DECAEVOLVE: DECOMPOSE, ADAPT, AND EVOLVE, OR, THREE PILLARS OF EFFECTIVE LLM-BASED SCIENTIFIC EQUATION DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Finding mathematical relations underlying natural phenomena and scientific systems has been one of the fundamental tasks in the history of scientific discovery. Recent advancements in evolutionary search with Large Language Models (LLMs), with their embedded scientific knowledge, have shown great promise for this task. However, discovering such mathematical models governing scientific observations still remains significantly challenging, as it requires navigating vast combinatorial hypothesis spaces with an explosion of possible relations. Existing LLM-based approaches overlook the impact of data on the structure of mathematical relations, and treat LLMs as a static hypothesis generator unaware of the observed scientific system. This leads to suboptimal and inefficient exploration of the hypothesis space with over-reliance on LLMs' internal priors. To bridge this gap, we introduce *Decompose, Adapt, and Evolve* (**DecAEvolve**), a framework that leverages granular feedback from symbolic term decomposition and LLM refinement through reinforcement learning (RL) fine-tuning to enhance both robustness and efficiency of evolutionary discovery frameworks. DecAEvolve unifies symbolic decomposition with test-time RL adaptation, enabling adaptive rather than static hypothesis generation and reducing error by up to an order of magnitude compared to state-of-the-art baselines. Our experiments across diverse datasets demonstrate that DecAEvolve significantly improves the accuracy of discovered equations and the efficiency of the discovery process compared to the state-of-the-art baselines.

1 INTRODUCTION

The emergence of Large Language Models (LLMs) has fundamentally transformed automated problem-solving across diverse domains. Beyond their well-established capabilities in natural language understanding and programming (Achiam et al., 2023; Touvron et al., 2023), LLMs have recently demonstrated remarkable reasoning abilities that enable them to tackle complex optimization and discovery tasks. Their capacity to leverage embedded domain knowledge, interpolate between them, generate structured hypotheses and engage in iterative refinement, positions LLMs as powerful engines for systematic exploration of complex solution spaces towards discovery goals (Romera-Paredes et al., 2024; Novikov et al., 2025; Surina et al., 2025). This potential extends naturally to scientific discovery tasks, where the combination of domain expertise and systematic search/exploration in the hypothesis space can unlock new approaches to longstanding challenges of scientific inquiry (Shojaee et al., 2025a).

Scientific equation discovery—the process of uncovering compact and interpretable mathematical models that govern natural phenomena—represents one of the fundamental tasks in automated scientific discovery, with applications across many fields of science such as physics, biology, and material science (Makke & Chawla, 2024). Traditional approaches in Symbolic Regression (SR) rely on genetic programming and evolutionary strategies (Koza, 1994b; Cava et al., 2021b); however, these approaches often struggle with scalability limitations and inefficient exploration of the vast combinatorial hypothesis space (Virgolin & Pissis, 2022). More recent advances have introduced neural-guided approaches, where deep learning architectures are trained to generate or refine symbolic expressions (Udrescu & Tegmark, 2020b; Bruneton, 2025a), and transformer-based methods that are pre-trained with large-scale synthetic data to directly model symbolic sequences as language

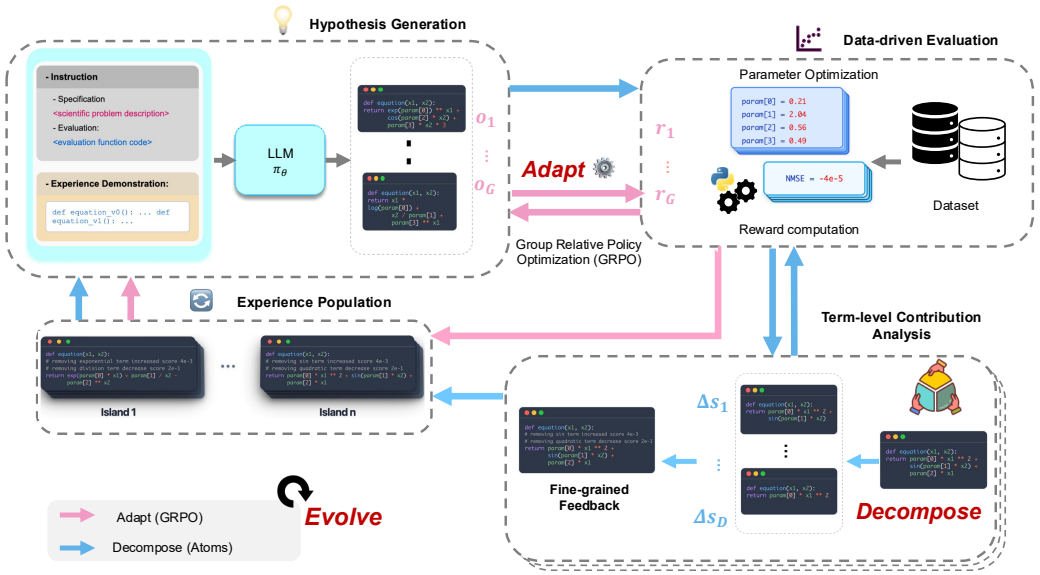

Figure 1: **Overview of the DecAEvolve framework.** The framework integrates *Adaptation* (LLM fine-tuning via reinforcement learning using Group Relative Policy Optimization with data-driven rewards) and *Decomposition* (granular-level feedback through symbolic atomic term analysis) within an *Evolutionary* search process. The adaptation aligns the LLM to the target scientific system beyond its internal priors, while decomposition provides fine-grained guidance for hypothesis refinement. Iterating these three key components enables effective and efficient exploration of the combinatorial hypothesis space in equation discovery.

generation tasks (Kamienny et al., 2022a; Shojaee et al., 2023a; Meidani et al., 2024b). These developments have demonstrated promising capabilities in data-driven learning, yet are limited in balancing learning and search components and in incorporating scientific prior knowledge into the process of discovery.

Several works have recently introduced promising frameworks to integrate LLMs for scientific equation discovery, leveraging their scientific priors and reasoning capabilities to navigate the complex landscape of mathematical expressions more effectively. Notably, LLM-SR (Shojaee et al., 2025a) combines LLMs' scientific knowledge with multi-island evolutionary search, generating equation hypotheses as Python function skeletons guided by data feedback. LaSR (Grayeli et al., 2024b) introduces a concept learning approach that extracts abstract textual concepts from successful equation hypotheses, using these concepts to guide both evolutionary search (with PySR (Cranmer, 2023)) and LLM-based hypothesis generation, and SGA (Ma et al., 2024b) employs a bilevel optimization framework that iteratively combines LLMs for discrete hypothesis generation with physical simulations for continuous parameter optimization. These methods demonstrate this potential by combining LLMs' domain expertise with systematic search strategies, treating equation discovery as a program synthesis problem guided by scientific knowledge (Shojaee et al., 2025b; Reddy & Shojaee, 2025).

Our key insight is that equation discovery benefits from both adaptation (aligning the model with data distributions) and decomposition (understanding which symbolic components matter), neither of which prior frameworks integrate. However, current LLM-based discovery methods exhibit fundamental limitations that constrain their effectiveness. First, they treat LLMs as static hypothesis generators, where the model's parameters remain fixed regardless of the problem domain, nuances of the specific observed system or, insights gained during the search process. This prevents LLMs from adapting their generation strategies based on the specific problem, the data, and the domain-specific requirements. Second, existing approaches mainly provide coarse-grained feedback about solution quality, typically limited to scalar reward signals (e.g., Mean squared error) from execution of whole hypothesis that indicate which hypotheses perform well respectively, without revealing why specific mathematical components or patterns drive success. This limited feedback mechanism prevents LLMs from understanding the underlying symbolic structure of successful solutions and refining their search strategies accordingly.

To address these limitations, we introduce **DecAEvolve** (Decompose, Adapt, and Evolve), a novel framework that enhances the effectiveness and efficiency of LLM-based equation discovery through several synergistic contributions:

- We develop a systematic methodology for providing LLMs with interpretable directional feedback about which components of their generated hypothesis prove effective. Through structured hypothesis decomposition and evaluations, the contributions of individual terms and their interactions are quantified and provided as feedback. This enables LLMs to understand not just which hypotheses succeed, but *why* specific mathematical building blocks are effective, transforming blind generation into informed iterative refinement.
- We employ reinforcement learning with Group-Relative Policy Optimization (GRPO) to implicitly distill the data distribution into the model's parameters for better hypothesis generation process. This test-time adaptation/training approach allows the LLM to learn from successful equation discoveries without directly observing raw data, progressively aligning its hypothesis generation with the underlying symbolic relationships through reward-weighted gradient updates.
- We demonstrate that these synergistic contributions dramatically improve search efficiency, requiring significantly fewer iterations to discover accurate symbolic expressions. Our comprehensive evaluation across multiple benchmarks shows superior performance compared to LLM-SR and other baselines in both in-domain and out-of-domain settings, validating the effectiveness of our guided discovery approach.

2 PRELIMINARIES

Problem Formulation. In scientific equation discovery, the goal is to find a compact mathematical expression (hypothesis) $h(x; \mathcal{T})$ that approximates an unknown target function $f : \mathbb{R}^d \to \mathbb{R}$ within the context of a specific problem $\mathcal{T}$, using a dataset of input-output pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. The objective is to discover functional relationships such that $f(x_i) \approx y_i$ for all i, producing expressions that are both interpretable and capable of generalizing to unseen data. Performance is typically evaluated using fitness to data with metrics such as mean squared error: $\text{MSE}(h, \mathcal{D}) = -\frac{1}{n}\sum_{i=1}^{n}(h(x_i; \mathcal{T}) - y_i)^2$.

LLM-SR Framework. A recent advance in this space is LLM-SR (Shojaee et al., 2025a), which reframes symbolic regression as a program synthesis task. Instead of traditional expression trees, equations are represented as executable Python functions with placeholder parameters. The framework leverages large language models (LLMs) to iteratively generate equation program skeletons, drawing on their embedded scientific priors and reasoning abilities. Each proposed skeleton undergoes parameter optimization (via BFGS or Adam) to fit the data, and high-scoring hypotheses are stored in a dynamic experience buffer. Subsequent LLM prompts incorporate these top hypotheses as in-context examples, enabling iterative refinement of the search process. Our work is based on this work, enhancing it with decomposition and adaptation mechanisms for a more accurate and efficient discovery.

Group-Relative Policy Optimization (GRPO). GRPO (Shao et al., 2024) is an effective reinforcement learning technique to fine-tune LLMs with verifiable outcome rewards. The key innovation of GRPO is its use of group-relative advantages computed from multiple completions per prompt to provide stable reward signals. For each prompt x with completions $\{y_i\}_{i=1}^G$ from old policy $\pi_{\theta_{\text{old}}}$ and corresponding rewards $\{r_i\}_{i=1}^G$, the method calculates group-relative advantages $A_i = r_i - b(x)$ where baseline $b(x) = \frac{1}{G}\sum_{j=1}^{G} r_j$ serves as the per-prompt baseline. This per-prompt normalization provides variance reduction and stable credit assignment across candidates. The GRPO objective optimizes:

$$\mathcal{L}(\theta) = -\,\mathbb{E}_{x,\{y_i\}\sim\pi_{\theta_{\text{old}}}} \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\Big\{\min\Big[\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x,y_{i,<t})}A_{i,t}, \text{clip}\Big(\frac{\pi_\theta(y_{i,t}|x,y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x,y_{i,<t})}, 1-\epsilon, 1+\epsilon\Big)A_{i,t}\Big] + \beta\,\mathbb{E}_x\big[KL(\pi_\theta(\cdot|x)\,\|\,\pi_{\text{ref}}(\cdot|x))\big]\Big\}, \tag{1}$$

where π_{ref} is the frozen reference model, $\beta > 0$ controls the KL regularization strength, and ϵ controls the clipping ratio to avoid excessive single-step updates to the policy. This approach enables efficient adaptation to task-specific rewards while maintaining model stability.

Algorithm 1. DecAEvolve

Input: LLM π_θ; dataset $\mathcal{D}$; problem $\mathcal{T}$; N GRPO steps; G GRPO completions; T iterations; k in-context examples; b samples/prompt
Output: Best equation h^* and score s^*

```
# Stage 1: Test-time adaptation with GRPO
```
for $n \leftarrow 1$ **to** N **do**
  $\{h_i\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid \mathcal{T})$ `# Sample hypothesis group`
  $\{r_i\}_{i=1}^{G} \leftarrow \text{Score}_{\mathcal{T}}(\{h_i\}_{i=1}^{G}, \mathcal{D})$
  $\pi_\theta \leftarrow \text{GRPO\_Update}(\pi_\theta, \{r_i\}_{i=1}^{G})$
```
# Stage 2: Evolutionary discovery with decomposition feedback
```
$\mathcal{P}_0 \leftarrow \text{InitPop}()$ `# Initialize population`
$h^* \leftarrow$ null, $s^* \leftarrow -\infty$
for $t \leftarrow 1$ **to** $T-1$ **do**
  $E \leftarrow \{e_j\}_{j=1}^{k}$ where $e_j \sim \text{SampleExp}(\mathcal{P}_{t-1})$
  $\mathbf{p} \leftarrow \text{MakeFewShotPrompt}(E)$
  $\mathcal{H}_t \leftarrow \{h_j\}_{j=1}^{b}$ where $h_j \sim \pi_\theta(\cdot \mid \mathbf{p})$ `# Generate candidates`
  for $f \in \mathcal{H}_t$ **do**
    $s \leftarrow \text{Score}_{\mathcal{T}}(h, \mathcal{D})$ `# Data-driven score`
    $\{u_m\} \leftarrow \text{Decompose}(h)$ `# AST-based term extraction`
    for *each term* u_m **do**
      $c_m \leftarrow \text{TermContribution}(u_m, h, \mathcal{D})$
    $f \leftarrow \text{Annotate}(h, \{c_m\})$ **if** $s > s^*$ **then**
      $h^* \leftarrow h;\ s^* \leftarrow s$
    $\mathcal{P}_t \leftarrow \mathcal{P}_{t-1} \cup \{(h, s, \{c_m\})\}$ `# Update population`

3 METHOD

DecAEvolve (Decompose, Adapt, and Evolve) combines adaptation, decomposition, and evolutionary search. Adaptation employs GRPO to align the LLM with data-driven rewards, while decomposition delivers term-level feedback that highlights which symbolic components drive accuracy. Coupled with evolutionary search, these mechanisms form a feedback-driven loop that improves both equation quality and the model's generation policy. We illustrate the pseudocode for **DecAEvolve** in Algorithm 1 and emphasize its two key contributions: (i) reinforcement-learning-based test-time adaptation and (ii) fine-grained decomposition feedback.

3.1 TEST-TIME ADAPTATION WITH GRPO

In the adaptation stage of DecAEvolve, the generation policy π_θ^t is updated to improve its ability to propose valid and accurate symbolic equations. After each iteration of hypothesis group generation, the policy is fine-tuned using GRPO (Shao et al., 2024) objective function (equation 1) on the accumulated dataset $\{(\mathbf{p_i}, \{h_i, r_i\}_{i=1}^{G})\}$, where each prompt $\mathbf{p_i}$ is paired with multiple hypothesis completions $\{h_i\}$ and their data-driven rewards $r_i = \text{Score}_{\mathcal{T}}(\{h_i\}_{i=1}^{G}, \mathcal{D})$. Our formulation directly optimizes scalar rewards and integrates invalid completions by assigning them a fixed floor reward $r = 0.01$. Retaining invalid samples ensures that the model receives an explicit negative signal, which suppresses invalid generations over time rather than discarding them. By continually including both valid and invalid completions, the GRPO update not only reinforces promising functional forms but also suppresses degenerate outputs, ensuring that the model learns to generate programs that are executable and accurate.

Fine-tuning is performed with LoRA adapters (Hu et al., 2022), enabling efficient updates while the base parameters remain fixed as π_{ref}. This prevents catastrophic drift while still allowing the adapted policy to progressively shift probability mass toward valid, high-reward symbolic structures for the given problem. Full implementation details and hyperparameters are reported in Appendix B.

3.2 DIRECTIONAL FEEDBACK WITH TERM-LEVEL CONTRIBUTION

At the heart of our framework is an iterative discovery process in which the LLM performs a form of self-reflection: it not only generates candidate symbolic equations but also receives feedback about why certain symbolic components succeed or fail. Rather than relying solely on coarse error scores,

we introduce a contribution analysis that quantifies the role of each term and its pairwise interactions, producing interpretable signals that guide subsequent generations.

Concretely, each candidate program is parsed into an abstract syntax tree (AST; see Appendix. D) and decomposed into atomic symbolic units $\{u_m(x)\}_{m=1}^{M}$ where M is the total number of units. To evaluate the contribution of each single term u_i (or a pair (u_i, u_j)), we first construct ablated hypotheses $f_{\setminus u_i}(x) = \sum_{m \neq i} w_m u_m(x)$ (or similarly $f_{\setminus \{u_i,u_j\}}(x) = \sum_{m \notin \{i,j\}} w_m u_m(x)$). Subsequently, we re-fit the parameters $\mathbf{w}$ on D, and compute the marginal effect $\Delta_{u_i} = S(f, D) - S(f_{\setminus u_i}, D)$ (or $\Delta_{u_i,u_j} = S(f, D) - S(f_{\setminus \{u_i,u_j\}}, D)$), where $S(f, D) = -\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$. This ensures that each Δ_{u_i} or Δ_{u_i,u_j} reflects the true incremental value of a term or pair, independent of parameterization artifacts. Positive values indicate components that improve predictive accuracy, while near-zero or negative values reveal redundancy.

Importantly, these contribution signals are serialized directly into the synthesized programs as *inline comments* (without affecting executability) and stored in the evolving population. In the next iteration, the LLM samples from this population, and the annotated programs are reused as in-context examples. This design ensures that the model is not only guided by global error metrics but also reflects on explicit evidence of which symbolic building blocks mattered, progressively refining its generation strategy. Full details of the directional feedback mechanism appear in Appendix. A.

4 EXPERIMENTS

We evaluate DecAEvolve on benchmark datasets for LLM-based scientific equation discovery from (Shojaee et al., 2025a), covering domains like physics, biology, and materials science:

Nonlinear Oscillator: Simulates two nonlinear damped oscillators (*Oscillator1*, *Oscillator2*) governed by second-order differential equations in displacement and velocity. Both systems are designed with complex but solvable nonlinear structures that differ from standard oscillator models to challenge LLMs towards discovery through data-driven reasoning.

Bacterial Growth: Models E. coli growth under varying conditions of density, substrate, temperature, and pH. Novel nonlinear terms designed for temperature and pH introduce complexities that require exploration and discovery and are hard to recover from LLM recall.

Stress-Strain Behavior: Captures tensile response of aluminum alloy across temperatures. This dataset uses experimental measurements, providing a more realistic setting with experimental data that challenge LLM-based models beyond synthetic formulations.

We compare DecAEvolve against state-of-the-art non-LLM symbolic regression (SR) baselines, including evolutionary approaches such as GPlearn[1] and PySR[2] (Cranmer, 2023), deep learning methods like DSR (Petersen et al., 2021) and uDSR (Landajuela et al., 2022), and pre-trained Transformer SR models NeSymReS (Biggio et al., 2021) and E2E (Kamienny et al., 2022b). In addition, we evaluate against the leading LLM-based SR baseline, LLM-SR (Shojaee et al., 2025a), under same configurations: 3,000 LLM calls per problem with sampling temperature $\tau = 0.8$. In both approaches, equation parameters are optimized with the BFGS solver from SciPy python library and a 30s timeout used for the execution of each hypothesis. In the GRPO adaptation phase, we use batch size of 16 per device, gradient accumulation 4, learning rate 10^{-5}, and KL coefficient $\beta = 0.05$. For fine-tuning, we use LoRA adapters with $r = 16$. Decomposition analysis is also conducted based on the AST extracted from the equation program to define each term and pairwise term contributions. We conduct experiments on six open-source models (Qwen2.5-1.5B and Qwen2.5-3B, Qwen2.5-7B, Llama-3.2-1B, Llama-3.1-3B, and Llama-3.1-8B) to evaluate effectiveness across different model variants as well as the scaling behaviors across different model capacities within our computational constraints for fine-tuning.

For the analysis, we use the normalized mean squared error (NMSE) as in (Shojaee et al., 2025b): $\text{NMSE} = \frac{\sum_{i=1}^{N_{\text{test}}}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N_{\text{test}}}(y_i - \bar{y})^2}$ on both in-domain (ID) and out-of-domain (OOD) test settings, where N_{test}

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

[2] https://github.com/MilesCranmer/PySR

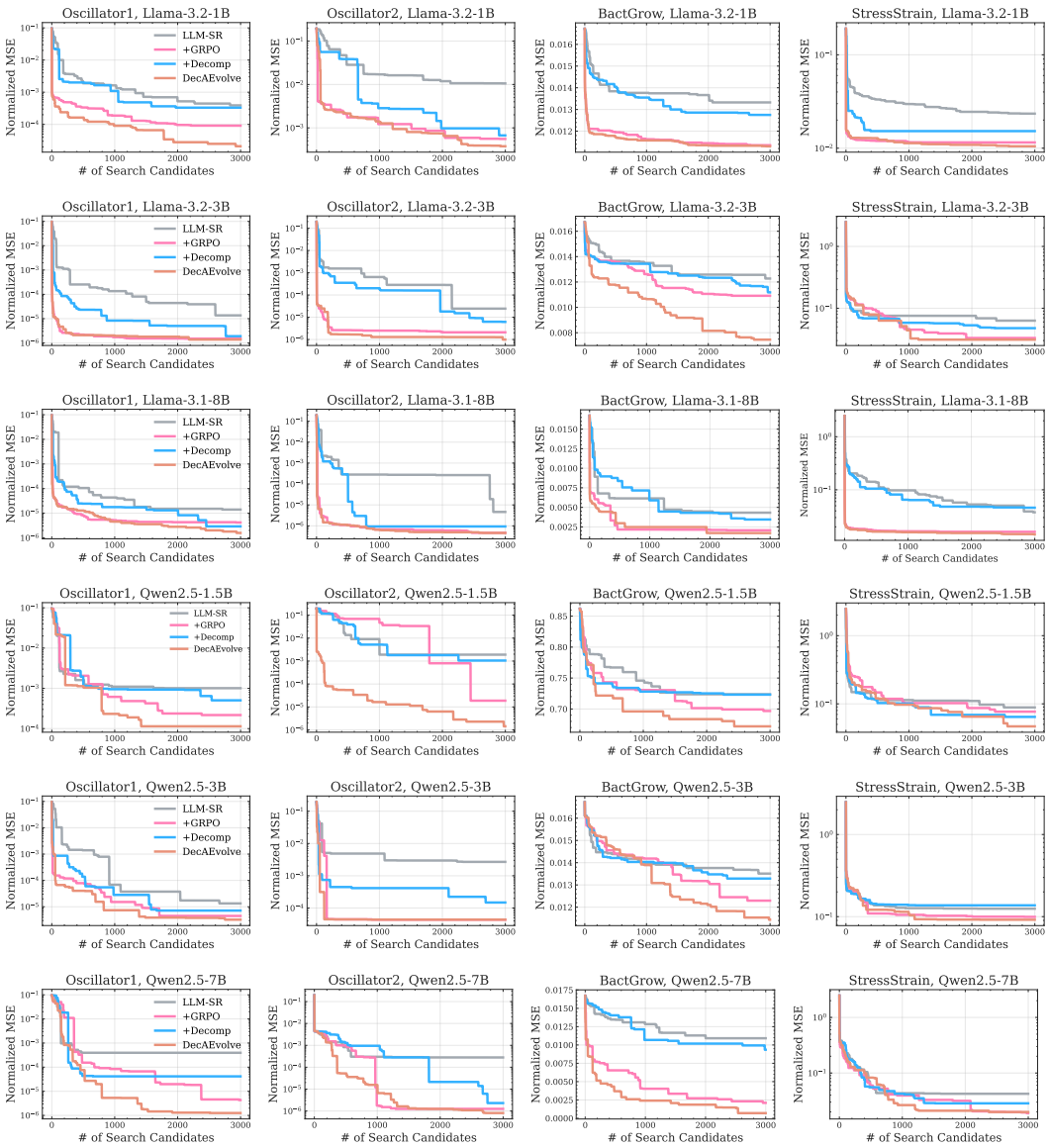

Figure 2: Best-score trajectories of DecAEvolve and its variants against the LLM-SR baseline across benchmark problems. Adaptation (+GRPO) and decomposition (+Decomp) each enhance discovery effectiveness and efficiency, yielding more accurate final equations with fewer search candidates. Their integration in DecAEvolve achieves the best result across all datasets (lower is better).

is the test size and $\bar{y}$ the mean target value. NMSE normalizes errors by scale of dataset variance, enabling comparison across datasets.

4.1 Results

To assess the contribution of our proposed framework DecAEvolve and its key components—decomposition and adaptation—on top of the default evolutionary discovery framework, we compared against the LLM-SR baseline under the same LLM backbones across multiple benchmark datasets. Figure 2 reports the discovery trajectories, showing the progression of the best-achieved normalized MSE (NMSE) across the search process. The results highlight three consistent trends. First, both ablated variants improve over the baseline: the Adaptation (+GRPO) and Decomposition (+Decomp) modules both help to accelerate convergence and lower discovery error. Second, these improvements hold across diverse LLM backbones (Llama-3.2-1B, Llama-3.2-3B, Llama-3.1-8B,

Model	Oscillation 1		Oscillation 2		E. coli growth		Stress-Strain	
	ID↓	OOD↓	ID↓	OOD↓	ID↓	OOD↓	ID↓	OOD↓
GPlearn	0.0155	0.5567	0.7551	3.188	1.081	1.039	0.1063	0.4091
NeSymReS	0.0047	0.5377	0.2488	0.6472	N/A ($d > 3$)		0.7928	0.6377
E2E	0.0082	0.3722	0.1401	0.1911	0.6321	1.4467	0.2262	0.5867
DSR	0.0087	0.2454	0.0580	0.1945	0.9451	2.4291	0.3326	1.108
uDSR	0.0003	0.0007	0.0032	0.0015	0.3322	5.4584	0.0502	0.1761
PySR	0.0009	0.3106	0.0002	0.0098	0.0376	1.0141	0.0331	0.1304
LLM-SR (Mixtral)	7.89e-8	<u>0.0002</u>	0.0030	0.0291	0.0026	<u>0.0037</u>	<u>0.0162</u>	0.0946
LLM-SR (GPT-3.5)	**4.65e-7**	0.0005	**2.12e-7**	3.81e-5	0.0214	0.0264	0.0210	0.0516
LLM-SR (Llama-3.2-1B)	0.0003	0.1121	0.0105	0.0543	0.0133	0.3544	0.0934	0.3821
LLM-SR (Llama-3.2-3B)	1.41e-5	0.0014	0.0021	0.0053	0.0122	0.0588	0.0629	0.1672
LLM-SR (Llama-3.1-8B)	1.36e-5	0.0009	4.61e-6	0.0001	0.0117	0.0240	0.0376	0.0761
LLM-SR (Qwen2.5-1.5B)	0.0011	0.1233	0.0027	0.0721	0.7237	0.9483	0.1249	0.2435
LLM-SR (Qwen2.5-3B)	0.0003	0.0168	0.0018	0.0432	0.0135	0.8011	0.0905	0.2085
LLM-SR (Qwen2.5-7B)	1.33e-5	0.0017	0.0002	0.0011	0.0109	0.1285	0.0423	0.1851
DecAEvolve (Llama-3.2-1B)	2.09e-5	0.0011	0.0018	0.0136	0.0114	0.0698	0.0704	0.0924
DecAEvolve (Llama-3.2-3B)	1.57e-6	0.0004	0.0003	0.0005	0.0074	0.0102	0.0311	<u>0.0358</u>
DecAEvolve (Llama-3.1-8B)	<u>1.37e-6</u>	<u>0.0002</u>	<u>3.64e-7</u>	<u>2.11e-5</u>	<u>0.0019</u>	0.0045	**0.0144**	**0.0322**
DecAEvolve (Qwen2.5-1.5B)	0.0001	0.0784	1.22e-6	0.0012	0.6719	0.9211	0.0916	0.1134
DecAEvolve (Qwen2.5-3B)	3.23e-6	<u>0.0002</u>	4.36e-5	0.0008	0.0115	0.0454	0.0487	0.1612
DecAEvolve (Qwen2.5-7B)	1.25e-6	**1.51e-5**	8.06e-7	**1.64e-5**	**0.0007**	**0.0012**	0.0198	**0.0322**

Table 1: Comparison of DecAEvolve with SR baseline models on different scientific benchmark problems, measured by Normalized Mean Squared Error (lower is better). **The best performance** for each dataset is in bold, and <u>the second best performance</u> is underlined.

Qwen-2.5-1.5B, Qwen-2.5-3B, Qwen-2.5-7B), indicating that the gains are not model-specific but instead stem from the principled design of the framework components that transfer across different backbones. Finally, the full DecAEvolve framework, which integrates all three components of evolution, decomposition, and adaptation, consistently delivers the lowest terminal NMSE and the fastest convergence rate in the discovery process.

Table 1 provides a quantitative comparison of DecAEvolve against both non-LLM baselines and the LLM-based baseline LLM-SR across in-domain (ID) and out-of-distribution (OOD) evaluations. We observe that DecAEvolve consistently outperforms state-of-the-art non-LLM methods (e.g., PySR, uDSR) as well as the LLM-SR baseline when evaluated under the same LLM backbones. These improvements are mostly robust across both ID and OOD test sets, demonstrating not only higher accuracy but also stronger generalization to unseen data distributions. Performance gains are particularly pronounced with larger backbones such as Llama-3.1-8B and Qwen-2.5-7B, which is expected given that the success of DecAEvolve relies on two key components: (1) decomposition, which requires sufficient reasoning capacity to interpret granular feedback, and (2) adaptation, which depends on reinforcement learning finetuning to exploit reward signals from observed scientific data. Larger models are better able to leverage both of these mechanisms, resulting in consistently stronger performance. Nevertheless, we also find that DecAEvolve with smaller backbones can achieve results that are competitive with, and in some cases better than, the originally reported LLM-SR performance using much larger models such as Mixtral and GPT-3.5. This underscores the critical role of adaptation in scientific discovery: by tailoring even modestly sized open-source models to the specific scientific system, DecAEvolve can surpass the performance of significantly larger general-purpose models.

Lastly, Figure 3 shows consistent reward improvement during GRPO adaptation across both model scales and all datasets, validating our reinforcement learning fine-tuning approach as test-time adaptation for equation discovery. Notably, we observe some scale-dependent behaviors where smaller models show more noise in their RL and reward improvement process than their larger model counterparts. Interestingly, the smaller model usually matches larger model performance eventually even on complex datasets, suggesting that targeted adaptation through GRPO can help to effectively bridge the capability gap between model scales for scientific discovery tasks.

5 Related Work

Symbolic regression. Early research in symbolic regression and scientific discovery established a foundation for automated equation finding, relying on genetic programming and evolutionary

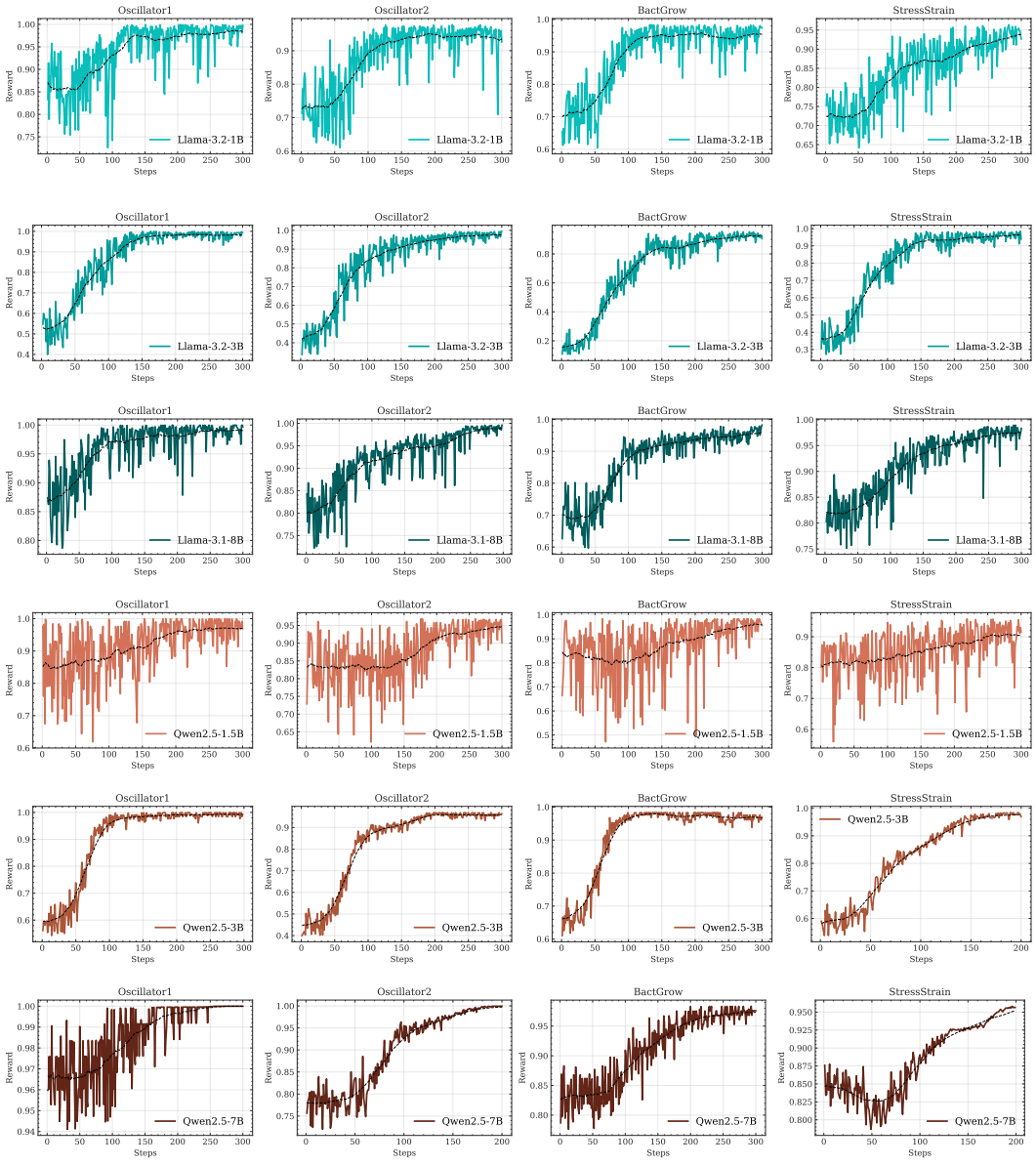

Figure 3: Reward improvements across different models and datasets, showing the success of adaptation with GRPO RL fine-tuning.

search to explore hypothesis spaces (Koza, 1994a; Cava et al., 2021a). While effective on small problems, these approaches struggled with scalability and tended to rediscover shallow functional forms. Later neural-guided methods, such as AI Feynman (Udrescu & Tegmark, 2020a), physics-inspired constraints (Bruneton, 2025b), and transformer-based symbolic generation (Kamienny & colleagues, 2022; Shojaee et al., 2023b; Meidani et al., 2024a), extended capabilities but remained constrained by limited expressivity and an inability to generalize beyond training distributions.

The rise of large language models shifted this landscape. Works such as LLM-SR (Shojaee et al., 2025a) reframed symbolic regression as program synthesis, allowing models to generate equation skeletons enriched by internal scientific priors. Subsequent frameworks expanded this view: LaSR (Grayeli et al., 2024a) guided search with abstracted concepts extracted from prior successes, while bilevel optimizers (Ma et al., 2024a) combined symbolic hypothesis generation with simulation-driven parameter tuning. Benchmarks such as LLM-SRBench (Shojaee et al., 2025c) highlighted both the promise of these methods and their limitations, showing that LLMs, even when coupled with evolutionary refinement, fails to capture the adaptive strategies that real scientific discovery demands.

Test-time adaptation. Test-time adaptation has recently emerged as a way to adapt models during inference, mitigating distribution shift without additional offline training. In reasoning benchmarks such as ARC, gradient-based test-time training (TTT) has shown great performance in better adapting models to tasks that require more novelty (Akyürek et al., 2024). The ARC Prize 2024 report similarly attributes recent state-of-the-art results to pipelines that incorporate test-time training components into the problem-solving process (Chollet & Team, 2024). Beyond empirical advances, recent theoretical analyses establish conditions under which a single gradient step at inference provably enhances transformers as in-context learners (Gozeten et al., 2025). Extending beyond supervised updates, Zuo et al. introduce test-time reinforcement learning (TTRL), where models adapt using consensus-based rewards rather than labels, yielding further improvements across reasoning and math tasks (Zuo et al., 2025). Despite these successes and potential benefits of test-time training in better adapting to novelty, test-time adaptation remains largely unexplored in evolutionary scientific discovery frameworks, leaving open how inference-time learning can directly align priors of pretrained model with the dynamics of specific scientific system during the evolutionary process of search and discovery.

Prompt Optimization and Evolution. A parallel line of work focuses on optimizing prompts rather than model weights, treating instructions and in-context exemplars as a search space. Yang et al. propose OPRO, which frames prompt design as black-box optimization and iteratively improves instructions through model feedback (Yang et al., 2023). Guo et al. extend this perspective with EvoPrompt, combining evolutionary operators such as mutation and crossover with LLMs to explore diverse prompt populations (Guo et al., 2025). More recently, Opsahl-Ong et al. develop MIPRO, a system that jointly optimizes instructions and demonstrations in multi-stage LM programs, demonstrating robust improvements without weight updates (Opsahl-Ong et al., 2024). Agrawal et al. introduce GEPA, which leverages reflective prompt evolution and self-feedback to surpass reinforcement learning baselines like GRPO, achieving higher efficiency in both code and reasoning tasks (Agrawal et al., 2025). Surveys on automatic prompt optimization synthesize these approaches and position prompt evolution as a label- and compute-efficient alternative to RL fine-tuning (Ramnath et al., 2025). Our framework builds on this perspective of self-evolving optimization via prompting along with the test-time adaptation to search deeper and more efficient in the scientific discovery hypothesis space. In this paper, we show how adaptation and prompt optimization can jointly advance evolutionary discovery frameworks.

6 Conclusion

We introduce DecAEvolve, a framework that enhances LLM-based equation discovery through granular term-level feedbacks, test-time adaptation via GRPO and, evolutionary search with LLMs. Our approach transforms static hypothesis generation into adaptive learning, enabling LLMs to progressively align with nuances of underlying observed scientific systems through reinforcement learning model adaptation and interpretable feedback mechanisms. Experimental results across diverse benchmark datasets demonstrate that DecAEvolve consistently outperforms state-of-the-art baselines in both discovery accuracy and search efficiency, while maintaining strong out-of-domain generalization. The success of smaller models through targeted test-time adaptation suggests promising directions for democratizing scientific discovery tools without requiring large, resource-intensive models. Future work could extend our simple decomposition mechanisms to more complex structures and explore better optimization strategies for the evolutionary process. The term-level feedback approach developed here may also prove valuable for broader program synthesis tasks requiring iterative refinement in the symbolic space of programs based on component-level understanding.

REFERENCES

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.

Shubham Agrawal et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Ekin Akyürek et al. The surprising effectiveness of test-time training for abstract reasoning. In *International Conference on Machine Learning (ICML)*, 2024. URL https://ekinakyurek.github.io/papers/ttt.pdf.

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.

J.-P. Bruneton. Enhancing symbolic regression with quality-diversity and physics-inspired constraints (qdsr). *arXiv preprint*, 2025a. URL https://arxiv.org/abs/2503.19043.

Jean-Philippe Bruneton. Enhancing symbolic regression with quality-diversity and physics-inspired constraints (qdsr). arXiv preprint arXiv:2501.01234, 2025b.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2021a.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance, 2021b. URL https://arxiv.org/abs/2107.14351.

François Chollet and ARC Prize Team. Arc prize 2024: Technical report. Technical report, ARC Prize, 2024. URL https://arcprize.org/.

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

Halil Alperen Gozeten, M. Emrullah Ildiz, Xuechen Zhang, Mahdi Soltanolkotabi, Marco Mondelli, and Samet Oymak. Test-time training provably improves transformers as in-context learners, 2025. URL https://arxiv.org/abs/2503.11842.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024a.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b. URL https://arxiv.org/abs/2409.09359.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers, 2025. URL https://arxiv.org/abs/2309.08532.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Author Kamienny et al. End-to-end transformer-based equation generation for symbolic regression. In *NeurIPS*, 2022a.

Pierre Kamienny and colleagues. End-to-end transformer-based equation generation for symbolic regression. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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*, 2022b.

John R. Koza. *Genetic Programming as a Means for Programming Computers by Natural Selection*, volume 4. 1994a.

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

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.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *International Conference on Machine Learning (ICML)*, 2024a.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. LLM and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=hz8cFsdz7P.

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

Kazem Meidani, Parshin Shojaee, Chandan K. Reddy, and Amir Barati Farimani. Snip: Bridging mathematical symbolic and numeric realms with unified pre-training. In *International Conference on Learning Representations (ICLR)*, 2024a.

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*, 2024b. URL https://openreview.net/forum?id=KZSEgJGPxu.

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

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs, 2024. URL https://arxiv.org/abs/2406.11695.

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.

Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, Haozhu Wang, Han Ding, Yuzhe Lu, Zhichao Xu, Yun Zhou, Balasubramaniam Srinivasan, Qiaojing Yan, Yueyan Chen, Haibo Ding, Panpan Xu, and Lin Lee Cheong. A systematic survey of automatic prompt optimization techniques, 2025. URL https://arxiv.org/abs/2502.16923.

Chandan K Reddy and Parshin Shojaee. Towards scientific discovery with generative ai: Progress, opportunities, and challenges. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28601–28609, 2025.

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. *Nat.*, 625(7995):468–475, January 2024. URL https://doi.org/10.1038/s41586-023-06924-6.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression, 2023b. URL https://arxiv.org/abs/2303.06833.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models, 2025a. URL https://arxiv.org/abs/2404.18400.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K. Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=SyQPiZJVWY.

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

Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. *arXiv preprint arXiv:2504.05108*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020a.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: a physics-inspired method for symbolic regression, 2020b. URL https://arxiv.org/abs/1905.11481.

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

Greg Yang et al. Large language models as optimizers. In *NeurIPS*, 2023. URL https://arxiv.org/abs/2309.03409.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2504.16084.

APPENDIX

A DETAILED TERM DECOMPOSITION AND CONTRIBUTION ATTRIBUTION

We give a complete account of how generated programs are decomposed into weighted symbolic units and how single and pairwise contributions are computed. This procedure mirrors the implementation used by the evaluator: the function body is parsed into an AST, simple assignment chains are inlined, the returned expression is decomposed at additive nodes, and ablation is carried out by rewriting only the final assignment `return` and re-executing in a sandbox. *All annotations are serialized as comments in the program without changing executable semantics*. Unless otherwise specified, all ablations include re-fitting of the remaining weights on the dataset to ensure that contributions reflect true incremental value independent of parameterization artifacts.

Scoring. Given data $D = \{(x_i, y_i)\}_{i=1}^{n}$ and a candidate program f, we evaluate

$$S(f, D) \;=\; -\frac{1}{n}\sum_{i=1}^{n}\big(f(x_i) - y_i\big)^2, \tag{2}$$

where f is executed inside a restricted sandbox with timeouts and numeric checks.

Program-to-symbol map and weighted additive form. Let r denote the expression returned by the function (or a final assigned variable). We parse the function body into an AST, build a line-level assignment map, inline r if it is an intermediate variable, and traverse the AST with the following rules: (i) *addition/subtraction* split terms, (ii) *multiplication/division/power* subtrees are preserved as atomic units, and (iii) *unary ops and function calls* (e.g., sin, exp, `np.abs`) are atomic.

The resulting model has the form

$$f(x; \mathbf{w}) \;=\; \sum_{t=1}^{T} w_t\, \tau_t(x), \tag{3}$$

where each $\tau_t(x)$ is an extracted atomic unit and w_t is a scalar weight optimized to fit the dataset. We estimate $\mathbf{w}$ using quasi-Newton optimization (BFGS) provided by NumPy/SciPy, minimizing the squared error loss. This ensures that decomposition is not only symbolic but also numerically calibrated to data.

Single-term ablation and contribution. For each index t, we form the ablated function

$$f_{\setminus t}(x; \mathbf{w}) \;\triangleq\; \sum_{\substack{j=1 \\ j\neq t}}^{T} w_j\, \tau_j(x), \tag{4}$$

by rewriting the final `return` so that term $w_t\tau_t(x)$ is removed. After ablation, the remaining weights $\mathbf{w}_{\setminus t}$ are re-optimized on D (using the same BFGS procedure as in Eq. 3), ensuring that the marginal effects reflect the true incremental value of each term independent of parameterization artifacts. Its marginal contribution is defined as

$$\Delta_t \;\triangleq\; S\big(f(\cdot; \mathbf{w}), D\big) \;-\; S\big(f_{\setminus t}(\cdot; \mathbf{w}_{\setminus t}), D\big). \tag{5}$$

If removal yields invalid outputs, we treat $w_t\tau_t(x)$ as essential and assign maximal contribution under the current score scale.

Pairwise interaction. Similarly, for a pair (t, u) we define

$$f_{\setminus\{t,u\}}(x; \mathbf{w}) \;\triangleq\; \sum_{\substack{j=1 \\ j\notin\{t,u\}}}^{T} w_j\, \tau_j(x), \qquad \Delta_{t,u} \;\triangleq\; S\big(f(\cdot; \mathbf{w}), D\big) \;-\; S\big(f_{\setminus\{t,u\}}(\cdot; \mathbf{w}_{\setminus\{t,u\}}), D\big). \tag{6}$$

As with the single-term case, the weights are re-optimized after removal to isolate the genuine interaction effect. These values reveal redundancy versus synergy by comparing $\Delta_{t,u}$ against Δ_t and Δ_u.

```python
def equation(x, v, params):
    """ Mathematical function for acceleration in damped nonlinear oscillator """

    # Individual Term Contributions:
    # [Ablation] Removing this term decreases the score by 0.00394026: params[0]*x
    # [Ablation] Removing this term decreases the score by 0.00000446: params[2]
    # [Ablation] Removing this term decreases the score by 0.00000001: params[1]*v

    # Term Pair Contributions:
    # [Ablation] Removing these terms together decreases the score by 0.00414153:
    #   Term 1: params[0]*x
    #   Term 2: params[2]
    # [Ablation] Removing these terms together decreases the score by 0.00394474:
    #   Term 1: params[0]*x
    #   Term 2: params[1]*v
    # [Ablation] Removing these terms together decreases the score by 0.00000450:
    #   Term 1: params[1]*v
    #   Term 2: params[2]

    return params[0] * x + params[1] * v + params[2]
```

Figure 4: **Example of program-level annotations.** A candidate equation for a damped nonlinear oscillator is annotated in-line with single-term and pairwise ablation contributions (as comments) immediately above the `return`. The evaluator computes these after optimizing weights $\mathbf{w}$ via BFGS, decomposing the return expression, and re-evaluating ablations in a sandbox. All ablations re-fit the remaining weights to data to ensure consistency with the definitions in Sec. 3.2 of the main paper.

Annotation and persistence. After computing $\{\Delta_t\}$ and $\{\Delta_{t,u}\}$, we serialize them as human-readable comments directly in the function body (above the `return`), and store the annotated program in the experience buffer. This preserves executable semantics while exposing interpretable, component-level feedback that guides subsequent in-context learning.

B Reinforcement Learning Formulation of Offline GRPO

We formulate our offline fine-tuning procedure as reinforcement learning over a deterministic Markov Decision Process (MDP)

$$\mathcal{M} = (S, A, R, T).$$

States. The state space S consists of partial sequences $(x, y_{1:k})$ where x is the prompt and $y_{1:k}$ is a prefix of the generated output.

Actions. At each step the action space A corresponds to selecting the next token $y_{k+1} \in V$.

Transitions. The transition function T is deterministic and appends the chosen token:

$$(x, y_{1:k}) \mapsto (x, y_{1:k+1}).$$

Rewards. Rewards are assigned only at terminal states. For a completed sequence $y_{1:K}$, we execute the synthesized program on a held-out test suite and compute the mean squared error (MSE). The scalar reward is

$$r(x, y_{1:K}) \;=\; \exp\big(-\mathrm{clip}\big(|\mathrm{MSE}(x, y_{1:K})|,\, 0,\, 10\big)\big)\,,$$

with invalid completions included in training and assigned a fixed floor reward $r = 0.01$. We use $\gamma = 1$ since rewards are terminal.

Offline contextual bandit view. The offline dataset consists of prompts paired with groups of candidate completions and their evaluator rewards:

$$D_{\text{off}} = \{(x, \{y_i, r_i\}_{i=1}^{G})\}.$$

This induces a contextual bandit formulation: the context is x, the action is the entire completion y, and the observed return is $r(x, y)$.

Policy objective with trust region. We optimize with a KL-regularized objective relative to a frozen reference model π_{ref}:

$$\max_{\theta} \ \mathbb{E}_{(x,\{y_i\})\sim D_{\text{off}}}\left[\frac{1}{G}\sum_{i=1}^{G} r(x,y_i)\right] \ - \ \beta\,\mathbb{E}_x\big[KL\big(\pi_\theta(\cdot|x)\,\|\,\pi_{\text{ref}}(\cdot|x)\big)\big]\,, \tag{7}$$

with $\beta = 0.05$.

Group-Relative Policy Optimization (GRPO). We adopt GRPO with per-prompt group baselines to reduce variance:

$$A_i(x,y_i) = r_i - b(x), \quad b(x) = \tfrac{1}{G}\sum_{j=1}^{G} r_j.$$

The GRPO objective becomes

$$\max_{\theta} \ \mathbb{E}_{x\sim D_{\text{off}}}\left[\sum_{i=1}^{G} A_i(x,y_i)\cdot \log \pi_\theta(y_i \mid x)\right] \ - \ \beta\,\mathbb{E}_x\left[KL\big(\pi_\theta(\cdot|x)\,\|\,\pi_{\text{ref}}(\cdot|x)\big)\right]. \tag{8}$$

Autoregressive factorization. For LMs,

$$\log \pi_\theta(y_i \mid x) = \sum_{t=1}^{T_i} \log \pi_\theta(y_{i,t} \mid x, y_{i,<t}),$$

and advantages are distributed uniformly across tokens (A_i/T_i for each position). We apply PPO-style clipping with $\epsilon = 0.2$ for stability.

Implementation details. We fine-tune using Adam with learning rate 10^{-6} and a warmup–stable–decay schedule (200 warmup steps). Training uses an effective batch size of 64 (16 per device with gradient accumulation 4). Each prompt is sampled with $G = 64$ completions at temperature 0.8 and top-$p = 0.9$. Only LoRA adapter parameters are updated ($r = 8$, $\alpha = 16$, dropout 0.05), while the base model remains frozen as π_{ref}.

C Inclusion of Invalid Examples

A key design choice in DecAEvolve is to retain invalid hypotheses during adaptation rather than discarding them. While the base model frequently produces non-executable fragments, filtering them out removes informative signals about failure modes. Instead, we assign invalid completions a fixed floor reward:

$$r_i = \begin{cases} \text{evaluator\_score}(h_i), & \text{if } h_i \text{ is valid,} \\ 0.01, & \text{otherwise.} \end{cases} \tag{9}$$

These values enter the Group-Relative Policy Optimization (GRPO) update through the loss

$$\mathcal{L}(\theta) = -\mathbb{E}_{x,\{h_i\}}\left[\sum_{i=1}^{G}\big(r_i - b(x)\big)\ \log \pi_\theta(h_i \mid x)\right] + \beta\,\mathbb{E}_x\Big[\mathrm{KL}\big(\pi_\theta(\cdot \mid x)\,\|\,\pi_{\text{ref}}(\cdot \mid x)\big)\Big], \tag{10}$$

where $b(x) = \frac{1}{G}\sum_{j=1}^{G} r_j$ is the per-prompt baseline and π_{ref} is the frozen reference policy. Since invalid completions always yield a reward below the baseline, they contribute negative advantages and shift probability mass away from degenerate outputs. Empirically, this mechanism steadily reduced the fraction of invalid generations and stabilized adaptation.

D Abstract Syntax Tree (AST)

An Abstract Syntax Tree (AST) is a language-agnostic representation of a program that makes explicit the hierarchical composition of an expression. Internal nodes correspond to operators or function applications (e.g., `+`, `*`, `**`, `np.sin`), and leaves correspond to parameters or input variables. In

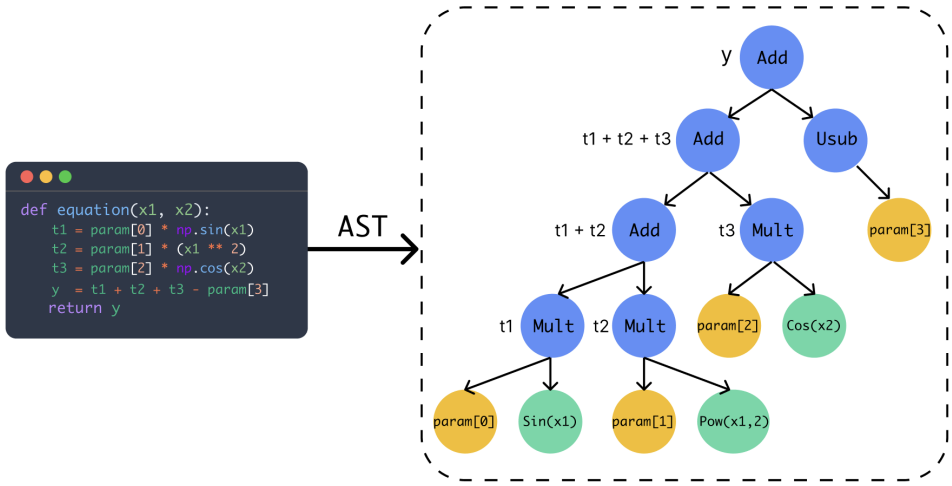

Figure 5: Parsing an equation program into an AST.

our setting, we parse each LLM-generated equation program into an AST and split only at additive nodes, while preserving multiplicative, divisional, power, and functional subtrees as atomic units. This yields a linear combination $f(x) = \sum_{t=1}^{T} \tau_t(x)$ of symbolic atoms τ_t. Figure 5 illustrates this mapping from the generated hypothesis function (left) to its AST (right): a return expression such as $y = t_1 + t_2 + t_3 - t_4$ becomes a top-level sum where each t_i is an intact subtree, enabling principled decomposition without altering operator precedence.

Operationally, the evaluator isolates the evolved function body, executes it in a sandbox to obtain a scalar score, and then reconstructs the returned expression by expanding intermediate assignments via an assignment map and dependency graph before parsing with Python's `ast` module. From the resulting atoms $\{\tau_t\}$, we perform ablation-based attribution: for each term t, we remove τ_t, rebuild a syntactically valid RHS with correct parentheses, re-execute the modified program, and compute a marginal contribution $\Delta_t = S(f, D) - S(f_{\setminus t}, D)$, where S is the evaluator score (negative MSE). We analogously compute pairwise signals $\Delta_{t,u}$ by removing (τ_t, τ_u). The evaluator writes these results back as inline comments within the function body, so subsequent iterations can consume structured, term-level feedback rather than a single scalar reward. This AST-centric pipeline is lightweight, robust to multi-line programs, and provides the granular guidance that underpins our evolutionary refinement.

E Input Prompts

The prompts in Figure 6 were used for evaluating DecAEvolve on the four regression tasks from the LLM-SR(Shojaee et al., 2025a) benchmark. Each prompt specifies the target problem and a Python function template with placeholder parameters. In all cases, the prompts shown below correspond to the initial call to the LLM. In subsequent iterations, DecAEvolve augments the prompt with examples drawn from top-performing hypotheses in the evolving buffer, enabling in-context learning from previously discovered candidates and their annotated contributions.

```
You are a helpful assistant tasked with discovering mathematical function structures for scientific
systems. You are given an example of the function signature in the first function equation_v0 below.
Your task is to complete the last 'equation' function with your mathematical relationship, considering
the physical meaning and relationships of inputs. Only give me the completion code of the BODY of the
current 'equation' FUNCTION. Do NOT give me a new function. Do NOT give me 'pass' instead of
completion. Just give me the new mathematical relationship with inputs for completion of current
function body. Do NOT use equation_v0 in your completion.

"""
Find the mathematical function skeleton that represents acceleration in a damped nonlinear oscillator
system with driving force, given data on position, and velocity.
"""

import numpy as np

#Initialize parameters
MAX_NPARAMS = 10
params = [1.0]*MAX_NPARAMS

def equation_v0(x: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Initial example of equation."""
    dv = params[0] * x  +  params[1] * v  + params[2]
    return dv

def equation_v1(x: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Improved version of `equation_v0`."""
```

((a)) Oscillator I

```
You are a helpful assistant tasked with discovering mathematical function structures for scientific
systems. You are given an example of the function signature in the first function equation_v0 below.
Your task is to complete the last 'equation' function with your mathematical relationship, considering
the physical meaning and relationships of inputs. Only give me the completion code of the BODY of the
current 'equation' FUNCTION. Do NOT give me a new function. Do NOT give me 'pass' instead of
completion. Just give me the new mathematical relationship with inputs for completion of current
function body. Do NOT use equation_v0 in your completion.

"""
Find the mathematical function skeleton that represents acceleration in a damped nonlinear oscillator
system with driving force, given data on time, position, and velocity.
"""

import numpy as np

#Initialize parameters
MAX_NPARAMS = 10
PRAMS_INIT = [1.0]*MAX_NPARAMS

def equation_v0(t: np.ndarray, x: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Initial example of equation."""
    dv = params[0] * t + params[1] * x  +  params[2] * v + params[3]
    return dv

def equation_v1(t: np.ndarray, x: np.ndarray, v: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Improved version of `equation_v0`."""
```

((b)) Oscillator II

```
You are a helpful assistant tasked with discovering mathematical function structures for scientific
systems. You are given an example of the function signature in the first function equation_v0 below.
Your task is to complete the last 'equation' function with your mathematical relationship, considering
the physical meaning and relationships of inputs. Only give me the completion code of the BODY of the
current 'equation' FUNCTION. Do NOT give me a new function. Do NOT give me 'pass' instead of
completion. Just give me the new mathematical relationship with inputs for completion of current
function body. Do NOT use equation_v0 in your completion.

"""
Find the mathematical function skeleton that represents E. Coli bacterial growth rate, given data on
population density, substrate concentration, temperature, and pH level.
"""

import numpy as np

#Initialize parameters
MAX_NPARAMS = 10
PRAMS_INIT = [1.0]*MAX_NPARAMS

def equation_v0(b: np.ndarray, s: np.ndarray, temp: np.ndarray, pH: np.ndarray, params: np.ndarray) ->
np.ndarray:
    """Initial example of equation."""
    return params[0] * b + params[1] * s + params[2] * temp + params[3] * pH + params[4]

def equation_v1(b: np.ndarray, s: np.ndarray, temp: np.ndarray, pH: np.ndarray, params: np.ndarray) ->
np.ndarray:
    """Improved version of `equation_v0`."""
```

((c)) BactGrow

```
You are a helpful assistant tasked with discovering mathematical function structures for scientific
systems. You are given an example of the function signature in the first function equation_v0 below.
Your task is to complete the last 'equation' function with your mathematical relationship, considering
the physical meaning and relationships of inputs. Only give me the completion code of the BODY of the
current 'equation' FUNCTION. Do NOT give me a new function. Do NOT give me 'pass' instead of
completion. Just give me the new mathematical relationship with inputs for completion of current
function body. Do NOT use equation_v0 in your completion.

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

import numpy as np

#Initialize parameters
MAX_NPARAMS = 10
params = [1.0]*MAX_NPARAMS

def equation_v0(strain: np.ndarray, temp: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Initial example of equation."""
    return params[0] * strain  +  params[1] * temp

def equation_v1(strain: np.ndarray, temp: np.ndarray, params: np.ndarray) -> np.ndarray:
    """Improved version of `equation_v0`."""
```

((d)) StressStrain

Figure 6: Input prompts used for evaluating DecAEvolve. (a) Oscillator I, (b) Oscillator II, (c) BactGrow, (d) StressStrain.