

LLM-SR: Scientific Equation Discovery via Programming with Large Language Models

Parshin Shojaee^{1*} Kazem Meidani^{2*} Shashank Gupta³
 Amir Barati Farimani² Chandan K. Reddy¹

¹Virginia Tech ²Carnegie Mellon University ³Allen Institute for AI

Abstract

Mathematical equations have been unreasonably effective in describing complex natural phenomena across various scientific disciplines. However, discovering such insightful equations from data presents significant challenges due to the necessity of navigating extremely high-dimensional combinatorial and nonlinear hypothesis spaces. Traditional methods of equation discovery, commonly known as symbolic regression, largely focus on extracting equations from data alone, often neglecting the rich domain-specific prior knowledge that scientists typically depend on. To bridge this gap, we introduce LLM-SR, a novel approach that leverages the extensive scientific knowledge and robust code generation capabilities of Large Language Models (LLMs) to discover scientific equations from data in an efficient manner. Specifically, LLM-SR treats equations as programs with mathematical operators and combines LLMs' scientific priors with evolutionary search over equation programs. The LLM iteratively proposes new equation skeleton hypotheses, drawing from its physical understanding, which are then optimized against data to estimate skeleton parameters. We demonstrate LLM-SR's effectiveness across three diverse scientific domains, where it discovers physically accurate equations that provide significantly better fits to in-domain and out-of-domain data compared to the well-established symbolic regression baselines. Incorporating scientific prior knowledge also enables LLM-SR to search the equation space more efficiently than baselines¹.

1 Introduction

The emergence of Large Language Models (LLMs) has marked a significant milestone in artificial intelligence, showcasing remarkable capabilities across various domains (Achiam et al., 2023). As LLMs continue to evolve, researchers are exploring innovative ways to harness their potential for solving complex problems such as scientific discovery (Wang et al., 2023; AI4Science & Quantum, 2023). Their ability to process and comprehend vast amounts of scientific literature, extract relevant information, and generate coherent hypotheses has recently opened up new avenues for accelerating scientific progress (Zheng et al., 2023b). Additionally, by leveraging their ability to understand and reason with the help of programming and execution, LLMs have shown the potential to enhance automatic reasoning and problem-solving capabilities (Romera-Paredes et al., 2024; Madaan et al., 2024). Motivated by these strengths, LLMs could be particularly helpful for the task of equation discovery, a fundamental task in science and scientific discovery.

Discovering accurate symbolic mathematical models from data is an important task in various scientific and engineering disciplines. The task of data-driven *equation discovery* (also commonly known as Symbolic Regression (SR)), aims to find abstract mathematical equations from data observations such that these equations are predictive of the underlying

*Equal contribution. Contact: parshinshojaee@vt.edu, mmeidani@andrew.cmu.edu

¹Code is available at: <https://github.com/deep-symbolic-mathematics/LLM-SR>

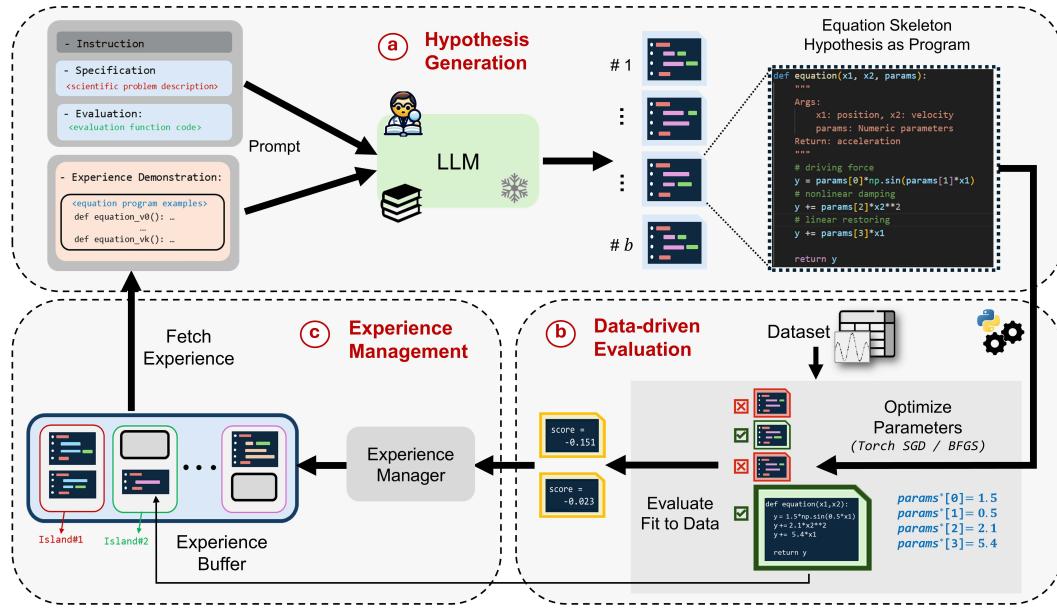


Figure 1: **The LLM-SR framework**, consisting of three main steps: (a) **Hypothesis Generation**, where LLM generates equation program skeletons based on a structured prompt; (b) **Data-driven Evaluation**, which optimizes the parameters of each equation skeleton hypothesis and assesses its fit to the data; and (c) **Experience Management**, which maintains a diverse buffer of high-scoring hypotheses to provide informative in-context examples into LLM’s prompt for iterative refinement.

data, are interpretable, and generalize to unseen data. Finding such equations offers several advantages over simply estimating a predictive model, as the resulting mathematical functions provide insights into the underlying physical processes, enable extrapolation beyond the observed data, and facilitate knowledge transfer across related problems (Schmidt & Lipson, 2009). However, while evaluating the fit of a proposed equation is relatively straightforward, the inverse process of obtaining these mathematical equations from data is a challenging problem, known to be NP-hard (Virgolin & Pissis, 2022). Current equation discovery methods encompass a wide variety of approaches from evolutionary search algorithms (Cranmer, 2023; Mundhenk et al., 2021; La Cava et al., 2021) to advanced deep learning methods using Transformers (Biggio et al., 2021; Kamienny et al., 2022). Most of the traditional symbolic regression techniques are built on top of Genetic Programming (GP) (Koza, 1994) evolutionary methods, representing mathematical equations as expression trees and searching the combinatorial space of possible equations through iterative mutation and recombination. However, current methods often struggle with the complexity of the vast optimization space and do not incorporate prior scientific knowledge, which leads to suboptimal solutions and inefficient exploration of the equation search space. Thus, there is a need for equation discovery methods that effectively integrate prior scientific knowledge into the navigation of equation search space, a strategy akin to a scientist’s reliance on foundational knowledge when formulating hypotheses for scientific discovery.

To address these limitations, we introduce LLM-SR (Fig. 1), a novel framework that combines the strengths of LLMs, reliable optimizers, and evolutionary search for data-driven equation discovery. At its core, LLM-SR is an iterative hypotheses refinement method that iteratively generates hypotheses, evaluates these hypotheses, and uses the evaluation signal to refine and search for better hypotheses. Specifically, given an initial equation hypothesis for the underlying data, LLM-SR utilizes LLMs to propose new equation hypotheses (1(a)), evaluates their fit on the data using off-the-shelf optimizers (1(b)), and uses this data-driven feedback and a carefully maintained dynamic memory of previous equations (1(c)) to iteratively guide the search towards better equations. LLM-SR leverages the scientific knowledge embedded in LLMs using short descriptions of the problem and the variables involved in a given system to generate educated hypotheses for equation skeletons (i.e.,

equation forms with placeholder parameters for numeric coefficients and constants). The LLM’s adaptive and informed mutation and crossover capabilities (Meyerson et al., 2023) are then employed to refine the suggested equations in an iterative process. LLM-SR incorporates data-driven feedback into the search process by evaluating the fitness of the generated equations against the observed data, guiding the search towards more accurate equation skeletons. By representing equations as programs, we take advantage of LLM’s ability to generate structured and executable code (Li et al., 2023; Shojaee et al., 2023) while providing a flexible and effective way to represent general mathematical relations. This representation also facilitates direct and differentiable parameter optimization to better optimize the coefficients or constants in the generated equations. We evaluated LLM-SR using GPT-3.5-turbo and Mixtral-8x7B-Instruct (Jiang et al., 2024) as backbone LLMs across four equation discovery problems spanning three scientific domains. LLM-SR consistently outperforms state-of-the-art symbolic regression methods, discovering physically accurate equations with better fit and generalization in both in-domain (ID) and out-of-domain (OOD) test settings. By leveraging the scientific prior knowledge, LLM-SR explores the equation search space more efficiently, requiring fewer iterations to find accurate equations. Our analysis also highlights the crucial role of iterative refinement and evolutionary search in LLM-SR’s superior performance. The major contributions of this work can be summarized as follows:

- We introduce LLM-SR, a novel method that combines the strengths of LLMs, off-the-shelf optimizers, and evolutionary search for data-driven equation discovery.
- We show that LLM-SR outperforms state-of-the-art SR methods by navigating the equation search space more efficiently and discovering more accurate equations with better out-of-domain generalization across 4 problems from 3 scientific domains.
- We demonstrate through a comprehensive ablation study that providing natural language descriptions of the problem and variables, data-driven feedback, and skeleton parameter optimization signals are vital for LLM-SR’s success.

2 LLM-SR Methodology

2.1 Problem Formulation

In the task of data-driven equation discovery, also known as symbolic regression (SR), the goal is to find a concise and interpretable symbolic mathematical expression \tilde{f} that closely approximates an unknown underlying equation $f : \mathbb{R}^d \rightarrow \mathbb{R}$, which maps a d -dimensional input vector \mathbf{x} to output y . Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ consisting of input-output pairs, SR methods seek to uncover the hidden mathematical relationship, such that $\tilde{f}(\mathbf{x}_i) \approx y_i$ for all i . The discovered equation should not only accurately fit the observed data points but also exhibit strong generalization capabilities to unseen data while maintaining interpretability.

Current SR methods typically represent equations using techniques such as expression trees (Cranmer, 2023), prefix sequences (Petersen et al., 2021; Biggio et al., 2021), or context-free grammars (Brence et al., 2021), constructing a search space \mathcal{F} . These representations provide limited and structured search spaces, enabling evolutionary search approaches like genetic programming (GP) to explore and find candidate expressions. In contrast, our work employs program functions to directly map inputs \mathbf{x} to output targets y : `def f(x) : ... return y`. This offers a flexible and expressive way to represent equations, allowing for a diverse set of mathematical operations and step-by-step instructions to capture complex mathematical relations. However, the space of possible programs can be vast, with extremely sparse solutions, which can hinder the effectiveness of traditional search methods like GP in finding the desired equation programs (Devlin et al., 2017; Parisotto et al., 2017). To tackle this challenge, we leverage the scientific knowledge and coding capabilities of LLMs to effectively navigate the program space for equation discovery. LLMs have the ability to generate syntactically correct and knowledge-guided code snippets, guiding the efficient exploration of the large equation program landscape.

Let π_θ denote a pre-trained LLM with parameters θ . In this task, we iteratively sample a set of equation program skeletons from LLM, $\mathcal{F} = \{f : f \sim \pi_\theta\}$. The goal is to find the

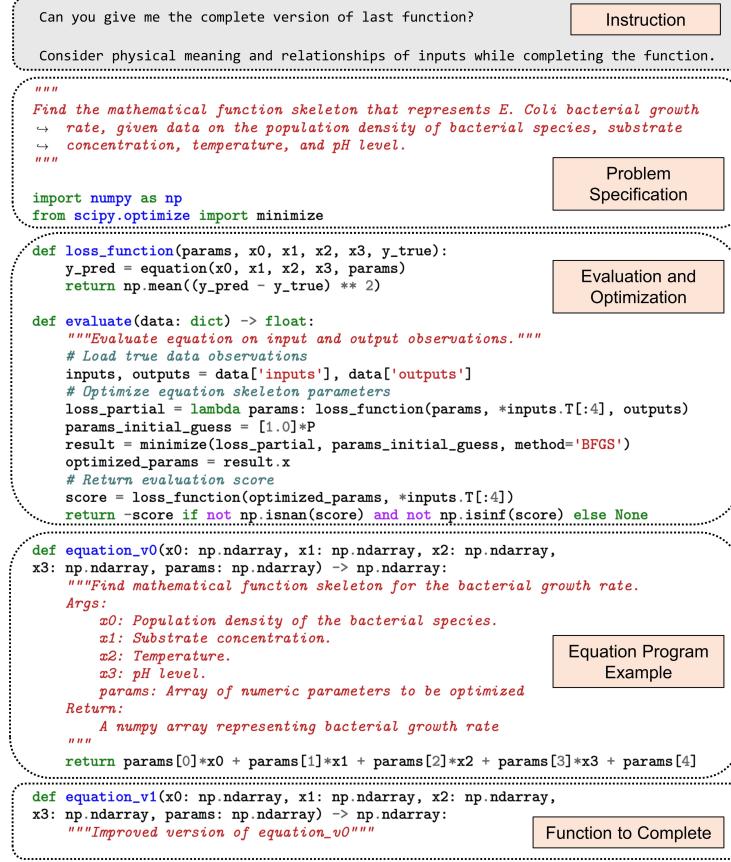


Figure 2: Example of an input prompt for E. Coli bacterial growth with problem specifications, evaluation and optimization function, and the initial input equation example.

skeleton that maximizes the reward (evaluation score) $\text{Score}_{\mathcal{T}}(f, \mathcal{D})$ for a given scientific problem \mathcal{T} and data observations \mathcal{D} . The optimization problem can be expressed as: $f^* = \arg \max_f \mathbb{E}_{f \in \mathcal{F}} [\text{Score}_{\mathcal{T}}(f, \mathcal{D})]$, where f^* denotes the optimal equation program discovered at the end of the LLM-guided search. LLM-SR prompts LLM to propose hypotheses given the problem specification, and experience demonstrations containing previously discovered promising equations. In LLM-SR, LLM only generates hypotheses as equation program skeletons where the coefficients or constants of each equation are considered as a placeholder parameter vector that can be optimized. The parameters are optimized using robust off-the-shelf optimizers and evaluated for their fit. Promising hypotheses are then added to a dynamic experience buffer, which guides the equation refinement process, while non-promising hypotheses are discarded. Below we explain the key components of this framework, shown in Fig. 1, in more detail.

2.2 Hypothesis Generation

As shown in Fig. 1 (a), the hypothesis generation step relies on a pre-trained LLM to propose diverse and promising equation program skeletons. By leveraging the LLM’s extensive knowledge and generative capabilities, we aim to efficiently explore the vast space of potential equations while maintaining syntactic correctness and scientific plausibility.

2.2.1 Prompt

To guide the LLM in generating relevant and well-structured equation program skeletons, we design prompts that provide essential context and specify the desired output format. Our prompt structure, shown in Fig. 2, consist of the following components:

Instruction. We begin the prompt with a clear instruction to the LLM, requesting the completion of the last function’s body in the provided code. The instruction emphasizes the importance of considering the physical meaning and relationships of the input variables while completing the function. This guidance helps steer the LLM’s output towards scientifically meaningful and coherent suggestions.

Problem Specification. Following the instruction, we include a concise description of the scientific problem at hand. This specification outlines the key variables, constraints, and objectives of the problem, providing the necessary context for the LLM to generate relevant equation program skeletons. By clearly defining the problem space, we enable the LLM to focus its efforts on the most pertinent aspects of the task.

Evaluation Function. To ensure that the generated equation program skeletons can be effectively assessed, we include the evaluation function in the prompt. This function includes the criteria used to measure the quality and fitness of the proposed equations. By providing the evaluation function upfront, we enable the LLM to generate skeletons that are aligned with the desired performance objectives.

Experience Demonstration. To further guide the LLM and provide concrete examples of equation program skeletons as well as their improvement trajectory, we incorporate experience demonstrations in the form of in-context examples (Fig. 2 only shows the initial example). These examples serve as a reference for the LLM to build upon and refine in its generated outputs.

2.2.2 Hypothesis Sampling

At each iteration t , we sample a batch of b equation skeletons $\mathcal{F}_t = \{f_i\}_{i=1}^b$ from the LLM π_θ , where each equation skeleton f_i is generated based on the constructed prompt p_t : $f_i \sim \pi_\theta(\cdot | p_t)$. To encourage diversity in the generated equation program skeletons, we utilize stochastic temperature-based sampling, which allows for a balance between exploration (creativity) and exploitation (prior knowledge) in the hypothesis space.

After obtaining the sampled equation programs and before processing to the data-driven evaluation step, we execute each sampled equation program f_i and discard those that fail to execute successfully. By discarding such equation programs, we ensure that only valid and executable hypotheses are considered for further evaluation. Also, to maintain computational efficiency and prevent excessively time-consuming evaluations, we impose a maximum execution time threshold for each equation program. If the execution time of an equation program exceeds this threshold, we discard the corresponding hypothesis.

2.3 Data-driven Evaluation

Following the hypothesis generation step, we evaluate and score the generated equation skeleton hypotheses using observed data. In this step, demonstrated in Fig. 1 (b), we first optimize the parameters of each hypothesis with regard to the dataset, and then use the optimized parameters to evaluate and score each hypothesis.

2.3.1 Hypothesis Optimization

Inspired by (Biggio et al., 2021), in LLM-SR, we decouple the equation discovery process into two steps: (*i*) discovering the equation program structures (skeletons) using the LLM, and (*ii*) optimizing the skeleton parameters/coefficients based on data. The LLM is responsible for generating equation skeletons and the core logic of the program, while the numeric values of the parameters are represented as placeholders in the form of a parameter vector `params` (as shown in Fig. 2). These placeholders are subsequently optimized to fit the observed data. Let \mathcal{F}_t denote the set of equation program skeletons generated by the LLM at iteration t . Each equation program skeleton $f \in \mathcal{F}_t$ is a function that takes both input features x and a vector of learnable parameters `params` as input arguments and returns the predicted target values: “`def f(x, params): ... return y`”.

To optimize the parameters for each skeleton hypothesis, we use two approaches: (1) `numpy+BFBS` and (2) `pytorch+Adam`. The BFBS algorithm (Fletcher, 1987), available in the `scipy` library (as represented in Fig. 2), is a gradient-free nonlinear optimization method.

In contrast, the Adam optimizer (Kingma & Ba, 2014), available in `pytorch`, is a direct gradient-based optimization algorithm. The program notion of equation skeletons employed in LLM-SR framework enables the use of differentiable programming techniques, which is not possible with other representations common in equation discovery such as expression trees or prefix notations. The choice of the optimization approach also depends on the specific characteristics of the problem and the equation skeleton. The `numpy+BFGS` method is often preferred when the number of parameters is relatively small, while `pytorch+Adam` is more suitable for larger-scale problems and complex equation structures that benefit from efficient gradient computation through differentiable programming.

2.3.2 Fitness Assessment

After finding the optimized values of the skeleton parameters, we assess the fitness of each equation program hypothesis by measuring its ability to capture the underlying patterns in the data. To obtain the predicted target values \hat{y} , we replace the placeholder parameter vector `params` in the equation program skeleton f with the optimized parameter values `params*` and evaluate the equation skeleton function over the input data: $\hat{y} = f(\mathbf{x}, \text{params}^*)$. We then compute the negative Mean Squared Error (MSE) between the predicted and true target values as the fitness evaluation score: $s = \text{Score}_{\mathcal{T}}(f, \mathcal{D}) = -\text{MSE}(\hat{y}, \mathbf{y})$, where s represents the evaluation score or reward corresponding to the equation skeleton f .

2.4 Experience Management

To better navigate the search landscape and avoid local minima, LLM-SR contains an experience management step, shown in Fig. 1 (c), which maintains a diverse population of high-quality equation programs in the experience buffer and effectively samples from this population to construct informative prompts for the LLM in next iteration. This step consists of two key components: a) experience manager for maintaining the experience buffer, and b) sampling from this experience buffer.

2.4.1 Buffer Maintenance

Following the data-driven evaluation, the pairs of equation skeleton hypotheses and their corresponding scores, denoted as (f, s) , are stored in an experience buffer \mathcal{P}_t for further refinement in subsequent iterations of the search process. To avoid local optima, we follow (Cranmer, 2023; Romera-Paredes et al., 2024) and adopt an islands model, also known as a multi-population model for managing the experience buffer. We initialize m islands with a copy of the equation program example provided in the initial prompt (`equation_v0` in Fig. 2). These islands evolve independently, allowing for parallel exploration of different regions in the equation program space. At each iteration t , the new equation program hypotheses \mathcal{F}_t , along with their scores, $\{(f, s) : f \in \mathcal{F}_t, s = -\text{Score}_{\mathcal{T}}(f, \mathcal{D})\}$, are added to the same island from which the in-context examples of prompts were sampled, if the new score s is better than the current best. Within each island, we further cluster the equation programs based on their signature, which is defined as their score. Equation programs with identical signatures are grouped together. This clustering approach helps preserve diversity by ensuring that equation programs with different performance characteristics are maintained in the population.

2.4.2 Experience Sampling

To construct informative prompts for the LLM, we sample equation programs from the experience buffer, updating the prompt with new experience demonstration in-context examples. As per (Romera-Paredes et al., 2024), this process follows a two-stage sampling method: (i) selecting a random island from the m available ones, and (ii) sampling k equation programs from the selected island for inclusion as in-context examples in the prompt. Cluster selection follows a Boltzmann sampling (Maza & Tidor, 1993), with a score-based probability of choosing cluster i : $P_i = \frac{\exp(s_i/\tau_c)}{\sum_{i'} \exp(s'_i/\tau_c)}$, where s_i denote mean score of the i -th cluster, and τ_c is a temperature parameter. After cluster selection, individual equation programs are sampled, favoring shorter programs, using the probability proportional to $\exp(-\tilde{l}_i/\tau_p)$, where \tilde{l}_i is the normalized program length, and τ_p is a temperature parameter. More details of these sampling steps are provided in App. B. The sampled programs are then included in

Algorithm 1: LLM-SR

```
Input : LLM  $\pi_\theta$ , dataset  $\mathcal{D}$ , scientific problem  $\mathcal{T}$ ,  
        $T$  iterations,  $k$  in-context examples,  
        $b$  samples per prompt

# Initialize population
 $\mathcal{P}_0 \leftarrow \text{InitPop}()$ 
 $f^*, s^* \leftarrow \text{null}, -\infty$ 
for  $t \leftarrow 1$  to  $T - 1$  do
    # Sample from LLM
     $\mathcal{F}_t \leftarrow \text{SampleLLM}(\pi_\theta, \mathcal{P}_{t-1}, k, b)$ 
    # Evaluation and population update
    for  $f \in \mathcal{F}_t$  do
         $s \leftarrow \text{Score}_{\mathcal{T}}(f, \mathcal{D})$ 
        if  $s > s^*$  then
             $f^*, s^* \leftarrow f, s$ 
             $\mathcal{P}_t \leftarrow \mathcal{P}_{t-1} \cup \{(f, s)\}$ 
        end
    end
end

Output :  $f^*, s^*$ 
```

Algorithm 2: SampleLLM

```
Input : LLM  $\pi_\theta$ , experience buffer  $\mathcal{P}$ ,  
        $k$  in-context examples,  
        $b$  samples per prompt

# Sample examples from buffer
 $E \leftarrow \{x_j\}_{j=1}^k, x_j = \text{SampleDB}(\mathcal{P})$ 
# Prompt with new examples
 $p \leftarrow \text{MakeFewShotPrompt}(E)$ 
# Sample from LLM
 $\mathcal{F} \leftarrow \{f_j\}_{j=1}^b, f_j \sim \pi_\theta(\cdot | p)$ 

Output :  $\mathcal{F}$ 
```

the prompt as in-context experience demonstration, providing the LLM with relevant and diverse examples to guide the generation of new equation program hypotheses.

Algorithm 1 presents a summarized pseudo-code of the LLM-SR framework. We first initialize the experience buffer population \mathcal{P}_0 using `InitPop` with simple linear initial equation skeleton example (as shown in Fig. 2 for *E. coli* growth problem). This equation skeleton serves as a starting point template for generating candidate equations, avoiding the need for manual prior knowledge incorporation. At each iteration t , `SampleLLM` (detailed in Algorithm 2), which represents Hypothesis Generation, samples b equation program skeletons from the LLM using an updated prompt with k in-context examples from \mathcal{P}_{t-1} . The sampled equations are evaluated using $\text{Score}_{\mathcal{T}}(f, \mathcal{D})$, and \mathcal{P}_t is updated with the equation-score pairs (f, s) . In Algorithm 2, `SampleDB` function which represents Experience Sampling is first called to sample in-context examples from the experience buffer and update prompt accordingly. Then, LLM samples are collected with the updated prompt. After T iteration, the best-scoring program f^* from \mathcal{P}_t and its corresponding score s^* are returned as the optimal solution found for the problem. LLM-SR iteratively refines equation skeletons using the LLM’s generative capabilities, guiding the search towards promising structures based on the evolving experience buffer population.

3 Experimental Setup

3.1 Benchmarks and Datasets

Current symbolic regression benchmarks (La Cava et al., 2021; Strogatz, 2015; Udrescu & Tegmark, 2020), mostly based on Feynman physics equations (Udrescu et al., 2020), may not effectively capture the true discovery process and reasoning capabilities of LLMs. Our findings show that LLM-SR rapidly achieves low Normalized MSE scores within very few iterations on Feynman problems, suggesting that LLMs have likely memorized these commonly known equations due to their prevalence in training data (check App. C for full results). Feynman equations are often commonly known, and easily memorized by LLMs, making them unsuitable evaluation benchmarks for this study. To overcome this limitation, we introduce novel benchmark problems across three diverse scientific domains that are designed to challenge the model’s ability to uncover complex mathematical relations while leveraging its scientific prior knowledge. These new benchmark problems are designed to simulate the conditions for scientific discovery, ensuring that the equations cannot be trivially obtained through memorization of the language model. By focusing on diverse scientific domains and introducing custom modifications to known equations, we encourage LLM-SR to creatively explore the equation search space and identify mathematical relations. We next discuss these new benchmark problems.

Nonlinear Oscillators. Nonlinear damped oscillators are ubiquitous in physics and engineering. The dynamics of oscillators are governed by differential equations that describe the relationship between the oscillator’s position, velocity, and the forces acting upon it. Specifically, the oscillator’s motion is given by a second-order differential equation: $\ddot{x} + f(t, x, \dot{x}) = 0$, involving time (t), position (x), and nonlinear function $f(t, x, \dot{x})$ accounts for forces. In this study, we explore two custom nonlinear forms: Oscillation 1: $\dot{v} = F \sin(\omega x) - \alpha v^3 - \beta x^3 - \gamma x \cdot v - x \cdot \cos(x)$; and Oscillation 2: $\dot{v} = F \sin(\omega t) - \alpha v^3 - \beta x \cdot v - \delta x \cdot \exp(\gamma x)$, with $v = \dot{x}$ as velocity and $\omega, \alpha, \beta, \delta, \gamma$ as constants. These forms are designed to challenge the model’s ability to uncover complex equations, avoiding simple memorization of commonly known oscillators. More details on the design and data generation of these oscillators are provided in App. D.1.

Bacterial Growth. The growth of Escherichia coli (E. coli) bacteria has been widely studied in microbiology due to its importance in various applications, such as biotechnology, and food safety (Monod, 1949; Rosso et al., 1995). Discovering equations governing E. coli growth rate under different conditions is crucial for predicting and optimizing bacterial growth (Tuttle et al., 2021). The bacterial population growth rate can be modeled using a differential equation with the effects of population density (B), substrate concentration (S), temperature (T), and pH level, which is commonly formulated as $\frac{dB}{dt} = f(B, S, T, pH) = f_B(B) \cdot f_S(S) \cdot f_T(T) \cdot f_{pH}(pH)$. To invoke the LLM’s prior knowledge about this problem, yet avoid memorization, we consider nonlinear arbitrary designs for $f_T(T)$ and $f_{pH}(pH)$ that share similar characteristics with the common models but are not trivial to recover from memory. More details on the specific differential equation employed and data for this problem are provided in App. D.2.

Material Stress Behavior. Understanding the stress-strain behavior of materials under various conditions, particularly the relationship between stress, strain, and temperature, is crucial for designing and analyzing structures in various engineering fields. In this problem, we use a real-world dataset from (Aakash et al., 2019), containing tensile tests on aluminum at six temperatures ranging from 20°C to 300°C. The data covers the elastic, plastic, and failure regions of the stress-strain curves, providing valuable insights into the material’s behavior. More details on the data and visualization of these stress-strain curves can be found in App. D.3.

3.2 Baselines

We compare LLM-SR against several state-of-the-art symbolic regression (SR) baselines, including **GPyLearn**¹, Deep Symbolic Regression (**DSR**) (Petersen et al., 2021), Unified DSR (**uDSR**) (Landajuela et al., 2022), and **PySR** (Cranmer, 2023). GPyLearn is a pioneering Genetic Programming (GP) based SR approach with the open-source gplearn package. DSR is a deep learning-based SR approach that employs reinforcement learning over symbolic expression sequence generations. uDSR extends DSR by incorporating large-scale pretraining with Transformers and neural-guided GP search at the decoding stage. PySR² is a recent SR method that uses multi-island asynchronous GP-based evolution with the aim of scientific equation discovery. For a fair comparison with LLM-SR, we allow all baselines to run for over 2M iterations until convergence to their best performance. More details on the implementation and parameter settings of each baseline are provided in App. A.

3.3 LLM-SR Configuration

We experiment with **GPT-3.5-turbo** and **Mixtral-8x7B-Instruct** as 2 different LLM backbones for LLM-SR. At each iteration, we sample $b = 4$ equation skeletons per prompt from the LLM using a temperature $\tau = 0.8$, optimize equation skeleton parameters via BFGS (30s timeout per program), and sample $k = 2$ in-context examples from the experience buffer for the refinement. We run LLM-SR variants for 2.5K iterations in all experiments. For more implementation details (incl. the experience buffer structure, prompt refinement strategy, and parallel evaluation), please refer to App. A.

¹<https://gplearn.readthedocs.io/en/stable/>

²<https://github.com/MilesCranmer/PySR>

Model	Oscillation 1		Oscillation 2		E. coli growth		Stress-Strain		All	
	ID↓	OOD↓	ID↓	OOD↓	ID↓	OOD↓	ID↓	OOD↓	ID↓	OOD↓
Glearn	0.0155	0.5567	0.7551	3.188	1.081	1.039	0.1063	0.4091	0.4894	1.298
DSR (Petersen et al., 2021)	0.0087	0.2454	0.0580	0.1945	0.9451	2.4291	0.3326	1.108	0.3361	0.9942
uDsr (Landajuela et al., 2022)	0.0003	0.0007	0.0032	0.0015	0.3322	5.4584	0.0502	0.1761	0.0964	1.409
PySR (Cranmer, 2023)	0.0009	0.3106	0.0002	0.0098	0.0376	1.0141	0.0331	0.1304	0.0179	0.3662
LLM-SR (Mixtral)	7.89e-8	0.0002	0.0030	0.0291	0.0026	0.0037	0.0162	0.0946	0.0054	0.0319
LLM-SR (GPT-3.5)	4.65e-7	0.0005	2.12e-7	3.81e-5	0.0214	0.0264	0.0210	0.0516	0.0106	0.0196

Table 1: Quantitative performance comparison of LLM-SR (with GPT-3.5 and Mixtral backbones), and SR baseline models on different scientific benchmark problems measured by Normalized Mean Squared Error. ‘All’ column indicates average scores over all datasets. SR baselines ran for over 2M iterations, while LLM-SR variants ran for around 2K iterations.

4 Findings

4.1 LLM-SR Discovers more Accurate Equations

Table 1 shows the performance comparison of LLM-SR (using GPT-3.5 and Mixtral backbones) against state-of-the-art symbolic regression methods on various scientific benchmark problems. In these experiments, symbolic regression methods were allowed to run for considerably longer iterations (over 2M) while LLM-SR variants ran for around 2K iterations. The table reports the Normalized Mean Squared Error (NMSE), where lower values indicate better performance. Under the in-domain (ID) test setting, where the test set is sampled from the same domain as the data used for equation discovery, LLM-SR variants significantly outperform leading symbolic regression baselines across all benchmarks, despite running for far fewer iterations. Among the symbolic regression baselines, PySR and uDSR exhibit superior ID performance compared to DSR and Glearn.

4.2 LLM-SR has better OOD generalization

The ability to generate equations that generalize well to domains beyond the training data, i.e., out-of-domain (OOD) data, is a crucial characteristic of effective equation discovery methods. To assess the generalization capability of the predicted equations, we evaluate LLM-SR and the symbolic regression baselines in the OOD test setting, focusing on their ability to predict systems outside the region encountered during training. Table 1 reveals that the performance gap between LLM-SR and the baselines is more pronounced in the OOD test setting. This observation suggests that equations discovered by LLM-SR exhibit superior generalization capabilities, likely attributable to the scientific prior knowledge embedded by LLMs in the equation discovery process. For instance, on the E. coli growth problem, LLM-SR achieves an OOD NMSE of around 0.0037, considerably better than symbolic regression methods which all obtain OOD NMSE > 1 . Fig. 3 also compares the ground truth distribution of E. coli growth problem with the predicted distributions obtained from LLM-SR, PySR, and uDSR. The shaded region and black points indicate in-domain (ID) data, while the rest represent out-of-domain (OOD). Results show that the distributions obtained from LLM-SR align well with the

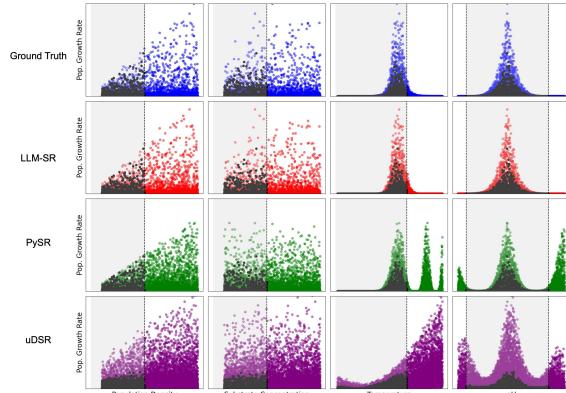


Figure 3: Comparison of E. coli growth rate distributions from LLM-SR, PySR, and uDSR. LLM-SR aligns well with the ground truth, even for OOD data (unshaded regions), demonstrating better generalization than PySR and uDSR, which overfit the ID data (shaded regions and black points).

ground truth, not only for ID data but also for OOD regions. This alignment demonstrates the better generalizability of equations discovered by LLM-SR to unseen data, likely due to the integration of scientific prior knowledge in the equation discovery process. In contrast, leading symbolic regression methods like PySR and uDSR tend to overfit the observed data, with their predicted distributions deviating significantly from the true distributions in OOD regions. This overfitting behavior highlights their limited ability to generalize beyond the training data and capture the true physical underlying patterns. For more detailed results and analyses for other benchmark problems, check App. E.

4.3 LLM-SR Discovers Equations More Efficiently

Fig. 4 shows the performance trajectories of LLM-SR variants and symbolic regression baselines across different scientific benchmark problems, depicting the maximum fitting scores achieved over search iterations. By leveraging scientific prior knowledge, LLM-SR needs to explore a considerably lower number of equation candidates in the vast optimization space compared to symbolic regression baselines that lack this knowledge. This is evident from the sharp drops in the error curves for LLM-SR variants, indicating they efficiently navigate the search space by exploiting domain knowledge to identify promising candidates more quickly. In contrast, the symbolic regression baselines show much more gradual improvements and fail to match LLM-SR’s performance even after running for over 2M iterations, as shown by their final results in Fig. 4. The performance gap between LLM-SR and symbolic regression baselines also widens as iterations increase, particularly in the Oscillation 2 and E. coli growth problems, highlighting the effectiveness of LLM-SR’s iterative refinement process.

5 Analysis

5.1 Ablation Study

Fig. 5 presents an ablation study on the Oscillation 2 problem with GPT-3.5 backbone to investigate the impact of different components in the LLM-SR performance.

Problem Specification. We create a “*w/o Problem Specification*” variant by removing the natural language description of the problem and variables of the system from the LLM’s input prompt. In this setup, the model operates solely as an adaptive evolutionary search agent without domain-specific information about the variables and the problem being studied. We find that this variant performs worse, with increases in the Normalized MSE from 2.12e-7 to 4.65e-5 for in-domain data and from 3.81e-5 to 7.10e-3 for OOD data. This highlights the significance of incorporating prior domain knowledge into the equation discovery process.

Iterative Refinement. The “*w/o Iterative Refinement*” variant is equivalent to the LLM sampling baseline, which generates more than 2K samples for the initial LLM-SR’s prompt without computing feedback and updating in-context examples over the sampling span. The absence of iterative refinement leads to a substantial performance drop, with the Normalized MSE increasing to 1.01e-1 for in-domain data and 1.81e-1 for OOD data. This emphasizes

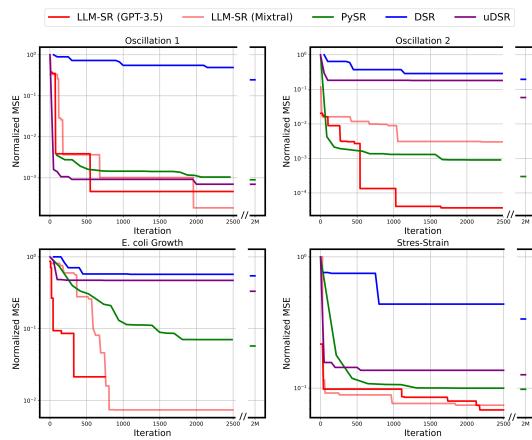


Figure 4: **Best score trajectories** of LLM-SR with GPT-3.5 and Mixtral against SR baselines across different benchmark problems. LLM-SR discovers accurate equations more efficiently, requiring fewer iterations to outperform baselines, which fail to match LLM-SR even after 2M iterations.

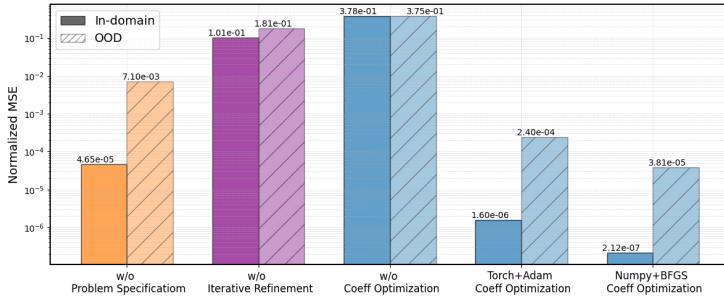


Figure 5: **Ablation results on the Oscillation 2 problem**, showing the impact of problem specification, iterative refinement, parameter optimization, and optimization methods on LLM-SR’s performance.

the importance of the evolutionary search and iterative refinement process in LLM-SR’s success.

Coefficient Optimization. We also create a “*w/o Coeff Optimization*” variant that requires the LLM to generate complete equations, including numerical coefficients and constants, in a single end-to-end step. This helps us study the role of parameter optimization using external optimizers in the equation discovery process. We find that this variant shows significantly worse results, with the Normalized MSE increasing to $3.75e-1$ for in-domain data and $3.78e-1$ for OOD data. These results indicate that the two-stage approach of generating equation skeletons followed by data-driven skeleton parameter optimization is essential for effective performance, as it helps navigate the intricate combinatorial optimization landscape involving both continuous parameter values and discrete equation structures.

Optimization method. LLM-SR relies on direct and differentiable parameter optimization, a capability not present in current symbolic regression methods. We compare two different optimization frameworks: *numpy*+BFGS and *torch*+Adam. The *numpy*+BFGS variant yielded better-performing equations on both ID ($2.12e-7$) and OOD ($3.81e-5$) evaluations compared to *torch*+Adam (ID: $1.60e-6$, OOD: $2.4e-4$). However, our initial analysis indicated that this discrepancy could be primarily due to the LLM’s higher proficiency in generating *numpy* code compared to *pytorch*, rather than the superiority of one optimization method over the other. Combining LLM-SR with LLM backbones that are better in generating *pytorch* code could potentially leverage differentiable parameter optimization to achieve better performance in future works.

5.2 Qualitative Analysis

Fig. 6 presents the final discovered equations for both Oscillation problems using LLM-SR and other symbolic regression baselines. A notable observation is that equations discovered by LLM-SR have better recovered the symbolic terms of the true equations compared to baseline models. Moreover, LLM-SR provides explanations and reasoning steps based on the scientific knowledge about the problem, leading to more interpretable terms combined as the final equation. For example, in both problems, LLM-SR identifies the equation structure as a combination of driving force, damping force, and restoring force terms, relating them to the problem’s physical characteristics. In contrast, baselines (DSR, uDSR, PySR) generate equations lacking interpretability and understanding of the physical meanings or relations behind the terms. The equations appear as combination of mathematical operations and variables without clear connection to the problem’s underlying principles. More detailed results and qualitative analysis on the discovered equations in other benchmark problems and across performance progression curves can also be found in App. E.

6 Related Work

LLMs and Optimization. LLMs have demonstrated remarkable capabilities in various domains, but they are prone to generating incorrect or inconsistent outputs due to either

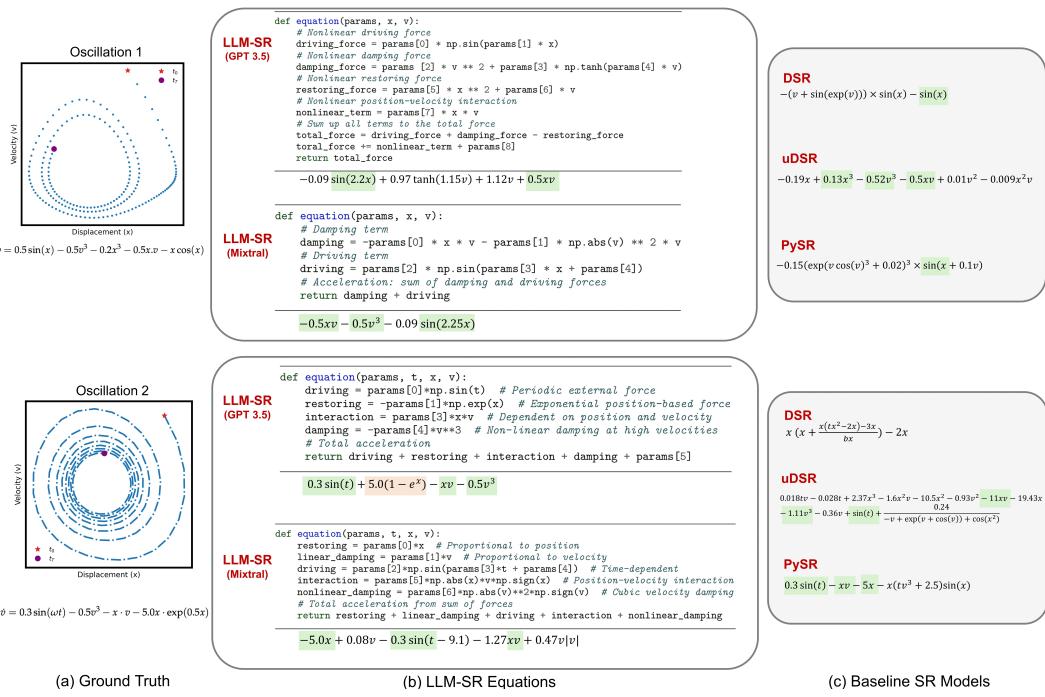


Figure 6: Discovered equations for Oscillation 1 (top) and Oscillation 2 (bottom) problems. (a) True equations and their phase diagram; (b) Equation program skeletons identified by LLM-SR, with simplified forms obtained after parameter optimization; and (c) Equations found using DSR, uDSR and PySR baselines. Shaded green terms denote recovered symbolic terms from true equations.

having flawed implicit knowledge or lacking access to external facts (Madaan et al., 2024; Zhu et al., 2023). This limitation, often referred to as hallucination, hinders the direct application of LLMs in tasks that require high accuracy. To mitigate these issues, researchers have explored combining LLMs with feedback or verification mechanisms, such as using the same model for feedback and self-refinement or employing external evaluators (Madaan et al., 2024; Yang et al., 2023b; Haluptzok et al., 2022). Recent works have also coupled LLMs with evaluators in an iterative optimization loop, where LLMs act as evolutionary search agents (Lehman et al., 2023; Liu et al., 2023; Wu et al., 2024; Lange et al., 2024). LLMs leverage their prior knowledge, reasoning, and generation capabilities to perform adaptive mutation and crossover operations (Meyerson et al., 2023). This integration has shown promising results in applications like prompt optimization (Yang et al., 2023a; Guo et al., 2024), neural architecture search (Chen et al., 2023; Zheng et al., 2023a), and heuristic discovery (Romera-Paredes et al., 2024). Most related to our work is FunSearch (Romera-Paredes et al., 2024) that combines LLMs with systematic evaluators to search for programs that push the boundaries in solving some established open mathematical problems. Building upon these ideas, our LLM-SR framework employs LLM as an informed optimizer, leveraging its scientific prior knowledge and data-driven evaluators to discover mathematical equations underlying scientific observations.

LLMs for Scientific Discovery. The use of LLMs in scientific discovery has gained significant attention in recent years (Wang et al., 2023). Researchers have explored the potential of LLMs across various scientific domains, such as drug discovery, biology, and materials design (AI4Science & Quantum, 2023). Recently, the ability of LLMs to propose plausible scientific hypotheses by leveraging their existing knowledge and their potential for data-driven discovery has been a topic of growing interest (Majumder et al., 2024; Zheng et al., 2023b; Qi et al., 2023). Despite the increasing exploration of LLMs in scientific contexts and question answering, their potential for tasks such as equation discovery and symbolic regression remains largely unexplored. Our work addresses this gap by introducing a novel approach that harnesses the scientific prior knowledge within LLMs and integrates it with data-driven evaluation. This approach showcases the potential of LLMs in scientific

equation discovery, contributing to the rapidly growing body of research on the use of LLMs for scientific advancement.

Symbolic Regression. Symbolic regression (SR) methods can be broadly categorized into search-based approaches, learning-based models, and hybrid methods. Search-based approaches mainly explore the space of mathematical expressions using evolutionary algorithms or reinforcement learning (Schmidt & Lipson, 2009; Cranmer, 2023; Petersen et al., 2021; Sun et al., 2023). Learning-based models, on the other hand, leverage large-scale synthetic data and Transformers to learn the mapping between numeric input observations and output mathematical expressions (Biggio et al., 2021; Kamienny et al., 2022). Hybrid methods aim to combine the strengths of both approaches, guiding the search by employing neural priors to improve the expressiveness and efficiency of the discovery process (Shojaee et al., 2024; Udrescu & Tegmark, 2020; Mundhenk et al., 2021; Meidani et al., 2023). Despite the progress made by these approaches, they often face limitations such as the lack of scientific prior knowledge incorporation and the restricted expressiveness of traditional equation representations like expression trees. Some works have attempted to address these issues by incorporating prior physical knowledge such as dimension analysis (Udrescu & Tegmark, 2020; Tenachi et al., 2023; Meidani & Barati Farimani, 2023) or using declarative bias and structures with pre-defined grammars (Todorovski & Dzeroski, 1997; Todorovski & Džeroski, 2007). However, these methods do not leverage the power of LLMs for this task. Our work advances this research direction by utilizing LLMs to efficiently search the combinatorial optimization space of equation discovery and generate meaningful equation structures based on the embedded scientific prior knowledge.

7 Discussion and Conclusion

In this work, we introduced LLM-SR, a novel framework that leverages the power of Large Language Models (LLMs) for scientific equation discovery. We demonstrated the effectiveness of LLM-SR by evaluating it on diverse set of scientific problems, showcasing its ability to uncover physically meaningful and accurate equations. Despite the promising results, LLM-SR has limitations that should be acknowledged. The computational resources associated with loading LLMs and generating a large number of equation programs can be substantial, especially when dealing with complex problems or large-scale datasets. Additionally, LLM-SR’s reliance on the scientific prior knowledge embedded within the LLMs may also be limited, biased, or incorrect in certain domains, potentially affecting the quality of the discovered equations. However, we believe that the potential of using LLM-SR framework can go beyond what has been studied in this work, and that LLM-enabled scientific equation discovery can facilitate this task in various fields of science and engineering. Moreover, with the rapid pace of LLM improvements, the computational cost is expected to decrease, making it more accessible and efficient.

Future research efforts can focus on several key areas to further enhance the capabilities of LLM-SR. Exploring the integration of more powerful or domain-specific language models can potentially improve the quality and relevance of the generated equations. Incorporating retrieval-augmented learning techniques, where the LLMs are augmented with literature-driven scientific knowledge, can also provide additional context and guidance during the equation discovery process. Developing methods to verify and guide the model towards generating equations that are consistent with established scientific principles can help ensure the physical validity of the discovered equations. Another potential future work is the development of more comprehensive benchmarks for evaluating LLM-based equation discovery methods. These benchmarks should either include a potentially empirical equation discovery process or be carefully designed to simulate such a process. This approach ensures that the equations generated by the models are not simply memorized but are derived through a genuine discovery process. As LLMs continue to evolve and become more powerful, LLM-SR has the potential to become an indispensable tool for researchers and scientists, accelerating scientific discovery and innovation across various domains. With the development of better benchmarks and evaluation protocols, we can unlock the full potential of LLM-based equation discovery and pave the way for accelerating scientific advancements.

References

- B.S. Aakash, JohnPatrick Connors, and Michael D. Shields. Stress-strain data for aluminum 6061-t651 from 9 lots at 6 temperatures under uniaxial and plane strain tension. *Data in Brief*, 25:104085, 2019. ISSN 2352-3409. doi: <https://doi.org/10.1016/j.dib.2019.104085>.
- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Microsoft Research AI4Science and Microsoft Azure Quantum. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.
- Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 936–945. PMLR, 18–24 Jul 2021.
- Jure Brence, Ljupčo Todorovski, and Sašo Džeroski. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224:107077, 2021. ISSN 0950-7051. doi: <https://doi.org/10.1016/j.knosys.2021.107077>.
- Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 7787–7817. Curran Associates, Inc., 2023.
- Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.
- Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural program learning under noisy I/O. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 990–998. PMLR, 06–11 Aug 2017.
- Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.
- Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.
- Patrick Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language models can teach themselves to program better. *arXiv preprint arXiv:2207.14502*, 2022.
- Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- Gordon R. Johnson and William H. Cook. Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures. *Engineering Fracture Mechanics*, 21(1):31–48, 1985. ISSN 0013-7944. doi: [https://doi.org/10.1016/0013-7944\(85\)90052-9](https://doi.org/10.1016/0013-7944(85)90052-9).
- Pierre-Alexandre Kamienny, Stéphane d’Ascoli, Guillaume Lample, and Francois Charton. End-to-end symbolic regression with transformers. In *Advances in Neural Information Processing Systems*, 2022.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. Contemporary symbolic regression methods and their relative performance. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Mikel Landajuela, Chak Lee, Jiachen Yang, Ruben Glatt, Claudio P. Santiago, Ignacio Aravena, Terrell N. Mundhenk, Garrett Mulcahy, and Brenden K. Petersen. A unified framework for deep symbolic regression. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022.

Robert Tjarko Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. *arXiv preprint arXiv:2402.18381*, 2024.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366. Springer, 2023.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Cheng-hao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Sanchaita Hazra, Ashish Sabharwal, and Peter Clark. Data-driven discovery with large generative models. *arXiv preprint arXiv:2402.13610*, 2024.

Michael de la Maza and Bruce Tidor. An analysis of selection procedures with particular attention paid to proportional and boltzmann selection. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 124–131, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1558602992.

Kazem Meidani and Amir Barati Farimani. Identification of parametric dynamical systems using integer programming. *Expert Systems with Applications*, 219:119622, 2023. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2023.119622>.

Kazem Meidani, Parshin Shojaee, Chandan K Reddy, and Amir Barati Farimani. Snip: Bridging mathematical symbolic and numeric realms with unified pre-training. In *The Twelfth International Conference on Learning Representations*, 2023.

Elliot Meyerson, Mark J Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K Hoover, and Joel Lehman. Language model crossover: Variation through few-shot prompting. *arXiv preprint arXiv:2302.12170*, 2023.

Jacques Monod. The growth of bacterial cultures. *Annual Review of Microbiology*, 3(Volume 3, 1949):371–394, 1949. ISSN 1545-3251. doi: <https://doi.org/10.1146/annurev.mi.03.100149.002103>.

Terrell N. Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P. Santiago, Daniel faissol, and Brenden K. Petersen. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

-
- Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=rJ0JwFcex>.
- Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.
- Biqing Qi, Kaiyan Zhang, Haoxiang Li, Kai Tian, Sihang Zeng, Zhang-Ren Chen, and Bowen Zhou. Large language models are zero shot hypothesis proposers. *arXiv preprint arXiv:2311.05965*, 2023.
- Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, Jan 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6.
- L Rosso, J R Lobry, S Bajard, and J P Flandrois. Convenient model to describe the combined effects of temperature and ph on microbial growth. *Applied and environmental microbiology*, 61(2):610–616, 1995. ISSN 0099-2240. Journal Article.
- Dipti Samantaray, Sumanta Mandal, and A.K. Bhaduri. A comparative study on johnson cook, modified zerilli-armstrong and arrhenius-type constitutive models to predict elevated temperature flow behaviour in modified 9cr-1mo steel. *Computational Materials Science*, 47(2):568–576, 2009. ISSN 0927-0256. doi: <https://doi.org/10.1016/j.commatsci.2009.09.025>.
- Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science Advance*, 324(5923):81–85, 2009. ISSN 0036-8075. doi: 10.1126/science.1165893.
- Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. Execution-based code generation using deep reinforcement learning. *arXiv preprint arXiv:2301.13816*, 2023.
- Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. Transformer-based planning for symbolic regression. *Advances in Neural Information Processing Systems*, 36, 2024.
- Steven H Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press, 2 edition, 2015. doi: 10.1201/9780429492563.
- Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *The Eleventh International Conference on Learning Representations*, 2023.
- Wassim Tenachi, Rodrigo Ibata, and Foivos I Diakogiannis. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *The Astrophysical Journal*, 959(2):99, 2023.
- Ljupco Todorovski and Saso Dzeroski. Declarative bias in equation discovery. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pp. 376–384, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604863.
- Ljupčo Todorovski and Sašo Džeroski. *Integrating Domain Knowledge in Equation Discovery*, pp. 69–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-73920-3. doi: 10.1007/978-3-540-73920-3\4.
- Amie R. Tuttle, Nicholas D. Trahan, and Mike S. Son. Growth and maintenance of escherichia coli laboratory strains. *Current Protocols*, 1(1):e20, 2021. ISSN 2691-1299. doi: 10.1002/cpz1.20.
- 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.

Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 4860–4871. Curran Associates, Inc., 2020.

Marco Virgolin and Solon P Pissis. Symbolic regression is NP-hard. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, Anima Anandkumar, Karianne Bergen, Carla P. Gomes, Shirley Ho, Pushmeet Kohli, Joan Lasenby, Jure Leskovec, Tie-Yan Liu, Arjun Manrai, Debora Marks, Bharath Ramsundar, Le Song, Jimeng Sun, Jian Tang, Petar Veličković, Max Welling, Linfeng Zhang, Connor W. Coley, Yoshua Bengio, and Marinka Zitnik. Scientific discovery in the age of artificial intelligence. *Nature*, 620 (7972):47–60, Aug 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06221-2.

Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *arXiv preprint arXiv:2401.10034*, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023a.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*, 2023b.

Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. Can gpt-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970*, 2023a.

Yizhen Zheng, Huan Yee Koh, Jiaxin Ju, Anh TN Nguyen, Lauren T May, Geoffrey I Webb, and Shirui Pan. Large language models for scientific synthesis, inference and explanation. *arXiv preprint arXiv:2310.07984*, 2023b.

Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. Large language models can learn rules. *arXiv preprint arXiv:2310.07064*, 2023.

Appendix

A Implementation Details

A.1 SR Baselines

We compare LLM-SR against several state-of-the-art Symbolic Regression (SR) baselines, including **GPyLearn**, Deep Symbolic Regression (**DSR**) (Petersen et al., 2021), Unified Deep Symbolic Regression (**uDSR**) (Landajuela et al., 2022), and **PySR** (Cranmer, 2023). GPyLearn is a pioneering and standard GP-based SR approach that uses the open-source `gplearn`³ package with parameters population size: 1000, tournament size: 20, and maximum generations: 2M. DSR employs a RNN-based reinforcement learning search over symbolic expressions. uDSR extends DSR by incorporating large-scale pretraining and GP search at the decoding stage. Both DSR and uDSR approaches are implemented using `deep-symbolic-optimization (DSO)`⁴ package with the standard default parameters: learning rate of 0.0005, batch size of 500, and a maximum of 2M iterations. PySR is also used with the state-of-the-art open-source `pysr`⁵ package for SR that uses asynchronous GP-based evolution, implemented with a population size of 4,000, up to 100 sub-populations, and a maximum of 2M iterations. For a fair comparison, we allowed the SR baselines to run for over 2M iterations until convergence to their best performance, while LLM-SR variants ran for only around 2K iterations. Each of SR baseline models was subjected to a long execution time 5 times, each time running for over 2 million iterations. This extensive number of evaluations and search iterations ensures robust evaluation of the models' capability to converge towards optimal solutions and effectively explore the vast equation space of each problem.

A.2 LLM-SR

The LLM-SR framework was evaluated using two distinct backbone LLM models: `GPT-3.5-turbo` and `Mixtral-8x7B-Instruct`. For both variants, the framework underwent around 2.5K iterations, a comparatively lower number of search iterations compared to symbolic regression baselines (around 2M) due to the enhanced efficiency of this approach because of embedded scientific prior knowledge. The best results obtained from these iterations were then documented and reported. Key parameters of the LLM-SR framework include: **Sampling Per Prompt**: LLM-SR generates $b = 4$ samples with temperature $\tau = 0.8$ per prompt; **Program Database Structure**: The framework incorporates a program database consisting of $m = 10$ islands. This database structure facilitates diverse program generation and efficient sampling. **Prompt Augmentation**: To refine the prompt's effectiveness, it includes $k = 2$ best-shot in-context examples as experience demonstration sampled from the experience buffer. This strategy leverages the buffer's accumulated knowledge to enhance prompt relevance and model performance over evolutionary mutations and the search process. **Execution Timeout**: A critical operational parameter in our setup was the execution timeout set at 30 seconds. If the execution of a program generated by the LLM exceeded this limit, the process was halted, and the evaluation score of `None` was returned. This constraint ensured timely progress and resource efficiency in the evaluation process. **Parallel Evaluation**: In this framework, we deployed $e = 4$ evaluators to operate concurrently. This parallelization allowed for rapid and efficient assessment of the generated programs per prompt. As pointed out, for the LLM backbone models, a sampling temperature of $\tau = 0.8$ was utilized. This temperature setting was chosen to balance creativity (exploration) and adherence to the problem constraints and reliance on the prior knowledge (exploitation), based on preliminary experiments.

B Additional Details of LLM-SR method

Hypothesis Generation and Data-driven Evaluation Fig. 2 provided an example of specification for Bacterial growth problem. Here, Fig. 7 showcases illustrative examples of

³<https://gplearn.readthedocs.io/en/stable/>

⁴<https://github.com/dso-org/deep-symbolic-optimization>

⁵<https://github.com/MilesCranmer/PySR>

```

"""
Find the mathematical function skeleton that represents second state equation of a
-- custom nonlinear damped oscillator ( $\frac{dv}{dt}$ ) dynamical system with nonlinear
-- terms for driving force, restoring force, and damping force, given data on
-- time ( $t$ ), position ( $x$ ), and velocity ( $v$ ).
"""

import numpy as np
#Initialize parameters with 1.0
params = [1.0]*10

def loss_function(params, x0, x1, y_true):
    """Return mean squared error loss for fitting to data."""
    ...

def evaluate(data: dict) -> float:
    """Evaluate discovered mathematical function on true inputs and outputs
    -- observations."""
    ...

def equation(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray)
    -> np.ndarray:
    """Find mathematical function skeleton for the acceleration in a damped
    -- nonlinear oscillator system with driving force.
    Args:
        x0: A numpy array representing time.
        x1: A numpy array representing observations of current position.
        x2: A numpy array representing observations of velocity.
        params: Array of numeric constants or parameters to be optimized
    Returns:
        A numpy array with representing acceleration as the result of applying the
        -- mathematical function to the inputs.
    """
    dv = params[0] * x0 + params[1] * x1 + params[2] * x2 + params[3]
    return dv

```

(a) Nonlinear Oscillation

```

"""
Find the mathematical function skeleton that returns stress, given data on strain
-- and temperature in an Aluminium rod for both elastic and plastic regions.
"""

import numpy as np
#Initialize parameters with 1.0
params = [1.0]*10

def loss_function(params, x0, x1, y_true):
    """Return mean squared error loss for fitting to data."""
    ...

def evaluate(data: dict) -> float:
    """Evaluate discovered mathematical function on true inputs and outputs
    -- observations."""
    ...

def equation(x0: np.ndarray, x1: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Find mathematical function skeleton for the stress given strain and
    -- temperature for elastic and plastic regions.
    Args:
        x0: A numpy array representing observations of strain.
        x1: A numpy array representing observations of temperature (in Celcius).
        params: Array of numeric constants or parameters to be optimized
    Returns:
        A numpy array representing stress as the result of applying the
        -- mathematical function to the inputs.
    """
    return params[0] * x0 + params[1] * x1

```

(b) Material Behavior Analysis

Figure 7: Example of input prompts program body for (a) Oscillation and (b) Stress-Strain problems, with problem specification, and the initial input equation program example (set as simple linear equation skeleton). For better readability, the evaluation function details are not included in this figure.

prompts and specifications tailored for the Oscillation and Stress-Strain problems. These prompts contain descriptions of the problem and relevant variables, expressed in natural language. By providing this context, the language model can leverage its existing knowledge about the physical meaning and relations of variables to generate scientifically plausible hypotheses for new equation programs.

Fig. 8 also shows an example of prompt and specification for LLM-SR that prompts the model to generate differentiable equation programs in pytorch using tensor operations. The prompt suggests using differentiable operators and replacing non-differentiable components (e.g., if-else conditions) with smooth differentiable approximations.

During each prompting step, the language model generates $b = 4$ distinct equation program skeletons using a generation temperature of $\tau = 0.8$. These equation skeleton programs are concurrently evaluated to gather feedback. Evaluation is constrained by time and memory limits set at $T = 30$ seconds and $M = 2\text{GB}$, respectively. Equation programs that exceed these limits are disqualified and considered as discarded hypotheses by returning `None` scores.

Experience Buffer Management. The pairs of equation skeleton hypotheses and their corresponding scores, denoted as (f, s) , are then stored in an experience buffer \mathcal{P}_t for further refinement in subsequent iterations of the search process. To encourage diversity and avoid getting stuck in local optima, we follow (Cranmer, 2023) and adopt an islands model, also known as a multiple population or multiple-deme model for managing the experience buffer (Cranmer, 2023). We initialize m separate islands, each containing a copy of the equation program example provided in the initial prompt (equation_v0 in Fig. 2). These islands evolve independently, allowing for parallel exploration of different regions in the equation program space. At each iteration t , the newly generated equation program hypotheses \mathcal{F}_t and their corresponding fitness scores $\{(f, s) : f \in \mathcal{F}_t, s = \text{MSE}(f, \mathcal{D})\}$ are added to the same island from which the prompts were sampled, if the score of new hypothesis s is better than the current best-score in the island. To maintain the quality and diversity of the experience buffer, we periodically reset the worst-performing islands. Every T_{reset} iterations (e.g., every 4 hrs), we identify the $m/2$ islands whose best equation programs have the lowest fitness scores. All the equation programs in these islands are discarded, and each island is reinitialized with a single high-performing equation program, obtained by randomly selecting one of the surviving $m/2$ islands and copying its highest-scoring equation program (favoring older programs in case of ties). This reset mechanism allows the framework to discard stagnant or unproductive regions of the equation program

```

"""
Find mathematical function form that fits data. This function form can
only contain binary and unary mathematical operators that are differentiable.
"""

import torch
import torch.nn as nn
import torch.optim as optim

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        # Initialize model parameters
        self.params = torch.nn.ParameterList([torch.nn.Parameter(torch.tensor(1.0))
                                             for _ in range(P)])
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return
        # Foward pass the model
        equation(x, self.params)

def evaluate(data: dict) -> float:
    """
    Evaluate equation on input and output observations.
    """
    # Load true data observations
    inputs , outputs = data['inputs'], data['outputs']
    # Define model
    model = Model()
    # Define optimizer
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    # Optimize equation skeleton parameters
    model.train()
    num_iterations = 10000
    for i in range(num_iterations):
        # Zero the gradients
        optimizer.zero_grad()
        # Forward pass: compute predicted outputs by passing inputs to the model
        y_pred = model(inputs)
        # Compute the loss
        loss = torch.mean((y_pred - outputs) ** 2)
        # Backward pass: compute loss gradient with respect to parameters
        loss.backward()
        # Update parameters using optimizer
        optimizer.step()
    #Return evaluation score
    return -loss.item() if not (torch.isnan(loss) | torch.isinf(loss)).any() else
        None

def equation(x: torch.Tensor, params: torch.nn.ParameterList) -> torch.Tensor:
    """
    Args:
        x (torch.Tensor): Input data.
        params (torch.nn.ParameterList): List of model numeric constant parameters.
    Return:
        torch.Tensor: The result of applying the mathematical function to the x.
    """
    return params[0]*x + params[1]

```

Figure 8: An example of prompt structure, containing problem specification, evaluation and optimization function, and equation program with pytorch and Adam Optimizer

space and focus on more promising areas. Within each island, we further cluster the equation programs based on their signature, which is defined as the equation program score. Equation programs with identical signatures are grouped together, forming clusters within each island. This clustering approach helps preserve diversity by ensuring that equation programs with different performance characteristics are maintained in the population.

Experience Sampling. To construct informative prompts for the LLM, we sample equation programs from the experience buffer and update the prompt to include new experience demonstration in-context examples. Here, we use a two-stage sampling process. First, we

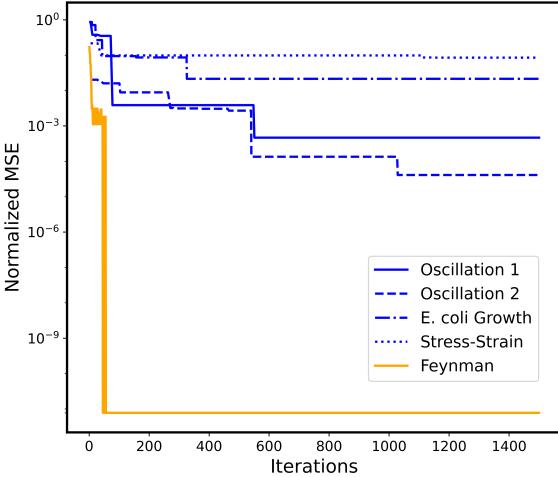


Figure 9: Trajectory of Normalized MSE score over iterations for LLM-SR (GPT-3.5) on Feynman benchmark equations versus new benchmark problems

randomly select an island from the m available islands. Then, within the selected island, we sample k equation programs (typically, $k = 2$) to be included as in-context examples in the prompt. When sampling equation programs within an island, we employ a two-step approach. First, we sample a cluster based on its evaluation score, favoring clusters with higher scores (i.e., higher-quality equation programs). Let s_i denote the score of the i -th cluster, defined as an aggregation (e.g., mean) of all the scores in the signature that characterizes that cluster. The probability P_i of choosing cluster i is given by:

$$P_i = \frac{\exp\left(\frac{s_i}{\tau_c}\right)}{\sum_{i'} \exp\left(\frac{s_{i'}}{\tau_c}\right)}, \quad \tau_c = T_0 \left(1 - \frac{u \bmod N}{N}\right)$$

, where τ_c is the temperature parameter, u is the current number of equation programs in the island, and T_0 and N are hyperparameters. This selection approach is known as the Boltzmann selection procedure (Maza & Tidor, 1993). Once a cluster is selected, we sample an equation program within that cluster, favoring shorter programs. Let l_i denote the negative length of the i -th program within the chosen cluster (measured as the number of characters), and let $\tilde{l}_i = \frac{l_i - \min_{i'} l_{i'}}{\max_{i'} l_{i'} + 10^{-6}}$. We set the probability of selecting each equation program proportional to $\exp(\tilde{l}_i / \tau_p)$, where τ_p is a temperature hyperparameter. The sampled programs are then included in the prompt as in-context experience demonstration, providing the LLM with relevant and diverse examples to guide the generation of new equation programs. By maintaining a diverse and high-quality population in the experience buffer and employing a strategic sampling approach, the experience management enables the LLM-SR framework to effectively explore the space of equation programs and iteratively refine its search based on the most promising candidates.

C Limitation of Feynman Benchmark Problems

The Feynman benchmark dataset (Udrescu & Tegmark, 2020), while commonly used for evaluating symbolic regression and equation discovery methods, may not effectively capture the true discovery process and reasoning capabilities of LLMs. Fig.9 compares the performance of LLM-SR with GPT-3.5 backbone across different benchmark problems, illustrating the best score trajectory of Normalized Mean Squared Error against the number of iterations for each problem. For the Feynman benchmark equations, LLM-SR rapidly achieves low MSE scores within very few iterations, suggesting that LLMs may have memorized or become overly familiar with these equations due to their prevalence in training data. Qualitative examples in Fig. 10, 11, and 12 further support this notion, as the LLM's

Input Prompt

```

Can you give me the complete version of last function? Instruction
Consider physical meaning and relationships of inputs while completing the function.

Find the mathematical function structure that represents the resultant intensity due
→ to the interference of two coherent light sources. Problem Specification

```

```

import numpy as np
from scipy.optimize import minimize

def loss_function(params, x0, x1, x2, y_true):
    y_pred = equation(x0, x1, x2, params)
    return np.mean((y_pred - y_true) ** 2)

def evaluate(data: dict) -> float:
    """Evaluate equation on input and output observations."""
    # Load true data observations
    inputs, outputs = data['inputs'], data['outputs']
    # Optimize equation skeleton parameters
    loss_partial = lambda params: loss_function(params, *inputs.T[:, 3], outputs)
    params_initial_guess = [1.0] * P
    result = minimize(loss_partial, params_initial_guess, method='BFGS')
    optimized_params = result.x
    # Return evaluation score
    score = loss_function(optimized_params, *inputs.T[:, 3])
    return -score if not np.isnan(score) and not np.isinf(score) else None

```

```

def equation_v0(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) ->
    np.ndarray:
    """Find mathematical function for the resultant intensity due to interference of
    → two waves given their individual intensities and the phase difference
    → between them.

    Args:
        x0: A numpy array representing the intensity of the first wave.
        x1: A numpy array representing the intensity of the second wave.
        x2: A numpy array representing the phase difference between the two waves.
        params: Array of numeric parameters to be optimized.

    Returns:
        A numpy array with the calculated resultant intensity.
    """
    return params[0]*x0 + params[1]*x1 + params[2]*x2 + params[3]

```

```

def equation_v1(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) ->
    np.ndarray:
    """Improved version of equation_v0"""

```

LLM Response

```

def equation(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) ->
    np.ndarray:
    """
    Find mathematical function for the resultant intensity due to interference of
    → two waves given their individual intensities and the phase difference
    → between them.
    """

    Args:
        x0: A numpy array representing the intensity of the first wave.
        x1: A numpy array representing the intensity of the second wave.
        x2: A numpy array representing the phase difference between the two waves.
        params: Array of model numeric constants or parameters to be optimized.

    Returns:
        A numpy array with the calculated resultant intensity.
    """
    I = params[0]*np.sqrt(x0*x1)*np.cos(x2) + params[1]*x0 + params[2]*x1 +
    params[3]
    return I

```

✓ $x_0 = I_1$
✓ $x_1 = I_2$
✓ $x_2 = \delta$

$$I_* = I_1 + I_2 + 2\sqrt{I_1 I_2} \cos(\delta)$$

Feynman I.37.4 Problem

Figure 10: An example of LLM response to Feynman I.37.4 problem, demonstrating model memorization without iterative search.

Input Prompt

```

Can you give me the complete version of last function? Instruction
Consider physical meaning and relationships of inputs while completing the function.

Find the mathematical function structure that represents the relationship between
→ the angle of incidence and the angle of refraction. Problem Specification

```

```

import numpy as np
from scipy.optimize import minimize

def loss_function(params, x0, x1, y_true):
    y_pred = equation(x0, x1, params)
    return np.mean((y_pred - y_true) ** 2)

def evaluate(data: dict) -> float:
    """Evaluate equation on input and output observations."""
    # Load true data observations
    inputs, outputs = data['inputs'], data['outputs']
    # Optimize equation skeleton parameters
    loss_partial = lambda params: loss_function(params, *inputs.T[:, 2], outputs)
    params_initial_guess = [1.0] * P
    result = minimize(loss_partial, params_initial_guess, method='BFGS')
    optimized_params = result.x
    # Return evaluation score
    score = loss_function(optimized_params, *inputs.T[:, 2])
    return -score if not np.isnan(score) and not np.isinf(score) else None

```

```

def equation_v0(x0: np.ndarray, x1: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Find mathematical function for the angle of refraction given the refractive
    → index and the angle of incidence.

    Args:
        x0: A numpy array representing the refractive index of the medium.
        x1: A numpy array representing the angle of incidence.
        params: Array of model numeric constants or parameters to be optimized.

    Returns:
        A numpy array with the calculated angle of refraction.
    """
    return params[0] * np.arcsin(params[1] * x0 * np.sin(x1)) + params[2]

```

```

def equation_v1(x0: np.ndarray, x1: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Improved version of equation_v0"""

```

LLM Response

```

def equation(x0: np.ndarray, x1: np.ndarray, params: np.ndarray) -> np.ndarray:
    """
    Find mathematical function for the angle of refraction given the refractive
    → index and the angle of incidence.

    Args:
        x0: A numpy array representing the refractive index of the medium.
        x1: A numpy array representing the angle of incidence.
        params: Array of model numeric constants or parameters to be optimized.

    Returns:
        A numpy array with the calculated angle of refraction.
    """
    return params[0] * np.arcsin(params[1] * x0 * np.sin(x1)) + params[2]

```

✓ $x_0 = n$
✓ $x_1 = \theta_2$

$$\theta_1 = \arcsin(n \sin(\theta_2))$$

Feynman I.26.2 Problem

Figure 11: An example of LLM response to Feynman I.26.2 problem, demonstrating model memorization without iterative search.

one-pass responses to some Feynman problems not only exhibit functional accuracy but also mirror the exact form of the corresponding physics expressions.

In contrast, the newly designed benchmark problems, which challenge the model with less familiar components, require more reasoning and refinement steps for LLM-SR to discover well-fitting equations. The increased number of iterations in the refinement process demonstrates that these new problems effectively simulate the discovery process rather than simply relying on retrieval from memory. This observation highlights the necessity for introducing novel benchmark problems that can better assess the true reasoning capabilities of LLMs in equation discovery while leveraging its scientific prior knowledge, as the

Input Prompt	LLM Response
<pre>Can you give me the complete version of last function?</pre> <p style="text-align: right;">Instruction</p> <pre>Consider physical meaning and relationships of inputs while completing the function.</pre> <p style="text-align: right;">Problem Specification</p> <pre>Find the mathematical function structure that represents the force on a charge due to electric and magnetic fields.</pre> <p style="text-align: right;">Evaluation and Optimization</p> <pre>import numpy as np from scipy.optimize import minimize def loss_fn(params, x0, x1, x2, x3, x4, y_true): y_pred = equation(x0, x1, x2, x3, x4, params) return np.mean((y_pred - y_true) ** 2) def evaluate(data: dict) -> float: """Evaluate equation on input and output observations.""" # Load true data observations loss_fn = lambda params: loss_function(params, *inputs[TI:6], outputs) loss_fn_partial = lambda params: loss_fn(params, *inputs[TI:6]) result = minimize(loss_fn_partial, params_initial_guess, method='BFGS') optimized_params = result.x # Return evaluation score score = result.fun if np.isnan(score) or not np.isinf(score): return -score else: return None def equation_v0(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, x3: np.ndarray, x4: np.ndarray, params: np.ndarray) -> np.ndarray: """Find mathematical function for force given a charge, electric field, magnetic field, velocity of the charge, and the angle between velocity and magnetic field. Args: x0: A numpy array representing the charge. x1: A numpy array representing the electric field. x2: A numpy array representing the magnetic field. x3: A numpy array representing the velocity of the charge. x4: A numpy array representing the angle between velocity and magnetic field direction. params: Array of model numeric constants or parameters to be optimized. Returns: A numpy array with the calculated force. """ return params[0]*x0+x1*x1 + params[1]*x2*x2 + params[2]*x3*x3 + params[3]*x4*x4 #+ params[5] def equation_v1(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, x3: np.ndarray, x4: np.ndarray, params: np.ndarray) -> np.ndarray: """Improved version of equation_v0""" pass</pre> <p style="text-align: right;">Function to Complete</p>	<pre>def equation(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, x3: np.ndarray, x4: np.ndarray, params: np.ndarray) -> np.ndarray: """ Find mathematical function for force given a charge, electric field, magnetic field, velocity of the charge, and the angle between velocity and magnetic field. Args: x0: A numpy array representing the charge. x1: A numpy array representing the electric field. x2: A numpy array representing the magnetic field. x3: A numpy array representing the velocity of the charge. x4: A numpy array representing the angle between velocity and magnetic field direction. params: Array of model numeric constants or parameters to be optimized. Returns: A numpy array with the calculated force. """ x0 = q x1 = E_f x3 = v x4 = theta F = params[0]*x0+x1*x1 + params[1]*x2*x2 + params[2]*x3*x3 + params[3]*x4*x4 return F</pre> <p style="text-align: right;">LLM Response</p> <p style="text-align: center;">$\xrightarrow{\quad} \checkmark \quad x_0 = q \quad x_3 = v$ $x_1 = E_f \quad x_4 = \theta$</p> <p style="text-align: center;">$F = q(E_f + B v \sin(\theta))$</p>

Figure 12: An example of LLM response to Feynman I.12.11 problem, demonstrating model memorization without iterative search.

commonly used Feynman equations may not provide an accurate evaluation due to the models’ over-exposure and memorization of these exact physic equations.

D Additional Details on Benchmark Problems and Datasets

D.1 Nonlinear Oscillator Equations

Dataset	Time range	initial values	F	α	β	δ	γ	ω
Oscillator 1	(0, 50)	{x=0.5, v=0.5}	0.8	0.5	0.2	-	0.5	1.0
Oscillator 2	(0, 50)	{x=0.5, v=0.5}	0.3	0.5	1.0	5.0	0.5	-

Table 2: Parameter values for Oscillator datasets.

In this work, we simulate two nonlinear oscillators using ‘solve_ivp’,⁶ in `scipy` library to generate data. The parameters and initial values of these simulations are provided in Table 2, and the equations used to generate data for this task are repeated here for ease of reference:

$$\dot{v} = F \sin(\omega x) - \alpha v^3 - \beta x^3 - \gamma x \cdot v - x \cos(x)$$

and

$$\dot{v} = F \sin(\omega t) - \alpha v^3 - \beta x \cdot v - \delta x \cdot \exp(\gamma x).$$

Fig. 13 illustrates the phase plane diagrams of these nonlinear damped oscillators. These oscillators have nonlinear terms for driving force, restoring force, and damping force, making them challenging problems for equation identification. To assess the generalization capability of the predicted equations, we partition the data into training, in-domain validation, and out-of-domain validation sets based on the trajectory time in simulations. Specifically, we utilize the time interval $T = [0, 20]$ to evaluate the out-of-domain generalization of the equations.

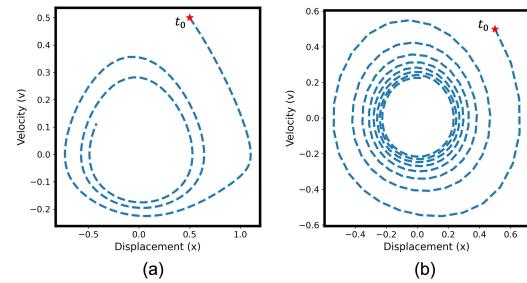


Figure 13: Phase diagrams of trajectories corresponding to custom oscillators: (a) Oscillator 1 and (b) Oscillator 2

⁶https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

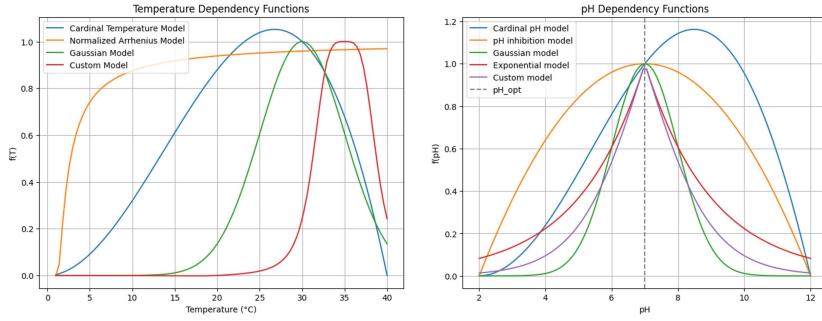


Figure 14: Scheme of some common temperature and pH models in bacterial growth, as well as the custom model behavior.

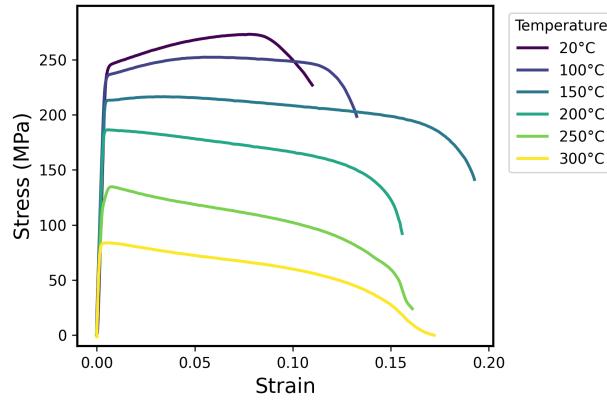


Figure 15: Stress-strain curves of Aluminium 6061-T651 under various temperatures (data from (Aakash et al., 2019))

D.2 E. coli Growth Rate Equations

The custom equation used to generate data for this task is:

$$\frac{dB}{dt} = \mu_{max} B \left(\frac{S}{K_s + S} \right) \frac{\tanh(k(T - x_0))}{1 + c(T - x_{decay})^4} \exp(-|pH - pH_{opt}|) \sin\left(\frac{(pH - pH_{min})\pi}{pH_{max} - pH_{min}}\right)^2,$$

where temperature and pH dependency are approximated by complex nonlinear functions with new unseen operators, the effect of population density is assumed to be linear (single bacterial population), and $\left(\frac{S}{K_s + S}\right)$ denotes Monod equation (Monod, 1949) for substrate concentration model. Fig. 14 shows the numeric behavior of custom designed temperature and pH functions, $f_T(T)$ and $f_{pH}(pH)$, along with some previous forms studied in literature (Rosso et al., 1995).

D.3 Material Stress Behavior Analysis

Fig. 15 depicts the stress-strain curves of Aluminium 6061-T651 at different temperatures, showing the elastic, plastic, and failure regions of material and highlighting the significant influence of temperature on the material's response. The stress-strain relationship for aluminum alloys can be described using various constitutive models (Samantaray et al., 2009), such as the Johnson-Cook model (Johnson & Cook, 1985) which incorporates the effects of strain, strain rate, and temperature on the stress. A simplified version of J-C equation (by neglecting the effect of strain rate) is:

$$\sigma = (A + B\varepsilon^n) \left(1 - \left(\frac{T - T_r}{T_m - T_r} \right)^m \right),$$

where σ is the stress, ϵ is strain, T is the current temperature, T_r is the reference temperature, T_m is the melting temperature, and A, B, C, n, m are material constants.

For this problem, the data represents the tensile behavior of material under uniaxial tension for 6 different temperatures. We allocate the data corresponding to $T = 200^\circ\text{C}$ for use as the validation set.

E Further Results and Visualizations

Performance Trajectory In this section, we evaluate the progress of generated equations using LLM-SR over iterations. This analysis can illustrate the qualitative evolutionary refinement of the discovered equations given the best-shot sampled in-context equation examples provided as experience demonstration sampled from the experience buffer.

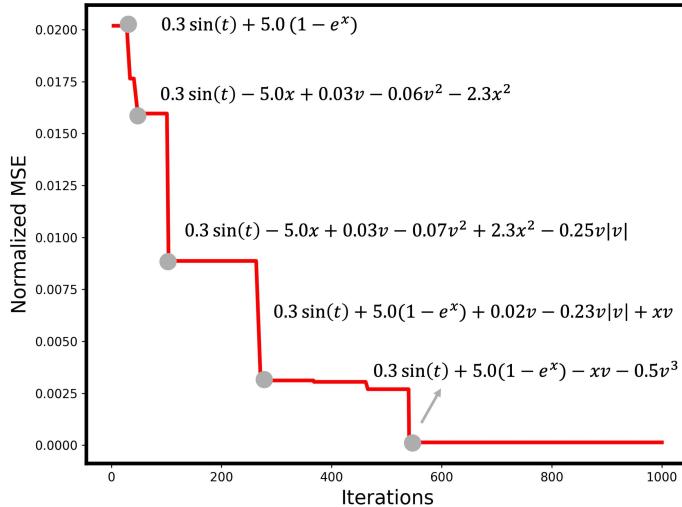


Figure 16: Performance Trajectory of LLM-SR along with the best-scoring found equation programs over iterations on the Oscillation 2 problem.

Fig. 16 shows the NMSE values for the Oscillation 2 dataset. For simplicity, we have provided the simplified equation versions of programs with their optimized parameters. We observe that some of the common nonlinear terms such as sinusoidal driving force term are found early in the search, while more complicated nonlinear terms are found later in the search. An interesting observation here is that while ground truth equation for this dataset is $\dot{v} = 0.3 \sin(t) - 0.5v^3 - x \cdot v - 5.0x \cdot \exp(0.5x)$, LLM-SR has discovered the equation $\dot{v} = 0.3 \sin(t) - 0.5v^3 - x \cdot v + 5.0(1 - \exp(x))$. By evaluating the different terms in these two forms, we observe that in fact $5.0(1 - \exp(x)) \approx -5.0x \cdot \exp(0.5x)$ for $x \in (-2, 2)$, which is the approximate range of displacement in this dataset.

Fig. 17 illustrates the evolution of equations for the Oscillation 1 problem. It is evident that the model attempts to incorporate various nonlinear terms corresponding to driving, restoring, and damping forces, as evidenced by comments or variable names within the code, aiming to enhance accuracy. An intriguing observation emerges wherein the model identifies a trivial solution for the nonlinear oscillation problem, exploiting the flexibility in its code representations. As depicted in Fig. 17, the last step discovered equation yielding very low error turns out to be a trivial solution based on the physical interpretation of variables in this context. This solution was discovered through the utilization of the `np.gradient` function in the `numpy` library, employing numerical differentiation of $0.5v^2$ with respect to displacement (x), leading to an approximation of acceleration: $-6.67 \times \frac{d}{dx}(-0.076v^2) = 0.5 \frac{d}{dx}(v^2) = \frac{dv}{dt}$. Discovery of this edge case by at last iterations is a very interesting observation.

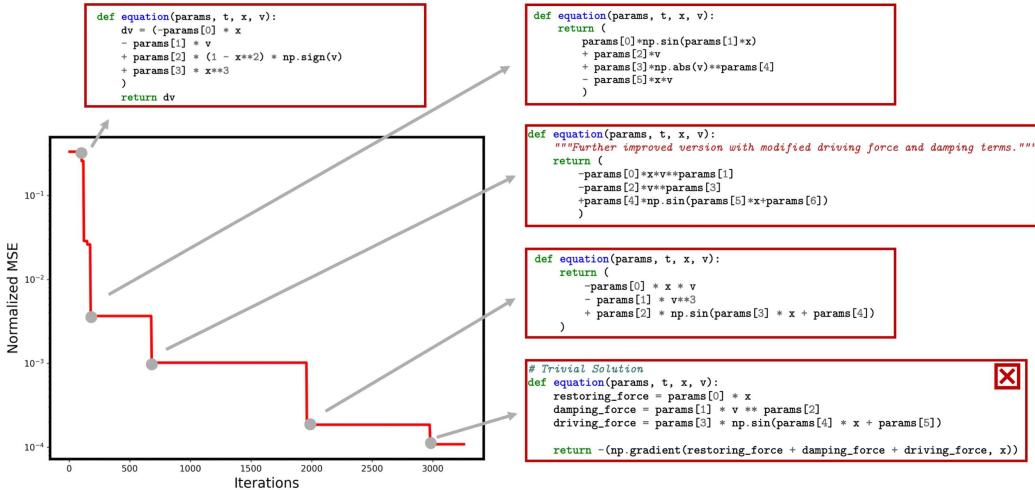


Figure 17: Performance Trajectory of LLM-SR along with the best-scoring found equation programs over iterations on the Oscillation 1 problem.

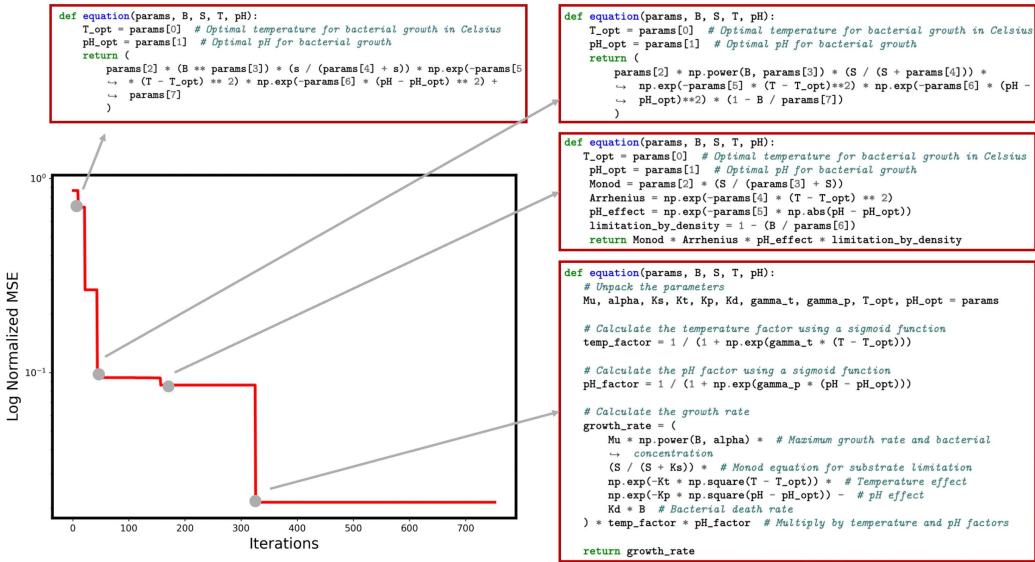


Figure 18: Performance Trajectory of LLM-SR along with the best-scoring found equation programs over iterations on the E. coli growth problem.

Similarly, Fig. 18 presents an annotated performance curve illustrating LLM-SR’s performance on the E. coli growth rate equation discovery benchmark problem. It becomes apparent that the model recognizes the potential presence of optimal values for temperature and pH (pH_{opt} and T_{opt}) from the early iterations which comes from model prior knowledge about the bell-shaped effect of these variables on the growth rate. To enhance accuracy, the model necessitates exploration and incorporation of various nonlinear forms. Notably, LLM-SR directs its evolutionary changes towards the more critical and variable aspects of the problem, specifically the pH and temperature effects, as opposed to other components such as substrate concentration represented by the Monod equation $\frac{S}{K+S}$. Additionally, the figure demonstrates LLM-SR’s comprehension that different components of the function should be multiplied together in the final step, underscoring how prior knowledge of the problem structure can guide LLM-SR’s evolutionary steps.

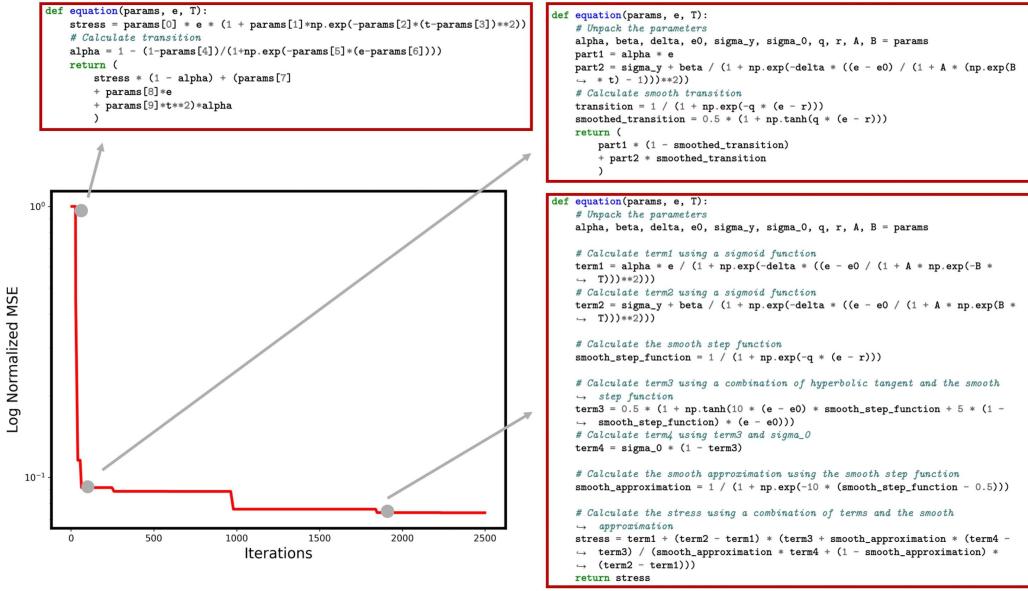


Figure 19: Performance Trajectory of LLM-SR along with the best-scoring found equation programs over iterations on the Stress-Strain problem.

Fig. 19 displays three distinct equation skeleton programs discovered by LLM-SR for the stress-strain problem over search iterations. As in previous cases, we notice the model’s enhancement through exploration and incorporation of various nonlinear terms into the equations. An additional significant observation for this problem is that stress-strain relationships often exhibit piece-wise behavior (as it can also be observed in Fig. 15 representing data obesrvations), which traditional symbolic regression models struggle to identify. However, LLM-SR represents equation skeletons as programs, thus, it can employ conditional rules (If-Else) or their continuous relaxations, utilizing step-wise nonlinear differentiable functions such as the `sigmoid` function to model piece-wise relations. This differentiability and smooth approximation of if-else conditions are particularly helpful for the parameter optimization step, providing smooth functions for the optimizer to navigate.

Qualitative Analysis Fig. 20 and Fig. 21 depict the equation programs identified by LLM-SR and other Symbolic Regression (SR) methods for the E. coli growth and the stress-strain problems, respectively. The diverse range of equation forms identified by different SR baselines reflects the challenges posed by the datasets. Notably, in both datasets, the SR methods yield either lengthy or highly nonlinear equations that deviate from the prior knowledge of the systems, as evidenced by the poor out-of-domain (OOD) performance scores in Table 1. In contrast, LLM-SR finds flexible equation programs that are more interpretable and aligned with the domain-specific scientific priors of the systems.

Fig. 22 and Fig. 23 offer a qualitative comparison by visually presenting the outputs of the equations obtained using LLM-SR, PySR, and uDSR. Upon examination, it becomes evident that the predictions generated by LLM-SR exhibit a notable degree of alignment with both the in-domain and out-of-domain regions of these systems. This alignment suggests that LLM-SR effectively captures the underlying patterns and dynamics of the data, enabling it to better generalize to unseen data points beyond the training domain.

```

Custom E. coli Growth Rate Equation

$$\mu_{max} B \left( \frac{S}{K_s + S} \right) \tanh \left( k(T - x_0) \right) \exp \left( -|pH - pH_{opt}| \right) \sin \left( \frac{(pH - pH_{min})\pi}{pH_{max} - pH_{min}} \right)^2$$


```

```

LLM-SR (GPT 3.5)

$$\text{def equation(params, B, S, T, pH):}$$


$$\quad \# Unpack the parameters$$


$$\quad Mu, K, Ks, Kt, Kb, E, F, G, H, I, T_opt, pH_opt = params$$


$$\quad \# Calculate the temperature factor$$


$$\quad e_factor = np.exp(-Kt * np.abs(I - T_opt)) / E - ((T - T_opt) ** 2) / (2 * E ** 2)$$


$$\quad \# Calculate the pH factor$$


$$\quad pH_factor = 1 / (1 + Kb * np.exp(np.abs(pH - pH_opt) / F))$$


$$\quad \# Calculate the inhibition factor$$


$$\quad I_factor = 1 + 1 * np.clip(np.abs(pH - pH_opt) + np.exp(-np.abs(pH - pH_opt)), 0, 10)$$


$$\quad \# Calculate the temperature-dependent growth factor$$


$$\quad h_factor = G * (T - T_opt) + H$$


$$\quad growth_factor = (1 + np.tanh(0.5 * (h_factor + np.arctan(h_factor)))) * pH_factor * I_factor$$


$$\quad \# Calculate the kinetic factor$$


$$\quad kinetic_factor = S / (Ks + S)$$


$$\quad \# Calculate the growth rate$$


$$\quad growth_rate = Mu + B * np.where($$


$$\quad \quad kinetic_factor > 1,$$


$$\quad \quad kinetic_factor / (1 + np.exp(-K * (kinetic_factor - 1))),$$


$$\quad \quad kinetic_factor)$$


$$\quad ) * e_factor * growth_factor * np.maximum(1 - np.abs(I - T_opt) / 10, 0)$$


$$\quad \text{return growth\_rate}$$


```

LLM-SR (Mixtral)

```

LLM-SR (Mixtral)

$$\text{def equation(params, B, S, T, pH):}$$


$$\quad \# Unpack the parameters$$


$$\quad Mu, K, Ks, Kt, Kb, E, F, G, H, I, T_opt, pH_opt = params$$


$$\quad \# Calculate the temperature factor$$


$$\quad e_factor = np.exp(-Kt * np.abs(I - T_opt)) / E - ((T - T_opt) ** 2) / (2 * E ** 2)$$


$$\quad \# Calculate the pH factor$$


$$\quad pH_factor = 1 / (1 + np.exp(np.abs(pH - pH_opt) / F))$$


$$\quad \# Calculate the inhibition factor$$


$$\quad I_factor = 1 + 1 * np.clip(np.abs(pH - pH_opt) + np.exp(-np.abs(pH - pH_opt)), 0, 10)$$


$$\quad \# Calculate the temperature-dependent growth factor$$


$$\quad h_factor = G * (T - T_opt) + H$$


$$\quad growth_factor = (1 + np.tanh(0.5 * (h_factor + np.arctan(h_factor)))) * pH_factor * I_factor$$


$$\quad \# Calculate the kinetic factor$$


$$\quad kinetic_factor = S / (Ks + S)$$


$$\quad \# Calculate the growth rate$$


$$\quad growth_rate = Mu + B * np.where($$


$$\quad \quad kinetic_factor > 1,$$


$$\quad \quad kinetic_factor / (1 + np.exp(-K * (kinetic_factor - 1))),$$


$$\quad \quad kinetic_factor)$$


$$\quad ) * e_factor * growth_factor * np.maximum(1 - np.abs(I - T_opt) / 10, 0)$$


$$\quad \text{return growth\_rate}$$


```

DSR

$$\frac{T \log(\log(pH))}{pH \times (b + s.pH + T \log(b))}$$

uDSR

$$\begin{aligned} & (-9.1757 \times 10^{-5}B^3 + 3.29934 \times 10^{-4}B^2S + 1.5599 \times 10^{-5}B^2pH - 3.96705 \times 10^{-5}B^2pH \\ & + 6.62027 \times 10^{-5}B^2 + 4.40703 \times 10^{-4}BS^2 - 6.75274 \times 10^{-5}BST - 1.55073 \times 10^{-4}BSpH \\ & - 2.9761 \times 10^{-3}BS - 5.48518 \times 10^{-3}BT^2 - 1.27722 \times 10^{-4}BTpH + 1.40836 \times 10^{-3}BT \\ & + 8.81533 \times 10^{-7}BpH^2 - 1.14007 \times 10^{-2}BpH + 3.00965 \times 10^{-2}B - 2.33351 \times 10^{-5}S^3 \\ & + 3.01844 \times 10^{-5}S^2pH - 1.31112 \times 10^{-5}S^2pH + 1.63188 \times 10^{-5}S^2 - 1.33397 \times 10^{-5}S^2 \\ & - 1.7635 \times 10^{-5}STpH + 4.1777 \times 10^{-5}ST + 2.98769 \times 10^{-5}pH^2 - 2.5354 \times 10^{-5}pH \\ & + 3.0621 \times 10^{-5}S + 4.97662 \times 10^{-6}T^2 + 3.06327 \times 10^{-4}TpH^2 - 4.20817 \times 10^{-3}TpH \\ & + 1.13549 \times 10^{-7} + 1.23534 \times 10^{-4}pH^3 - 4.55259 \times 10^{-4}pH^2 + 7.83677 \times 10^{-2}pH - 0.231108) \\ & \times \exp \left(\sin \left(\frac{pH}{B + pH} \right) \right) \end{aligned}$$

PySR

$$B \times \sin(3.5 \times 10^{-5} \times T^3)^6 \times \sin(\cos(|\sin(0.45pH)| + 0.2))^6$$

Figure 20: Discovered equations from LLM-SR and other SR baseline methods for E. coli bacterial growth rate problem.

```

LLM-SR (Mixtral)

$$\text{def equation(params, e, T):}$$


$$\quad \# e: Strain, T: temperature$$


$$\quad \# Unpack the parameters$$


$$\quad alpha, beta, delta, e0, sigma_y, sigma_0, q, r, A, B = params$$


$$\quad \# Calculate term1 using a sigmoid function$$


$$\quad term1 = alpha * e / (1 + np.exp(-delta * ((e - e0) / (1 + A * np.exp(-B * T)))**2)))$$


$$\quad \# Calculate term2 using a sigmoid function$$


$$\quad term2 = sigma_y + beta / (1 + np.exp(-delta * ((e - e0) / (1 + A * np.exp(-B * T)))**2)))$$


$$\quad \# Calculate the smooth step function$$


$$\quad smooth_step_function = 1 / (1 + np.exp(-q * (e - r)))$$


$$\quad \# Calculate term3 using a combination of hyperbolic tangent and the smooth step$$


$$\quad \text{function}$$


$$\quad term3 = 0.5 * (1 + np.tanh(10 * (e - e0)) * smooth_step_function + 5 * (1 - smooth_step_function) * (e - e0))$$


$$\quad \# Calculate term4 using term3 and sigma_0$$


$$\quad term4 = sigma_0 * (1 - term3)$$


$$\quad \# Calculate the smooth approximation using the smooth step function$$


$$\quad smooth_approximation = 1 / (1 + np.exp(-10 * (smooth_step_function - 0.5)))$$


$$\quad \# Calculate the stress using a combination of terms and the smooth$$


$$\quad \text{approximation}$$


$$\quad stress = term1 + (term2 - term1) * (term3 + smooth_approximation * (term4 - term3) / (smooth_approximation * term4 + (1 - smooth_approximation) * (term2 - term1)))$$


$$\quad \text{return stress}$$


```

```

LLM-SR (GPT 3.5)

$$\text{def equation(params, e, T):}$$


$$\quad \# e: strain, T: temperature$$


$$\quad \# Unpack the model parameters$$


$$\quad k1, k2, k3, k4, k5, k6, k7, k8, k9, k10 = params$$


$$\quad \# Calculate the stress in the elastic region$$


$$\quad elastic_region = 1 - e / k2 * T$$


$$\quad \# Calculate the stress in the plastic region$$


$$\quad plastic_region = k3 * (e ** 2) + k4 * T + k5$$


$$\quad \# Calculate the stress in the failure region$$


$$\quad failure_region = k6 * np.exp(k7 * e) + k8 * (T ** 2) + k9$$


$$\quad \# Define the smooth transition function between elastic and plastic regions$$


$$\quad \# using a sigmoid function$$


$$\quad elastic_to_plastic = 1 / (1 + np.exp(-k8 * (e - k9)))$$


$$\quad \# Define the smooth transition function between plastic and failure regions$$


$$\quad \# using a sigmoid function$$


$$\quad plastic_to_failure = 1 / (1 + np.exp(-k10 * (e - k9)))$$


$$\quad \# Combine the stresses from elastic, plastic, and failure regions using smooth$$


$$\quad \text{transitions}$$


$$\quad stress = ($$


$$\quad \quad elastic_region * (1 - elastic_to_plastic) * (1 - plastic_to_failure)$$


$$\quad \quad + plastic_region * elastic_to_plastic * (1 - plastic_to_failure)$$


$$\quad \quad + failure_region * plastic_to_failure$$


$$\quad )$$


$$\quad \text{return stress}$$


```

DSR

$$\sin \left(\exp \left(-e \cdot T \cdot \exp \left(T^2 \right) + e - T \right) \right)$$

uDSR

$$-3.51e^3 + 2.29e^2 \cdot T + 3.81e^2 - 0.92e \cdot T^2 - 0.30e \cdot T - 4.09e + 1.01T^3 - 2.01T^2 + 0.87T - \log \left(T - \log \left(e^2 \right) \right) + 2.71$$

PySR

$$\sin \left(1.13e^2 \cdot \cos \left(\frac{T}{\cos(e)} \right)^2 \cdot \left(\frac{e}{e + 0.004} \right)^2 \right)$$

Figure 21: Discovered equations from LLM-SR and other SR baseline methods for Stress-Strain problem.

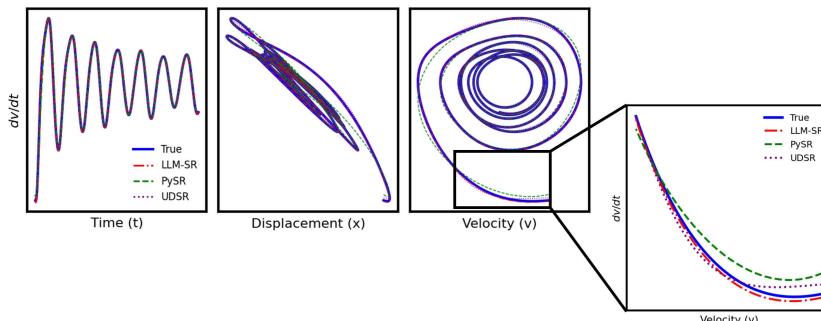


Figure 22: Qualitative evaluation of the performance of LLM-SR on Oscillation 2 problem compared to uDSR and PySR baselines. Plots show the target acceleration \ddot{x} with respect to time (t), displacement (x), and velocity (v).

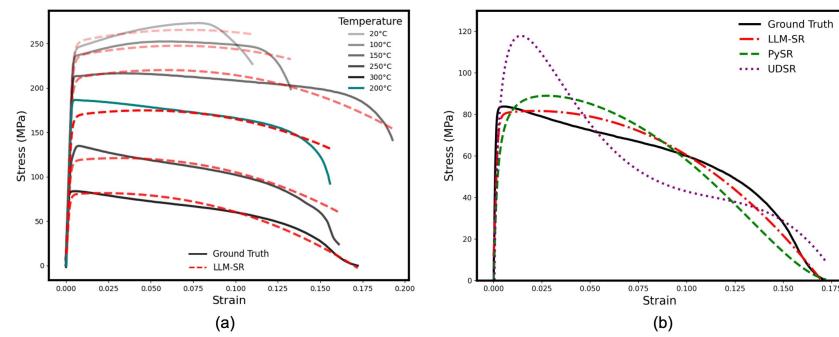


Figure 23: Qualitative evaluation of LLM-SR performance for Stress-Strain problem compared to uDSR and PySR baselines.