S., Yu, T., Tian, W., Wang, W., Yu, W., Tam, W. L., Liang, W., Liu, W., Wang, X., Jia, X., Gu, X., Ling, X., Wang, X., Fan, X., Pan, X., Zhang, X., Zhang, X., Fu, X., Zhang, X., Xu, Y., Wu, Y., Lu, Y., Wang, Y., Zhou, Y., Pan, Y., Zhang, Y., Wang, Y., Li, Y., Su, Y., Geng, Y., Zhu, Y., Yang, Y., Li, Y., Wu, Y., Li, Y., Liu, Y., Wang, Y., Li, Y., Zhang, Y., Liu, Z., Yang, Z., Zhou, Z., Qiao, Z., Feng, Z., Liu, Z., Zhang, Z., Wang, Z., Yao, Z., Wang, Z., Liu, Z., Chai, Z., Li, Z., Zhao, Z., Chen, W., Zhai, J., Xu, B., Huang, M., Wang, H., Li, J., Dong, Y., and Tang, J. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025a. URL https://arxiv.org/abs/2508.06471.

Zeng, W., Huang, Y., Liu, Q., Liu, W., He, K., Ma, Z., and He, J. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025b. URL https://arxiv.org/abs/2503.18892.

Appendix

A. Baseline Implementation Details

GPLearn The open-source `gplearn`[4] library is a Python package built on scikit-learn that uses genetic programming to perform SR. We set the population size to 500, the number of generations to 200, and the tournament size to 20. The function set includes addition, subtraction, multiplication, division, sine, cosine, square root, logarithm, absolute value, negation, and inverse.

End-to-End Symbolic Regression (E2E) E2E (Kamienny et al., 2022) is a method that uses a single deep learning model, often a Transformer, to directly predict a complete mathematical expression, including both its structure and numerical constants. We implement the method using the `symbolicregression` Facebook repository[5]. For this method, we set the maximum input points to 200 and the number of trees for refinement to 10.

NeSymReS NeSymReS (Biggio et al., 2021) is a deep learning approach that uses a pre-trained Transformer model to directly predict a mathematical expression from data points. We use the `NeuralSymbolicRegressionThatScales`[6] repository to implement this method. We set the number of data points passed to the Transformer to 500 and the beam size (expression sampling size) to 32.

Deep Symbolic Regression (DSR) DSR (Petersen et al., 2021) is a deep learning approach that uses a recurrent neural network and reinforcement learning to search for the mathematical expression that best fits a given dataset. We implement the method using the `deep-symbolic-optimization`[7] repository. For this method, we set the number of samples to 100,000, the batch size to 512, and the learning rate to 0.0005. The function set includes addition, subtraction, multiplication, division, sine, cosine, exponentiation, and logarithm, with expression lengths ranging from 1 to 9.

[4] https://gplearn.readthedocs.io/en/stable/

[5] https://github.com/facebookresearch/symbolicregression

[6] https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales

[7] https://github.com/dso-org/deep-symbolic-optimization

Unified Deep Symbolic Regression (uDSR) uDSR (Landajuela et al., 2022) is a deep learning framework that unifies multiple symbolic regression strategies into a single modular system. It extends DSR by combining approaches like neural-guided search, pre-training, and linear models. We implement it using the `deep-symbolic-optimization` repository. For the genetic programming configuration, we set the population size to 100, generations to 20, crossover probability to 0.5, mutation probability to 0.5, tournament size to 5, and maximum mutation tree depth to 3. Other parameters are the same as DSR's. The function set is: add, sub, mul, div, sin, cos, exp, log, and poly.

Python Symbolic Regression (PySR) The PySR Python package[8] (Cranmer, 2023) uses a powerful Julia backend to find simple, interpretable mathematical expressions that best fit the observed data. We set the number of iterations to 125, cycles per iteration to 550, populations to 15 with a population size of 33, maximum size to 30, and the randomization weight to 0.1. Its binary operators are $+, -, *, /$, pow, and its unary operators[9] are $\exp, \log, \text{sqrt}, \sin, \cos$.

Library-Augmented Symbolic Regression (LaSR) LaSR (Grayeli et al., 2024) uses LLMs to discover and evolve a library of abstract, natural-language concepts. We implement the method using the `LibraryAugmentedSymbolicRegression.jl`[10] repository. For this method, we set the number of iterations to 25, cycles per iteration to 550, populations to 10 with a population size of 33, maximum size to 30, and the maximum number of concepts to 20. The LLM operation weights for crossover, mutation, and randomization are each set to 0.02. The binary operators, unary operators, nested constraints, and other constraints are the same as those in PySR.

LLM-SR LLM-SR (Shojaee et al., 2025a) leverages the extensive scientific knowledge and code-generation capabilities of LLMs to discover mathematical equations. It treats equations as programs and combines an LLM's scientific priors with an evolutionary search to find accurate and interpretable formulas. We implement it using the `llm-srbench`[11] repository with its default configuration and set the maximum number of LLM calls to 1,000 for each problem.

[8] https://astroautomata.com/PySR/

[9] Nested Constraints: $\sin : \{\sin : 0, \cos : 0\}, \cos : \{\sin : 0, \cos : 0\}, \exp : \{\exp : 0, \log : 0\}, \log : \{\exp : 0, \log : 0\}, \text{sqrt} : \{\text{sqrt} : 0\}$; Constraints: $\{\sin : 10, \cos : 10, \exp : 20, \log : 20, \text{sqrt} : 20, \text{pow} : (-1, 20)\}$

[10] https://github.com/trishullab/LibraryAugmentedSymbolicRegression.jl

[11] https://github.com/deep-symbolic-mathematics/llm-srbench

B. Details of SR-SCIENTIST Inference Framework

For each iteration, we set the initial MAPE goal to 0.1% and the termination MAPE threshold to 0.0001%. For each LLM, we set the sampling temperature to 0.7 and the maximum completion length per call to 8,192. For the `equation evaluator` tool, to control the length and the complexity of the generated equations, we set the maximum number of parameters to 10 in all experiments, following Shojaee et al. (2025a). To ensure a fair comparison with methods like LLM-SR, the optimization goal for the BFGS algorithm is the mean square error. For parameter optimization, we use the `scipy` library for nonlinear optimization with the BFGS algorithm. The prompt for the LLM agent is shown in Figure 18 and Figure 13.

C. Details of SR-SCIENTIST Training Framework

C.1. Training Data Synthesis

We synthesize our training data following the approach of Shojaee et al. (2025a), using a hybrid model-based and rule-based method. The problems cover four scientific disciplines: materials science (Stress with respect to Strain and Temperature), chemistry (Reaction rate with respect to Time and Concentration), biology (Growth rate with respect to Time and Population size), and physics (Acceleration with respect to Time, Displacement, and Velocity). The detailed procedure is as follows:

(1) *Equation Skeleton Generation*: Each equation skeleton contains at least one known and one novel term. The known terms refer to common concepts from an LLM's pre-training knowledge, and the novel terms refer to terms outside the LLM's prior knowledge. We synthesize both types of terms by prompting Claude 4 Sonnet. These terms are then combined into equation skeletons using addition. To ensure a moderate level of complexity, each equation is limited to 2-4 total terms. To prevent potential data leakage between our training set and the benchmark data, two authors of the paper independently identify equations that are identical or too similar and discuss them for further reconciliation. Any equations deemed too similar are discarded.

(2) *Parameter Instantiation*: For each equation skeleton, we transform it into a complete equation by assigning values to each constant. This process is not a random assignment; rather, it is guided by the scientific context and significance of each term. For example, in the context of material science, the reference temperature

T_0 is set to 273.15 K (the triple point of water). We instruct the LLMs to perform value assignment, which is then followed by human validation.

(3) *Data Point Generation*: For the material science equations, which represent static systems, we generate data points by uniformly sampling across a defined range of parameters. The temperature T is sampled from the range of $[273\text{ K}, 573\text{ K}]$, and strain ϵ is sampled from $[0, 0.6]$. A total of 5000 points are generated using an evenly spaced grid. The OOD test set is created by taking the 500 data points with the highest temperature values. The remaining 4500 points are then randomly shuffled, with 500 points allocated to the test set and the rest forming the training set. For the chemistry, biology, and physics equations, which represent dynamic systems, we use the `solve_ivp` function from the SciPy Python package with the RK45 method to solve the differential equations. Given initial conditions at $t = 0$, we solve the equations to obtain the relationship between variables over time (equations that cannot be solved are simply discarded). We generate 5000 data points by uniformly sampling the time variable t from the range of $[0, 60]$. The OOD test set is created by selecting the 500 data points with the highest time values. The remaining 4500 points are randomly partitioned into a 500-point test set and the remaining training set.

(4) *Filtering and Final Dataset Assembly*: In the filtering stage, we compute the statistical properties of the resulting data. Equations with data points that are scientifically anomalous are filtered out. This process ensures that our final dataset contains only scientifically meaningful equations with plausible data points. Finally, the instantiated equations, their corresponding numerical data points, and other relevant information (variable symbols, equation names, etc.) are packed for the subsequent training.

C.2. Reward Design

Besides the log-linear reward design, we also explore the stepwise reward function as follows:

$$\mathcal{R} = \begin{cases} 1.0 & \text{if } s < 0.001 \\ 0.5 & \text{if } 0.001 \leq s < 0.01 \\ 0.25 & \text{if } 0.01 \leq s < 0.1 \\ 0.1 & \text{if } 0.1 \leq s < 1 \\ 0.0 & \text{if } s \geq 1 \end{cases} \tag{4}$$

where s denotes the MAPE value. As shown in Table 4, the log-linear reward function performs better than the stepwise function.

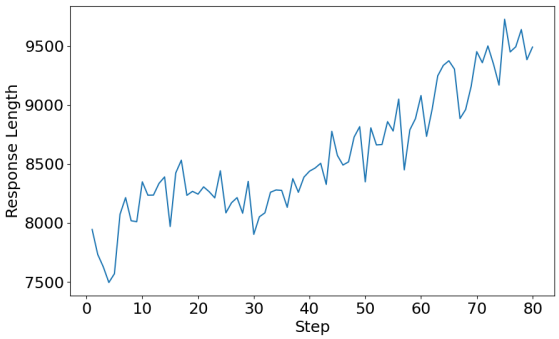

Figure 7. The change in response length during training.

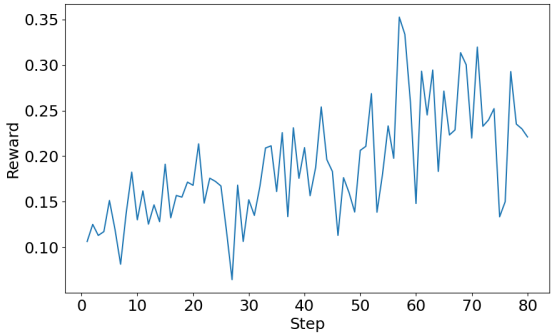

Figure 8. The change in reward score during training.

C.3. Training Details

We conduct RL training on 32 NVIDIA H200 GPUs. For the infrastructure, we use the `verl`[12] framework, employing SGLang as the rollout engine and FSDP as the training engine. For rollouts, we implement batch-level asynchronous rollouts to reduce the rollout time. We set the temperature to 1.0, the maximum response length to 10,240, and the maximum number of turns to 20. We use a prompt batch size of 32 and generate 8 rollouts per prompt. The KL loss coefficient is set to 0 to encourage exploration. The remaining parameters use the default settings.

Figures 7 and 8 present the training dynamics. As shown, the reward progressively increases and saturates at around 60 steps. The response length also increases, demonstrating that the model learns a long-horizon problem-solving strategy.

C.4. The Effect of LLM Backbone

To further investigate the effect of RL training on different models, we conduct RL training on the Qwen3-8B, Qwen3-14B, and Qwen3-32B models using the same training config-

[12] https://github.com/volcengine/verl

Method	Overall		Material Science		Chemistry		Biology		Physics	
	$Acc_{0.01}$	$Acc_{0.001}$	$Acc_{0.01}$	$Acc_{0.001}$	$Acc_{0.01}$	$Acc_{0.001}$	$Acc_{0.01}$	$Acc_{0.001}$	$Acc_{0.01}$	$Acc_{0.001}$
SR-SCIENTIST	32.30	16.02	81.33	52.00	22.22	5.56	22.22	8.33	18.18	8.33
+ RL (log-linear)	40.92	20.69	85.33	65.33	37.38	7.46	29.17	11.11	25.00	11.37
+ RL (stepwise)	37.21	17.83	84.00	56.00	27.78	2.78	33.33	16.67	20.45	9.09

Table 4. The performance of different reward functions. 'log-linear' refers to the reward function corresponding to Equation 2 and 'stepwise' refers to the reward function corresponding to Equation 4.

uration detailed in Appendix C.3 and present the results in Table 5. All models achieve significant performance gains after RL training, with most improving their performance across all subjects. When comparing LLM-SR and SR-SCIENTIST without additional RL training, SR-SCIENTIST outperforms LLM-SR on $Acc_{0.01}$ and matches it on $Acc_{0.1}$ when using Qwen3-32B as the LLM backbone. However, it lags behind when using the Qwen3-8B and Qwen3-14B models. This highlights the importance of a model's intrinsic ability in a framework that provides the model with greater autonomy.

D. Supplementary Analysis and Evaluation Details

D.1. Analysis for the evaluation metrics

Table 7 presents the results measured by the R^2 metric. As shown, the differences between the various methods are small, making it difficult to distinguish their performance. Additionally, we present the NMSE results in Table 6. The average NMSE is sensitive to maximum values, which makes it an unreliable indicator of overall performance. Although quantitative methods could be employed, such as truncating errors that exceed a certain constant, this threshold is difficult to define. Using the median value is also insufficient, as it cannot account for the entire performance distribution. Therefore, we abandoned these metrics and instead used accuracy-to-tolerance as the main metric.

D.2. Cost

For LLM-based methods, the cost mainly comes from API calls to LLMs. In Table 8, we present the estimated cost of SR-SCIENTIST with GPT-OSS-120B and GPT-OSS-20B as the LLM backbones. We calculated the cost for each problem based on token consumption and common API pricing. As shown, the cost is acceptable for practical usage, and when considering cached tokens, the price can be further reduced. Additionally, we deployed GPT-OSS-120B on a local server with 2 NVIDIA H100s for batch-level inference on 129 problems and recorded the wall-clock times. The maximum time was no more than 5 hours, a duration that is acceptable for practical usage, as typical scenarios involve only a few problems.

D.3. OOD Performance of Other Models

We illustrate the OOD performance using other models as LLM backbones in Figures 9, 10, 11, and 12. For models including GLM-4.5-Air, GPT-OSS-120B, and GPT-OSS-20B, SR-SCIENTIST consistently outperforms the other methods in overall accuracy and most subjects. For Qwen3-Coder-30B, it slightly lags behind the method PySR on the OOD data. After RL training, it not only enhances its performance on ID data but also on OOD data, showing the generalization of the discovered equations.

D.4. Other Qualitative Results

To demonstrate how the agent gains insight from data analysis and long-horizon exploration, we provide two snippets of the agent trajectories in Figure 16 and Figure 17. As shown by the reasoning process in the dashed box, the agent forms its hypothesis for the equation based on its analysis of the data through code, along with the performance of historical equations. This analysis grounded in the data helps decrease the search space compared to previous methods. The reasoning and analysis process can also inspire human scientists towards better equation design.

D.5. Sensitivity Analysis of Hyperparameters

We present a sensitivity analysis of the hyperparameters in Table 9 and Table 10, covering the number of equations fetched from the experience buffer and the initial MAPE goal. Regarding the number of fetched equations, our method remains robust when this value is relatively small. However, performance declines slightly when it is increased to 10. This suggests that including a large number of fetched equations in the context window may strain the model's capability. Regarding the initial MAPE goal, performance is insensitive to relatively easy targets, as our dynamic goal update mechanism effectively prevents trivial goals. However, an extremely challenging goal (e.g., 0.0001) leads to degraded performance, causing the model to become trapped during the optimization process.

D.6. Symbolic Accuracy Calculation

We calculate symbolic accuracy using a two-step evaluation strategy: an initial assessment by an LLM, followed by

Method	Overall		Material Science		Chemistry		Biology		Physics	
	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
Qwen3-32B										
LLM-SR	30.49	9.04	34.67	18.67	38.89	5.56	15.28	4.17	29.55	9.09
SR-SCIENTIST	29.46	15.50	72.00	48.00	22.22	8.33	20.83	4.17	15.91	9.09
+RL	39.53	20.16	80.00	64.00	38.89	8.33	25.00	4.17	25.00	13.64
Qwen3-14B										
LLM-SR	27.91	9.30	56.00	32.00	16.67	2.78	8.33	0.00	31.82	6.82
SR-SCIENTIST	18.60	6.98	48.00	28.00	13.89	0.00	12.50	4.17	9.09	2.27
+ RL	23.26	10.08	52.00	40.00	16.67	2.78	20.83	0.00	13.64	4.55
Qwen3-8B										
LLM-SR	22.48	8.53	48.00	28.00	16.67	5.56	8.33	0.00	20.45	4.55
SR-SCIENTIST	15.50	4.65	40.00	16.00	8.33	5.56	16.67	0.00	6.82	0.00
+ RL	20.93	9.30	68.00	44.00	16.67	0.00	4.17	0.00	6.82	2.27

Table 5. Performance comparison of different LLM backbones for RL training, including Qwen3-8B, Qwen3-14B, and Qwen3-32B.

Method	NMSE (Average)	NMSE (Max)	NMSE (Min)	NMSE (Medium)
LaSR	1.53e-02	1.67e-01	2.66e-13	1.84e-03
LLM-SR	3.19e-03	7.83e-02	1.72e-14	8.87e-06
SR-SCIENTIST	1.29e-03	4.06e-02	1.57e-14	2.64e-06

Table 6. Comparison of SR-SCIENTIST and LLM-based SR baseline methods measured by NMSE. We report the result of Qwen3-Coder-480B on the physics subject.

Method	Overall (R^2)	Material Science (R^2)	Chemistry (R^2)	Biology (R^2)	Physics (R^2)
LaSR	0.991	0.998	0.996	0.988	0.985
LLM-SR	0.998	0.996	1.000	1.000	0.997
SR-SCIENTIST	0.998	0.996	1.000	0.995	0.999

Table 7. Comparison of SR-SCIENTIST and LLM-based SR baseline methods measured by R^2. We report the result of Qwen3-Coder-480B.

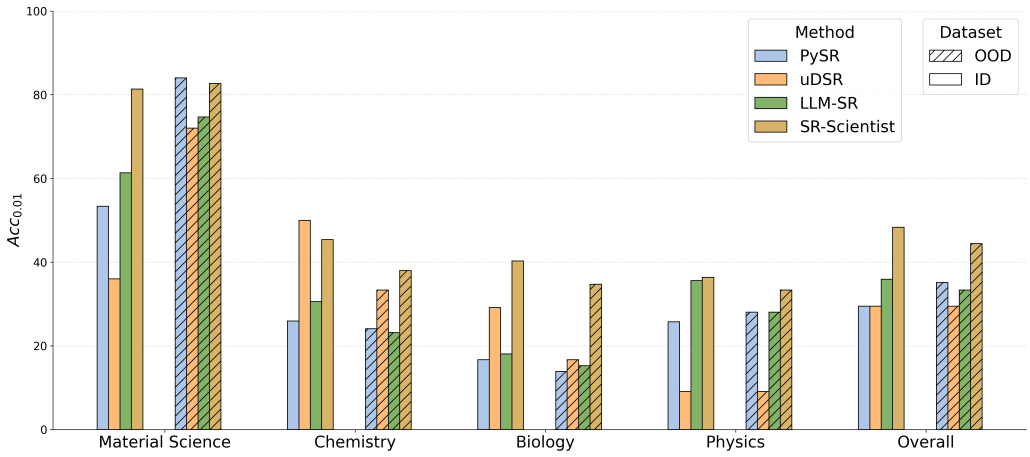

Figure 9. Detailed results of in-domain (ID) and out-of-domain (OOD) performance using $\text{Acc}_{0.01}$ across various LSR-Synth scientific domains. (with GLM-4.5-Air as LLM backbone)

human verification. Specifically, we instruct GPT-OSS-120B to determine the equivalence between a predicted equation and its ground truth counterpart using the prompt shown in Figure 14. To ensure reliability, the LLM evaluates each problem 10 times. Our preliminary studies on 121 cases reveal a 98.3% consistency rate between LLM and human evaluations for instances where the 10 voting results were unanimous. Therefore, our practical workflow involves

Method	API Price	API Cost Per Problem
SR-SCIENTIST (GPT-OSS-120B)	Input: $0.05/1M; Output: $0.25/1M	$0.25
SR-SCIENTIST (GPT-OSS-20B)	Input: $0.03/1M; Output: $0.15/1M	$0.1

Table 8. The estimated cost of SR-SCIENTIST. This calculation does not consider cached tokens; including them would reduce the cost further.

Table 9. Sensitivity analysis of the number of equations K fetched from the experience buffer. (with GPT-OSS-120B as LLM backbone)

K	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
1	63.57	49.61
3	63.57	49.35
5	63.31	50.90
10	61.50	47.29

Table 10. Sensitivity analysis of the initial MAPE goal s_{goal}. (with GPT-OSS-120B as LLM backbone)

s_{goal}	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
0.1	64.34	48.84
0.01	63.31	49.61
0.001	63.57	49.35
0.0001	51.74	37.70

first using the LLM for evaluation. However, for any case where the LLM's votes are inconsistent, we pass the problem to human evaluators for a final decision.

D.7. Tool Call Analysis

We instruct GPT-OSS-120B to perform tool call analysis using the prompt in Figure 15. We check 54 cases and find a 94.4 % consistency between human and LLM-based evaluation.

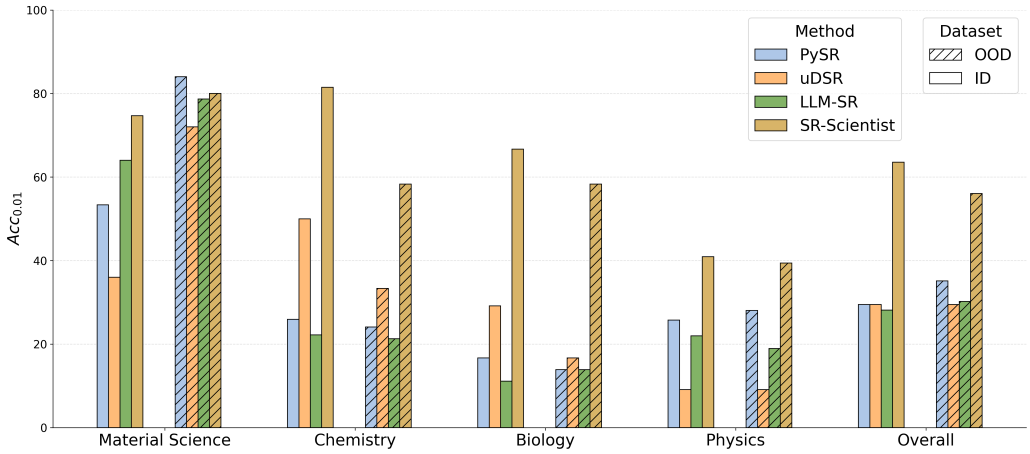

Figure 10. Detailed results of in-domain (ID) and out-of-domain (OOD) performance using $Acc_{0.01}$ across various LSR-Synth scientific domains. (with GPT-OSS-120B as LLM backbone)

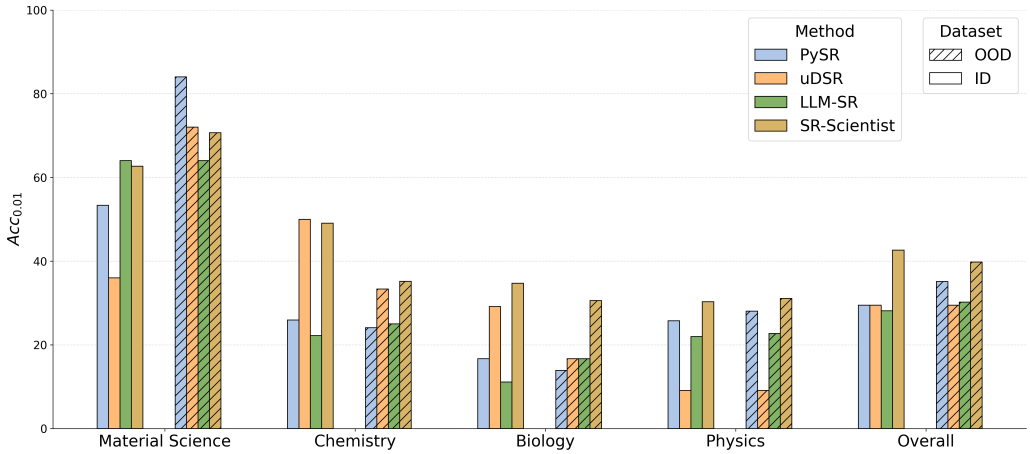

Figure 11. Detailed results of in-domain (ID) and out-of-domain (OOD) performance using $Acc_{0.01}$ across various LSR-Synth scientific domains. (with GPT-OSS-20B as LLM backbone)

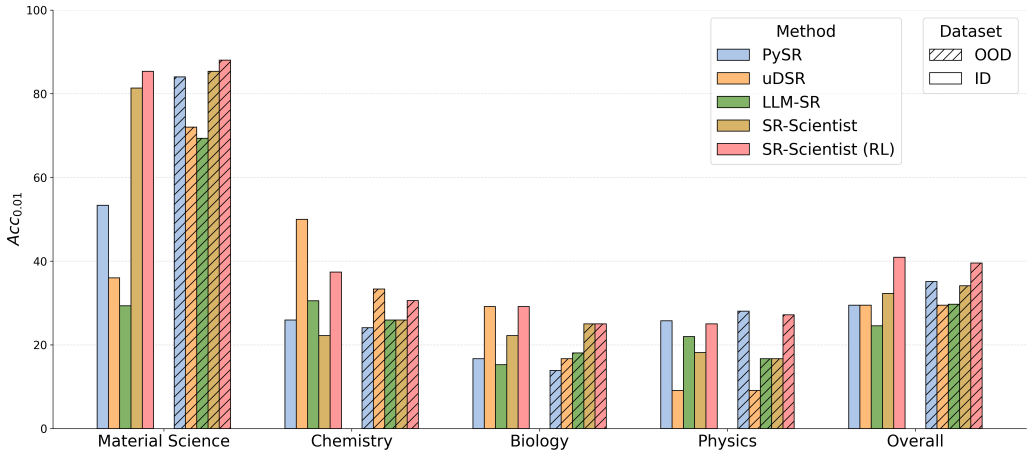

Figure 12. Detailed results of in-domain (ID) and out-of-domain (OOD) performance using $Acc_{0.01}$ across various LSR-Synth scientific domains. (with Qwen3-Coder-30B as LLM backbone)

```
Find the mathematical function skeleton that represents Acceleration in Nonl-linear Harmonic
Oscillator, given data on Position at time t, Time, and Velocity at time t.


Follow these steps to solve the problem:

**1. Implement the Equation in Code**

* Based on your knowledge and analysis, identify the standard equation and implement it in the code.
* Your equation will likely have one or more constants. Use elements from the `params` list
(e.g., `params[0]`, `params[1]`, `params[2]`) to represent these constants, as the `equation_evaluator`
tool is designed to optimize them. Note that the `params` list has a fixed size of 10
(`MAX_NPARAMS = 10`), so you can use up to 10 parameters in your model.

**2. Test, Analyze, and Refine**
    * Evaluate the equation's goodness of fit using the `equation_evaluator` tool.
    You need to pass the entire function, including the function header, to the tool. Here is an example:
```python
def equation(x: np.ndarray, t: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
""" Mathematical function for Acceleration in Nonl-linear Harmonic Oscillator
    Args:
        x: A numpy array representing observations of Position at time t.
        t: A numpy array representing observations of Time.
        v: A numpy array representing observations of Velocity at time t.
        params: Array of numeric constants or parameters to be optimized
    Return:
        A numpy array representing Acceleration in Nonl-linear Harmonic Oscillator as the result
        of applying the mathematical function to the inputs.
"""
    output = params[0] * x + params[1] * t + params[2] * v + params[3]
    return output
```
    You can modify the function body, but the function header must remain unchanged.
    * Your goal is to reduce the mean absolute percentage error to less than 0.1000%.
    Meeting this condition indicates that your equation is a good fit for the data.
    * If this goal is not met, refine your equation in Python and observe its performance.
    You can write your own data exploration snippet and use the `data_analyzer` tool to execute it,
    allowing you to inspect the data for potential relationships or anomalies.
3. Submit Your Final Answer
    * Once you are confident your equation has met the condition, or if you conclude after numerous
    attempts that you cannot meet it, provide the completed Python function as your answer.
"""
```

Figure 13. User prompt for the agent.

```
Given the ground truth mathematical expression A and the hypothesis B, determine if there exist any
constant parameter values that would make the hypothesis equivalent to the given ground truth expression.
Let's think step by step. Explain your reasoning and then provide the final answer as:
{{ "reasoning": "Step-by-step analysis", "answer": "Yes/No" }}

Ground Truth A: {gt_equation}
Hypothesis B: {pred_equation}
```

Figure 14. Prompt for symbolic assessment.

```
You are an expert code analyst. Your task is to classify Python code snippets written by an AI agent that
is trying to discover scientific equations from data.
Analyze the primary purpose of the code snippet and assign it a category based on the definitions below.
Category Definitions:
- Data Statistics: The code calculates a statistical property of the data. Examples include
calculating the mean, variance, correlation, minimum, or maximum values.
  - IMPORTANT: For this category, you MUST respond in the format: Data Statistics: [STATISTIC_NAME].
    For example: Data Statistics: Correlation or Data Statistics: Mean.
- Data Print: The code prints raw data samples for the purpose of initial inspection and understanding.
A common example is printing the first few rows.
- Residual Error Analysis: The code evaluates a proposed mathematical equation by calculating the
residuals (the difference between predicted and actual values) or other error metrics to determine its
goodness of fit to the data.
If the code's purpose does not fit any of the categories listed above, create a new, concise, and
specific category name.
For all categories except 'Data Statistics', respond with ONLY the category name.
Classify the following Python code snippet.
Code Snippet:
---
{code}
---
```

Figure 15. Prompt for tool call analysis.

```
Agent: Now I'll analyze the data further to understand the relationship better. Let me look at how the
       growth rate changes with respect to population and time.
       ```
       import json
       import sys
       import numpy as np

       # load the data
       input_data_str = sys.stdin.read()
       data_list = json.loads(input_data_str)

       # Convert to numpy arrays
       data_array = np.array(data_list)
       outputs = data_array[:, 0] # Population growth rate
       t_values = data_array[:, 1] # Time
       P_values = data_array[:, 2] # Population

       # Look at the pattern when population is high (near carrying capacity)
       high_pop_mask = P_values > 90
       high_pop_data = data_array[high_pop_mask]
       print(f"Number of high population (>90) data points: {len(high_pop_data)}")
       if len(high_pop_data) > 0:
           high_pop_growth_rates = high_pop_data[:, 0]
           print(f"High population growth rates: min={np.min(high_pop_growth_rates):.6f},
           max={np.max(high_pop_growth_rates):.6f}, mean={np.mean(high_pop_growth_rates):.6f}")

       # Look at the pattern when population is low
       low_pop_mask = P_values < 20
       low_pop_data = data_array[low_pop_mask]
       print(f"Number of low population (<20) data points: {len(low_pop_data)}")
       if len(low_pop_data) > 0:
           low_pop_growth_rates = low_pop_data[:, 0]
           print(f"Low population growth rates: min={np.min(low_pop_growth_rates):.6f},
           max={np.max(low_pop_growth_rates):.6f}, mean={np.mean(low_pop_growth_rates):.6f}")

       # Look at early time vs late time patterns
       early_time_mask = t_values < 10
       late_time_mask = t_values > 40
       early_data = data_array[early_time_mask]
       late_data = data_array[late_time_mask]

       print(f"Early time growth rates: min={np.min(early_data[:, 0]):.6f},
       max={np.max(early_data[:, 0]):.6f}, mean={np.mean(early_data[:, 0]):.6f}")
       print(f"Late time growth rates: min={np.min(late_data[:, 0]):.6f},
       max={np.max(late_data[:, 0]):.6f}, mean={np.mean(late_data[:, 0]):.6f}")

       # Check if there's a quadratic relationship
       P_squared = P_values ** 2
       correlation_p2_o = np.corrcoef(P_squared, outputs)[0, 1]
       print(f"Correlation between P^2 and Growth Rate: {correlation_p2_o:.6f}")
       ```

Tool:   Number of high population (>90) data points: 3468
       High population growth rates: min=-0.051324, max=5.084093, mean=0.118920
       Number of low population (<20) data points: 234
       Low population growth rates: min=1.016704, max=16.429510, mean=6.147035
       Early time growth rates: min=0.792945, max=24.778742, mean=10.340174
       Late time growth rates: min=-0.048202, max=0.077139, mean=0.008463
       Correlation between P^2 and Growth Rate: -0.704877
-----------------------------------------------------------------------------------------------------
Agent: Based on my analysis, I can see that this appears to be a population growth model with
      characteristics of logistic growth: High population values (>90) have very low growth rates,
      approaching zero. Low population values (<20) have high growth rates. Early times have higher
      growth rates than late times. There's a strong negative correlation between P^2 and growth rate.
      This suggests a logistic growth model where the growth rate decreases as population approaches a
      carrying capacity. Let me try a more refined logistic model:
-----------------------------------------------------------------------------------------------------
       def equation(t: np.ndarray, P: np.ndarray, params: np.ndarray) -> np.ndarray:
           r, K, alpha, beta = params[0], params[1], params[2], params[3]

           # Ensure parameters stay in reasonable ranges
           K = np.abs(K) + 90.0  # Near maximum observed population
           alpha = np.abs(alpha)  # Ensure positive decay rate

           logistic_term = r * P * (1 - P / K)
           time_decay = np.exp(-alpha * t)
           output = logistic_term * time_decay + beta

           return output
```

Figure 16. The snippet of the agent trajectory for the BPG18 biology problem. (with Qwen3-Coder-480B as LLM backbone)

```
Agent: We need to improve model. Let's inspect data to see relationships. Use data_analyzer to view
       stats.

       ```
       import json, sys, numpy as np
       data_str = sys.stdin.read()
       data = json.loads(data_str)
       # Convert to arrays
       outputs = np.array([row[0] for row in data])
       inputs = np.array([row[1:] for row in data])
       X = inputs.T
       x, t, v = X
       print('samples', len(outputs))
       print('output stats: mean', outputs.mean(), 'std', outputs.std())
       print('x stats: mean', x.mean(), 'std', x.std())
       print('t stats: mean', t.mean(), 'std', t.std())
       print('v stats: mean', v.mean(), 'std', v.std())
       # compute correlation
       corr = np.corrcoef(np.vstack([outputs, x, t, v]))
       print('Correlation matrix:\n', corr)
       # maybe try simple linear regression coefficients via least squares to see which terms matter
       A = np.column_stack([x, t, v, np.ones_like(x)])
       coef, *_ = np.linalg.lstsq(A, outputs, rcond=None)
       print('Linear coeffs (x,t,v,const):', coef)
       ```

Tool: samples 4000
       output stats: mean -0.13570954609178262 std 7.8811164966115
       x stats: mean -0.04767970374011202 std 0.7673236558313943
       t stats: mean 18.040196053244408 std 10.417019343399922
       v stats: mean -0.024004985556995963 std 2.4674150781331674
       Correlation matrix:
       [[1. -0.99249334 -0.01201174 0.00565316][-0.99249334 1. 0.01074399 -0.00662076]
       [-0.01201174 0.01074399 1. 0.01175548] [0.00565316 -0.00662076 0.01175548 1.]]
       Linear coeffs (x,t,v,const): [-1.01937290e+01 -1.01219659e-03 -2.88143544e-03 -6.03552468e-01]

Agent: The data analysis shows high correlation: output(acceleration) strongly negatively correlated
       with x (position) -0.9925 correlation. t correlation also -0.9925? Actually correlation matrix
       shows output vs t correlation -0.9925 (since second row/column is t). Wait matrix: rows order:
       outputs, x, t, v? They stacked [outputs, x, t, v]. So correlation[0,2] is output vs t = -0.9925.
       Also output vs x = -0.9925? Actually correlation[0,1] is -0.9925 as well. So both x and t highly
       correlated with output, likely because time and position are linearly related (maybe x
       ~ something of t). The simple linear regression gave coeffs: x: -10.1937, t: -0.001012,
       v: -0.00288, const: -0.6035. The magnitude of x coefficient is large. Maybe output is
       approximated by -10 * x + small other terms. The correlation is near -1, suggests a linear
       relationship with x (and t maybe similar). So maybe acceleration = -k * x (like simple harmonic
       oscillator). For a non-linear oscillator, could have term -kx - alphax^3 but correlation near -1
       suggests linear dominates. Our previous simple cubic model gave huge MAPE, because we didn't
       include intercept and maybe scaling. Let's try a linear model: a = p0 * x + p1 (constant).
       Use params[0] * x + params[1]. That may achieve low error.
       Let's test.
----
       import numpy as np
       def equation(x: np.ndarray, t: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
           """Simple linear model: a = p0 * x + p1"""
           return params[0] * x + params[1]
```

Figure 17. The snippet of the agent trajectory for the PO29 physics problem. (with GPT-OSS-120B as LLM backbone)

```
Your goal is to determine the correct equation, implement it as a Python function, and optimize it until
the mean absolute percentage error is less than 0.1%.
You should use the `equation_evaluator` tool to evaluate the equation's goodness of fit and the
`data_analyzer` tool to write code for data analysis.
For the `equation_evaluator`, it is a code interpreter that wraps your function with the following code:
```python
import numpy as np
import sys
import json
# Initialize parameters
MAX_NPARAMS = 10
params = [1.0] * MAX_NPARAMS
# Example of a user-provided equation
def equation(x: np.ndarray, t: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
    """
    Mathematical function for Acceleration in Nonl-linear Harmonic Oscillator
    Args:
    x: A numpy array representing observations of Position at time t.
    t: A numpy array representing observations of Time.
    v: A numpy array representing observations of Velocity at time t.
    params: Array of numeric constants or parameters to be optimized
    Return:
        A numpy array representing Acceleration in Nonl-linear Harmonic Oscillator as the result of
        applying the mathematical function to the inputs.
    """
    output = params[0] * x + params[1] * t + params[2] * v + params[3]
    return output
def evaluate(data: list) -> float:
    # Load data observations
    outputs = np.array([row[0] for row in data])
    inputs = np.array([row[1:] for row in data])
    X = inputs
    # Optimize parameters based on data
    from scipy.optimize import minimize
    def loss(params):
        y_pred = equation(*X.T, params)
        return np.mean((y_pred - outputs) ** 2)
    loss_partial = lambda p: loss(p)
    result = minimize(loss_partial, [1.0] * MAX_NPARAMS, method='BFGS')
    # Return evaluation score
    optimized_params = result.x
    mse = result.fun
    if np.isnan(mse) or np.isinf(mse):
        return None, None, None
    var_outputs = np.var(outputs)
    if np.isclose(var_outputs, 0):
        nmse = 0.0 if np.isclose(mse, 0) else np.inf
    else:
        nmse = mse / var_outputs
    y_pred = equation(*X.T, optimized_params)
    zero_mask = np.isclose(outputs, 0)
    non_zero_mask = ~zero_mask
    mape = 0.0
    if np.any(non_zero_mask):
        relative_errors=np.abs((y_pred[non_zero_mask] - outputs[non_zero_mask])/outputs[non_zero_mask])
        mape = np.mean(relative_errors)
    return float(mse), float(nmse), float(mape)
if __name__ == '__main__':
    input_data_str = sys.stdin.read()
    data_list = json.loads(input_data_str)
    mse, nmse, mape = evaluate(data_list)
    if mse is not None:
        print(f"MSE:{{mse:.6e}};NMSE:{{nmse:.6e}};Mean absolute percentage error:{{mape:.4\%}}")
        if mape < {mape_threshold}:
            print("Success: The mean absolute percentage error is smaller than 0.1%.")
        else:
            print("Failure: The mean absolute percentage error is larger than 0.1%.")
''''
As shown, the `equation_evaluator` tool assesses your equation's goodness of fit. It uses SciPy's BFGS
optimizer to find the optimal parameters for your equation based on the dataset. It then provides an
output with performance metrics (Mean Squared Error, Normalized Mean Squared Error, and Mean
Absolute Percentage Error), the success status, and details of any bugs.
In utilizing the tool, you only need to pass the entire function, including the function header, to the
tool.
For the `data_analyzer` tool, it is a code interpreter that can run your data exploration snippet.
You can access the data as shown in the example.
However, you are forbidden from using any libraries like Matplotlib for plotting figures for analysis.
```python
import json
import sys
# load the data
input_data_str = sys.stdin.read()
data_list = json.loads(input_data_str)
# print the first 5 entries
# In each entry of data_list, the first value is the output to predict, and the rest are the inputs.
print(data_list[:5])
"""
```

Figure 18. System prompt for the agent.