Symbolic Regression with a Learned Concept Library

Arya Grayeli * 1 2 **Atharva Sehgal** * 1 **Omar Costilla-Reyes** 3 **Miles Cranmer** 4 **Swarat Chaudhuri** 1

Abstract

We present a novel method for symbolic regression (SR), the task of searching for compact programmatic hypotheses that best explain a dataset. The problem is commonly solved using genetic algorithms; we show that we can enhance such methods by inducing a library of abstract textual concepts. Our algorithm, called LASR, uses zero-shot queries to a large language model (LLM) to discover and evolve concepts occurring in known high-performing hypotheses. We discover new hypotheses using a mix of standard evolutionary steps and LLM-guided steps (obtained through zero-shot LLM queries) conditioned on discovered concepts. Once discovered, hypotheses are used in a new round of concept abstraction and evolution. We validate LASR on the Feynman equations, a popular SR benchmark, as well as a set of synthetic tasks. On these benchmarks, LASR substantially outperforms a variety of state-of-the-art SR approaches based on deep learning and evolutionary algorithms.

1. Introduction

Symbolic regression (SR) (Makke & Chawla, 2024) is the task of finding succinct programmatic hypotheses — written in a flexible, domain-specific programming language — that best explain a dataset. Initially proposed in the 1970s, SR has recently emerged as a prominent approach to automated scientific discovery, with applications in domains from astrophysics (Lemos et al., 2023; Davis & Jin, 2023) to chemistry (Batra et al., 2021; Hernandez et al., 2019) to medicine (Virgolin et al., 2020).

Computational complexity is a fundamental challenge in SR, as the space of hypotheses that an SR algorithm must search is discrete and exponential. Previous work has approached this challenge using methods like genetic programming (Schmidt & Lipson, 2009; Cranmer, 2023), neural-guided search (Cranmer et al., 2020; Shah et al., 2020), deep reinforcement learning (Petersen et al., 2019) and hybrid algorithms (Landajuela et al., 2022). However, new tools to enhance the scalability of SR remain a critical need for applications in SR and scientific discovery.

In this paper, we show that *abstraction* and *knowledge-directed discovery* can be powerful principles in building such scaling tools in SR. State-of-the-art genetic algorithms for SR (Cranmer, 2023) evolve pools of candidate hypotheses using random mutation and crossover operations. By contrast, a human scientist does not just randomly mutate their explanations of data. Instead, they synthesize background knowledge and empirical observations into abstract concepts, then use these concepts to derive new explanations. We show that zero-shot queries to large language models (LLMs) can be used to implement such a discovery process on top of a standard SR algorithm.

Concretely, we present a new method for symbolic regression, called LASR, that discovers a library of abstract, reusable and interpretable textual *concepts* and uses it to accelerate SR. LASR alternates between three phases: (i) *concept-directed hypothesis evolution*, where standard genetic operations over hypotheses are interleaved with LLM-guided mutation and crossover operations conditioned on known library concepts; (ii) the LLM-based *abstraction* of patterns in known high-performing hypotheses into new concepts; and (iii) the LLM-directed *evolution of concepts* into more succinct and general forms. Together, these three steps form an open-ended alternating maximization loop that combines evolutionary exploration with the exploitation of the LLM's background knowledge and in-context learning ability.

We experimentally compare LASR on Feynman Equations (La Cava et al., 2021) — a popular SR benchmark in which the goal is to discover 100 equations from the Feynman Lectures in Physics — against several state-of-the-art genetic and deep learning approaches. LASR can discover 66 of the 100 target equations, while the best existing approach can solve 59. To address the concern that LASR's performance could be attributed to test set leakage, we compare LASR with a state-of-the-art genetic approach on a suite of

---

*Equal contribution [1]UT Austin [2]Foundry Technologies [3]MIT CSAIL [4]University of Cambridge. Correspondence to: Atharva Sehgal <atharvas@utexas.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

synthetic benchmarks. We show that LASR substantially outperforms the baseline.

In summary, the contributions of this paper are as follows:

- We pose the problem of discovering an open-ended, reusable concept library that can accelerate solutions to downstream SR tasks.
- We present LASR, a method for combining zero-shot LLM queries and standard evolutionary operations to simultaneously induce a concept library and high-performing hypotheses. LASR's strategy of using LLMs to accelerate evolutionary algorithms may have future applications in settings beyond SR.
- We offer promising experimental results, including a demonstration that LASR outperforms state-of-the-art algorithms in standard SR tasks and synthetic domains.

2. Problem Formulation

Symbolic Regression. We formulate symbolic regression (SR) as a program synthesis (Chaudhuri et al., 2021) problem. The inputs to this problem include a language $\mathcal{L}$ of programmatic hypotheses and a dataset $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ of input-output examples. The syntax of $\mathcal{L}$ is described by a *context-free grammar* (Hopcroft et al., 2007). The grammar allows each hypothesis π to be represented using a set of mathematical operators (e.g., addition, multiplication, trigonometric functions) that facilitate the composition of simpler hypotheses into more complex ones. We abstractly define the *fitness* of a hypothesis π as the likelihood $p_{\mathcal{L}}(\mathcal{D} \mid \pi)$ that it generates $\mathcal{D}$.

In order to prevent finding non-useful solutions, we impose a *prior probability distribution* $p_{\mathcal{L}}(\pi)$ over hypotheses π that penalizes syntactically complex hypotheses. We now pose SR as the task of finding a hypothesis $\pi^\star$ that maximizes the fitness while minimizing syntactic complexity. The problem can be expressed as a maximum a posteriori (MAP) estimation problem (Ellis et al., 2020):

$$\pi^\star = \arg\max_{\pi} p_{\mathcal{L}}(\pi|\mathcal{D}) = \arg\max_{\pi} \underbrace{p_{\mathcal{L}}(\mathcal{D}|\pi)}_{\text{optimization}} \cdot \underbrace{p_{\mathcal{L}}(\pi)}_{\text{regularization}} \tag{1}$$

Recent work leverages large language models (LLMs) for program synthesis (Chen et al., 2021b; Grand et al., 2023; Li et al., 2022). Large language models (LLMs) approach program synthesis as a token prediction problem, directly approximating the likelihood of programs by training on internet-scale datasets. That is,

$$p_{\mathcal{L}}(\pi|\mathcal{D}) \approx p_{\text{LLM}}(\langle\pi\rangle \mid \langle\mathcal{L}\rangle, \texttt{desc}(\mathcal{D})), \tag{2}$$

where $\langle\pi\rangle$ and $\langle\mathcal{L}\rangle$ are, respectively, textual representations of π and a specification of the syntax of $\mathcal{L}$, and the *task description* $\texttt{desc}(\mathcal{D})$ is a few-shot serialization of a subset of the examples in $\mathcal{D}$.

Symbolic Regression with Latent Concept Libraries. Classical symbolic regression typically assumes no prior knowledge or intuition about the problem. In contrast, human scientific discovery often leverages empirical patterns (Wigner, 1990) and intuitions derived from previously observed data. For example, recognizing a 'power law relationship between variables' has led to the formulation of fundamental empirical laws across various fields, such as the Arrhenius equation in Chemistry, the Rydberg formula in Physics, Zipf's law in Linguistics, and Moore's law in Computer Science.

We model such empirical patterns as natural-language *concepts* drawn from a latent *concept library* $\mathcal{C}$. We frame the relationship between the concept library and programs as a Hierarchical Bayesian model consisting of: (i) a *prior* $p(\mathcal{C})$ representing the natural distribution over concept libraries; (ii) a model $p_{\mathcal{L}}(\pi \mid \mathcal{C})$ that quantifies the likelihood of various hypotheses for a given concept library $\mathcal{C}$; and (iii) the previously mentioned fitness function $p_{\mathcal{L}}(\mathcal{D} \mid \pi)$ for programs π. We assume that the distributions $p_{\mathcal{C}}$ and $p_{\mathcal{L}}(\pi \mid \mathcal{C})$ can be approximated using LLMs. That is, we can prompt an LLM to generate interesting concepts, and we can prompt an LLM with a set of concepts to generate token-sequence representations of hypotheses that adhere to the concepts. Now we state the problem of *symbolic regression with latent concept learning* as one of simultaneously inducing an optimal concept library and an optimal programmatic hypothesis:

$$\arg\max_{\pi,\mathcal{C}} p(\pi, \mathcal{C}|\mathcal{D}) = \arg\max_{\pi,\mathcal{C}} \underbrace{p(\mathcal{D}|\pi)}_{\text{By execution}} \cdot \underbrace{p(\pi|\mathcal{C})}_{\text{By LLM}} \cdot \underbrace{p(\mathcal{C})}_{\text{By LLM}} \tag{3}$$

3. Method

LASR performs a two-stage evolution over natural-language concepts and programmatic hypotheses. The two stages follow an alternating maximization strategy shown in Figure 1: (1) *Hypothesis evolution*: We fix the set of concepts and focus on maximizing the hypotheses' fitness to the dataset, and (2) *Concept abstraction and evolution*: We leverage the best hypotheses found to induce a new library of concepts.

In the rest of this section, we first describe PySR, the SR algorithm (Cranmer, 2023) that LASR extends. Next, we show how to modify this algorithm into one guided by natural-language concepts. Finally, we show how these concepts can be naturally extracted and evolved into new concepts. The full LASR algorithm is presented in Algorithm 1 and visualized in Figure 2. LASR is built in

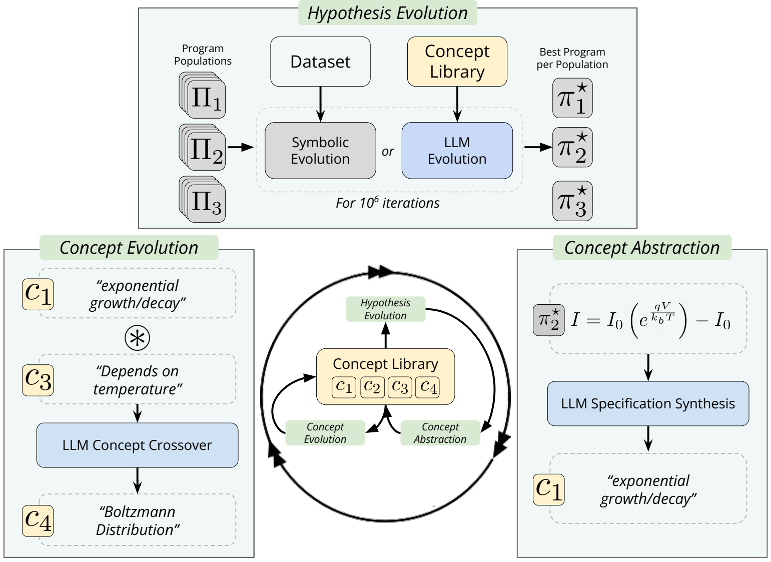


Figure 1: An overview of LaSR. LaSR iteratively refines a library of interpretable textual concepts which are used to bias the search for hypotheses for scientific discovery tasks. This involves three distinct phrases: (**Top**) finding optimal hypotheses within a concept-directed hypothesis evolution, (**Right**) leveraging the optimal hypotheses to find new concept abstractions, and (**Left**) iterating on learned concepts to discover new concepts to accelerate hypothesis evolution. LaSR introduces an orthogonal direction of improvement over current symbolic regression algorithms (Cranmer, 2023) (in gray).

Julia with an additional Python interface[1] and uses an open-source, optimized framework for LLM inference (Kwon et al., 2023).

Base Algorithm: PySR. LaSR builds on PySR (Cranmer, 2023), a scalable, parallelizable genetic search algorithm for SR. The search in PySR maintains multiple populations $\{\Pi_1, \ldots, \Pi_k\}$ of hypotheses, with each hypothesis represented as an expression tree. In its *initialization* step, captured by a procedure InitializePopulations, PySR creates a new expression at random to insert into a population. After running this step, PySR runs a genetic search, encapsulated in a procedure SRCycle, which evolve these populations in parallel, simplifies and optimizes the constants of the resulting hypotheses, and then migrates top-performing hypotheses between populations.

Like other evolutionary algorithms, the search in PySR uses symbolic *mutation* and *crossover* operations. The mutation step is broken into many categories, each with distinct weighting, to either mutate a constant, mutate an operator, add a node (append, prepend, insert), delete a subtree of an expression tree, simplify the tree, initialize a new tree, or do nothing. One of these operations is randomly selected at each call to a mutation request, and each operation executes itself at random but within user-provided constraints. For example, deleting a subtree is done by choosing a random node to replace with a randomly-generated leaf node such as a feature or constant. The crossover step involves swapping random subtrees of two expressions in a population.

LLM-guided Hypothesis Evolution. LaSR speeds up PySR by injecting natural language priors into its search procedure. To do this, we modify the InitializePopulations procedure to use an LLM-augmented initialization operation, and the SRCycle routine to use LLM-augmented versions of its symbolic mutation and crossover operations. The altered procedures are named LlmInit, LlmMutate, and LlmCrossover, respectively. These operations do not *replace* their standard genetic counterparts. Instead, we introduce a hyperparameter p that, with a fixed probability, substitutes the standard genetic operation with the LLM-based operation. This enables "doping" each population with a program that respects the language priors, while ensuring that we do not bottleneck the local exploration of the search space.

The LLM-guided operations follow the same base format: they sample multiple concepts from the concept library, concatenate these concepts with the task-specific variable names and language operations, and append a specialized prompt for each task. We employ zero-shot prompts (see Appendix A.2 for more details) to avoid sampling biases. In further detail:

- LlmInit: The LlmInit function takes an initial set of

---

[1] See code at `anonymous.4open.science/r/neurips24-lasr-70BD`

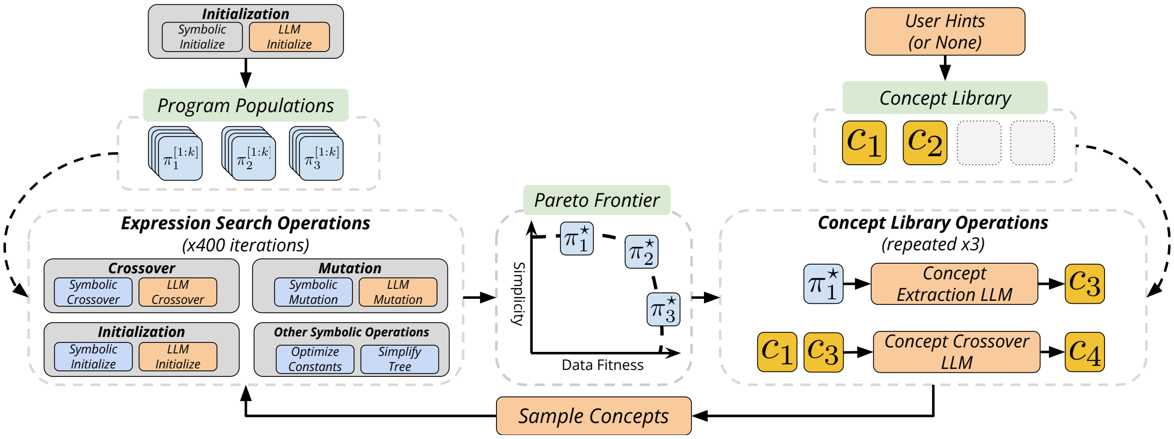


Figure 2: A single step of LaSR. LaSR induces multiple hypothesis populations that are evolved using a scalable evolutionary algorithm. Concept guidance is provided by randomly replacing symbolic operations with concept-directed LLM operations with probability p. After each iteration, the top-performing programs are summarized into natural language concepts, which are evolved to form new concepts that are sampled to guide the search in the next iteration.

concepts and uses them to initialize the populations for the evolutionary search step. The initial set of concepts can either be instantiated from an optional set of user-provided "hints" or generated by the LLM.

- LLMMUTATE: For mutation within a population, we sample a set of l concepts from the concept library C, and construct a prompt that uses this set of concepts to mutate an expression π_i into π_j. The prompt to the LLM takes inspiration from the standard genetic mutation operation, and asks it to mutate the expression given the concepts sampled from the library.
- LLMCROSSOVER: The LLMCROSSOVER function also samples a set of l concepts from the concept library along with two hypotheses π_i and π_j to construct a new expression π_k, which reuses sub-expression trees from the two hypotheses while respecting the sampled concepts. Our implementation is inspired by prior work (Romera-Paredes et al., 2024) — see Figure 4.

Concept Abstraction. After each iteration of symbolic regression, we use a function EXTRACTPARETOFRONTIER to collect: (i) the hypotheses, across all populations, that are Pareto-optimal with respect to the criteria of syntactic simplicity and dataset loss; (ii) the hypotheses with the worst loss across all populations. The resulting set of hypotheses $\mathcal{F} = \{\pi_1^\star., \pi_a^\star \dots \pi_1^-., \pi_b^-\}$ captures the trends that were most helpful and most detrimental to performance during hypothesis search. Now we use the CONCEPTABSTRACTION function, which uses a zero-shot prompt to extract a natural language concept $c^\star$ that summarizes the positive trends while eschewing negative trends. This concept is subsequently added to the concept library. The prompt for the function is presented in Figure 5.

Algorithm 1 Pseudocode for LaSR. LaSR takes as input an optional set of user-provided hints $\mathcal{C}_0$, a dataset of input-output pairs of high-dimensional data $\mathcal{D}$, and four hyperparameters: the number of iterations I, the number of populations K, the number of steps for concept evolution M, and the mixture probability of using LLM-based or GP-based evolutionary operators p. LaSR produces two artifacts: the evolved library of concepts $\mathcal{C}$ and the expression with the highest fitness score $\pi^\star$.

```
function LaSR($\mathcal{C}_0, \mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, I, K, M$, p)
  $\mathcal{C} \leftarrow$ INITIALIZECONTEXTLIBRARY($\mathcal{C}_0$) ▷ Add (optional) user hints to library.
  $\{\Pi_1, \dots \Pi_K\} \leftarrow$ INITIALIZEPOPULATIONS($\mathcal{C}, K$)
  for _ in range($N$) do
    for $i$ in range($K$) do
      $\Pi_i \leftarrow$ SRCYCLE($\Pi_i, \mathcal{D}, \mathcal{C}, p$) ▷ Interleaved Symbolic + LLM Search
    $\mathcal{F} \leftarrow$ EXTRACTPARETOFRONTIER($\{\Pi_1 \dots \Pi_K\}, \mathcal{D}$) ▷ Includes positive + negative programs
    $\mathcal{C} \leftarrow \mathcal{C} \cup$ CONCEPTABSTRACTION($\mathcal{F}, \mathcal{C}$)
    for _ in range($M$) do
      $\mathcal{C} \leftarrow$ CONCEPTEVOLUTION($\mathcal{C}$)
  $\pi^\star \leftarrow$ BESTEXPRESSION($\mathcal{F}$) ▷ Based upon both loss and complexity
  return $\mathcal{C}, \pi^\star$
```

Concept Evolution. Each concept in $\mathcal{C}$ represents trends that were useful at a previous state in the search process. After adding new concepts into the library, we use a function CONCEPTEVOLUTION to evolve the library to include new ideas that logically follow from the ideas in the current library. The implementation of this function follows that of the LLMCROSSOVER operation in that we are using multiple concepts as a reference to generate new ones, with the key distinction that, unlike in the LLMCROSSOVER operation, the fitness of each generated concept here is difficult to quantify. Thus, we include all the generated responses in the concept library. While these concepts may sometimes be inaccurate, they increase the evolutionary algorithm's exploration ability.

4. Experiments

We demonstrate the effectiveness of LASR on multiple tasks integral to scientific discovery. First, we evaluate LASR's performance on the Feynman Equation dataset, a widely adopted scientific discovery benchmark, under a variety of ablations and additional priors. Second, we measure the effect of data leakage by evaluating LASR's performance on a procedurally generated synthetic dataset of challenging equations. Finally, we evaluate LASR on a procedurally generated dataset to ensure that its performance is not affected by data leakage issues in the backbone LLM. LASR's main focus is to serve as a practical toolkit for scientists. Therefore, our evaluation primarily targets slightly noisy ('non-toy') environments, using exact solution rate to gauge performance rather than statistical similarity measures like correlation R^2, which are less relevant to scientific discovery applications.

4.1. Comparison against baselines in the Feynman Equation Dataset

Dataset: The Feynman Equation dataset is a widely adopted benchmark for scientific discovery (Udrescu & Tegmark, 2020). The dataset consists of 100 physics equations extracted from the Feynman lectures on Physics. Each equation is in the form $y = f(x_1, x_2, \dots)$. The number of input variables ranges from two to ten, and the dataset provides 100,000 samples for each equation. We compare against publically available methods benchmarked on SRBench (La Cava et al., 2021). SRBench is a continuously updated benchmark which catalogs the performance of various methods on the Feynman dataset as well as other symbolic regression problems. Specifically, we compare against GPlearn, AFP, AFP-FE, DSR, uDSR, PySR, and the original AI Feynman algorithm (Schmidt & Lipson, 2010; Stephens, 2024; Udrescu & Tegmark, 2020; Landajuela et al., 2022; Petersen et al., 2019). Within this subset, notably, PySR represents an ablation of our model without the LLM genetic operations and the concept evolution (Section 3). We evaluate on a slightly noisy version of this dataset in order to simulate experimental errors common in scientific discovery domains. Specifically, we compare numbers against those reported in and reproduced by SRBench with a target noise of 0.001.

Setup: We instantiate LASR using gpt-3.5-turbo-0125 (Brown et al., 2020) as the backbone LLM and calling it with $p = 0.01$ for 40 iterations, and compare our results with PySR which uses the same default hyperparameters. For the other baselines, we use the numbers reported in SRBench with one exception, uDSR (Landajuela et al., 2022), for which we couldn't find any benchmarking numbers. For this method, we derive the exact solve rate from a publically available figure (Organization).

Results: We showcase results in Table 1. We draw three observations from this experiment. First, LASR achieves a higher exact solve rate than all other baselines. Second, both PySR and LASR outperform the other baselines by a wide margin, indicating that scalable and efficient synthesis is imperative to practical scientific discovery algorithms. Finally, and most notably, a subset of the equations LASR finds could not be derived with any of the previous methods.

4.2. Cascading Experiments

LASR's performance is inherently bottlenecked by the reasoning capabilities of the backbone LLMs and the frequency of their invocation in each iteration. To evaluate the effect of the backbone LLM on LASR's performance, we instantiate a model cascade over two of LASR's hyperparameters: the backbone model (llama3-8b (AI@Meta, 2024), gpt-3.5-turbo-0125) and the probability p with which we call that model in the evolution step (p = [1%, 5%, 10%]).

Setup: Our cascade operates as a tournament. We start LASR with the configuration that provides the least language guidance (llama3-8b at $p = 1\%$) and progressively increase the value of p and then the backbone model. Each subsequent model is only evaluated on the problems that the previous model could not solve. We compare this against PySR's performance on the Feynman equation dataset. To ensure a fair comparison, we cascade PySR using the same procedure but find it does not solve any additional equations.

Metrics: For this experiment, we aim to evaluate the progression of different configuration towards solving equations. The quantitative metric used in the previous experiment, Exact Solve, does not allow for such fine-grained analysis. Therefore, we categorize the synthesized equations into four buckets: Exact Solve, Almost Solve, Close, and Not Close. Exact Solve is quantitatively evaluated using a symbolic match. An equation is tagged as 'Almost Solve' if the dataset loss is small but the generated equation has an extra term or lacks one term. A Close equation captures

GPlearn	AFP	AFP-FE	DSR	uDSR	AIFeynman	PySR	LaSR
20/100	24/100	26/100	23/100	40/100	38/100	59/100	**59 + 7/100**

Table 1: Results on 100 Feynman equations from (Udrescu & Tegmark, 2020). We report exact match solve rate for all models. LaSR achieves the best exact match solve rate using the same hyperparameters as PySR (Cranmer, 2023).

the general structure of the solution (such as a square root nested in an exponential) but not more than that, and Not Close includes all equations that are far from the solution.

Results: Our results are presented in 2. We draw two key observations from these results. First, LaSR outperforms PySR even with minimal concept guidance (llama3-8b at $p = 1\%$). Second, increasing the backbone model size and the mixture probability significantly enhances LaSR's performance, indicating that as the language reasoning capabilities of LLMs improve, so will our performance.

4.3. Ablation Experiments

We conduct ablations on the use of Concept Evolution, Concept Crossover, variable names, and user hints. Figure 3 shows how these ablations affect performance over 40 iterations. We designate an equation as "solved" if, after N iterations, the MSE of our predicted equation is less than 10^{-11}. This metric differs from 'Exact Solved' as defined in the prior experiments: an equation can be 'exactly solved' yet have an MSE higher than 10^{-11} due to the noise floor in the target variables and equation can have low loss but not be an exact match. We observe from the results that: (1) Removing variable names results in a significant performance drop, as we lose semantic meaning provided by variables (for instance, observing θ could suggest employing trigonometric functions on θ). (2) Learning a concept library enables faster convergence to solutions. Without the concept library, task convergence is significantly slower and, in higher concept guidance regimes (adjusting mixture to $p > 0.1\%$), this gap would expand even further.

4.4. Qualitative Analysis and User Hints

The concept library provides an interpretable window into our evolutionary search process. To showcase the concepts learned by LaSR, we take a sample equation from the Feynman dataset, the electric field of a dipole $E_f = \frac{3p_d \cos\theta \sin\theta}{4\pi\epsilon r^3}$ and comment on the libraries learned at various intervals. We see rudimentary concepts emerge in the second iteration:

> *"The presence of basic trigonometric functions like sin in the good expressions contributes to their quality, indicating a connection to physical concepts such as waveforms or periodic phenomena."*

And, in subsequent iterations, the concepts become even more refined:

> *"The good mathematical expressions exhibit a balance between mathematical operations such as multiplication, division, and trigonometric functions, which are known to have physical interpretations and relevance in various scientific phenomena."*

This iterative refinement of concepts helps LaSR maintain consistently good concepts for all iterations. This allows LaSR to converge to an exact match solution within 40 iterations. By contrast, PySR and the concept library ablations fail to converge on an exact match solution, returning equations that — while low-loss — involve many extra terms and structures that aren't in the ground truth equation. This reinforces our hypothesis that injecting semantic meaning into the search process not only leads to more efficient search, but also serves as regularization against complex equations — as the LLM-generated concepts help filter out irrelevant terms. A deeper qualitative analysis is explored in Appendix A.5.

Extending LaSR with Hints: A benefit of LaSR is that its search can be initialized with a set of user-specified, natural-language "hints." To evaluate this capability, we generate vague hints for each equation based on variations of the chapter title of the Feynman lecture that the equation belongs to. We intentionally keep the hints vague to see if knowledge about just the general field is sufficient in improving LaSR's performance. We showcase results in Figure 3. We observe a noticeable boost in performance from injecting these hints, even for our weakest performing model. These findings indicate that even minimal user input can significantly enhance LaSR's effectiveness in scientific discovery tasks.

4.5. Data Leakage Validation

An important consideration in using LLMs for existing SR problems is the possibility that the LLM was exposed to the hold-out problems in the validation set, presenting an unfair advantage to LLMs trained on massive datasets. Intuitively, LaSR generates its own concepts which are conditioned on suboptimal programs, which are unlikely to be within the LLM's memorized responses. To validate this, we generate a dataset of 41 synthetic equations that are engineered to deviate from common physical and mathematical structures and have arbitrary variables. For example, one such equation is $y = \frac{0.782x_3+0.536}{x_2 e^{x_1}(\log x_2 - x_2 e^{\cos x_1})}$. Anecdotally, we find that PySR struggles to solve equations with these characteris-

Type of Solve	PySR	LASR (Llama3-8B) $p = 1\%$	LASR (Llama3-8B) $p = 5\%$	LASR (Llama3-8B) $p = 10\%$	LASR (GPT-3.5) $p = 1\%$
Exact Solve	59/100	63/100	65/100	65/100	66/100
Almost Solve	7/100	6/100	9/100	12/100	13/100
Close	16/100	13/100	14/100	11/100	9/100
Not Close	18/100	18/100	12/100	13/100	13/100

Table 2: Evaluation results on Feynman dataset by cascading LASR's LLM backbone (llama3-8b, gpt-3.5-turbo) and changing the probability of calling the model (p = [0.01, 0.05, 0.10]) in the order of increasing concept guidance. LASR outperforms PySR even with minimal concept guidance using an open-source LLM.

PySR	LaSR (Llama3-8B, 0.1%)
0.070	**0.874**

Table 3: Evaluation results of data leakage. We present the test set R^2 of PySR and of LASR on a synthetic Symbolic Regression dataset. Higher R^2 is better.

tics. Hence, sovling such equations hinges on the language guidance components.

We validate that PySR cannot solve these equations in 400 iterations and run LASR with Llama3-8B at 0.1%. We then compare our synthesized program's test set R^2 with that of PySR's. We justify using correlation instead of exact-match as we are not motivated by the application of results for scientific discovery in this experiment. Our results are summarized in Table 3 and show that LASR's concept-guided synthesis still provides a considerable performance boost compared to PySR – demonstrating that LASR's performance gains are not rooted in memorized responses and that LASR can learn low-level mathematical structures present in a novel synthetic domain.

5. Related Work

Symbolic Regression. The field SR started in the 1970s (Gerwin, 1974; Langley, 1977) and has recently become a prominent approach to AI-for-science (Makke & Chawla, 2024; Merler et al., 2024; Romera-Paredes et al., 2024). Two algorithmic themes here are:

Non-parametric Algorithms: Most work on SR focuses on improving search efficiency using heuristics or parallelization. Specifically, PySR (Cranmer, 2023) builds a multi-population evolutionary algorithm that incorporates various preexisting heuristics (Real et al., 2019), and introduces novel ones such as simulated annealing, an evolve-simplify-optimize loop, and an adaptive parsimony metric. PySR has been successfully applied to study problems in domains such as cosmology (Davis & Jin, 2023), international economics (Verstyuk & Douglas, 2022), and climate modeling (Grundner et al., 2024). Our algorithm extends PySR to enable the discovery of latent concepts.

Parametric Algorithms: Recent work on SR and program synthesis has often used neural networks to accelerate search (Shah et al., 2020; Romera-Paredes et al., 2024; Petersen et al., 2019; Landajuela et al., 2022; Merler et al., 2024; Devlin et al., 2017). The interplay between the neural and the symbolic components in these works can be abstracted into two categories: (1) leveraging LLMs to induce program scaffolds (Merler et al., 2024; Romera-Paredes et al., 2024), and (2) learning a neural policy to accelerate search (Petersen et al., 2019; Landajuela et al., 2022; Shah et al., 2020; Devlin et al., 2017). We highlight two methods from the first category: Funsearch (Romera-Paredes et al., 2024) and LLM-SR (Shojaee et al., 2024). Funsearch (Romera-Paredes et al., 2024) uses a pretrained LLM to implement a mutation operator on a database of executable programs under a fixed specification to find super-optimized programs in extremal combinatorics. LASR is a generalization of FunSearch: while FunSearch conditions program generation on a static "specification" (analogous to our concept library), we discover the concept library in the course of the algorithm. As for LLM-SR (Shojaee et al., 2024), it leverages a pretrained LLM for generating program sketches (Murali et al., 2018). The sketch parameters are optimized and cached in a database which is in turn used to generate new sketches. Our work is an orthogonal direction of improvement. It is technically possible to "plug" the LLM-SR framework into LASR and use our generated ideas to guide the lower-level search component.

The second category includes methods like DSR (Petersen et al., 2019), which, just like LASR, frame SR as a sequence modeling problem. However, the search in LASR leverages a learned concept library and the language and code biases in LLMs, instead of relying on amortization alone.

Program Synthesis with Foundation Models. Recent work in program synthesis models program generation as a sequence prediction problem. Under this paradigm, the DSL

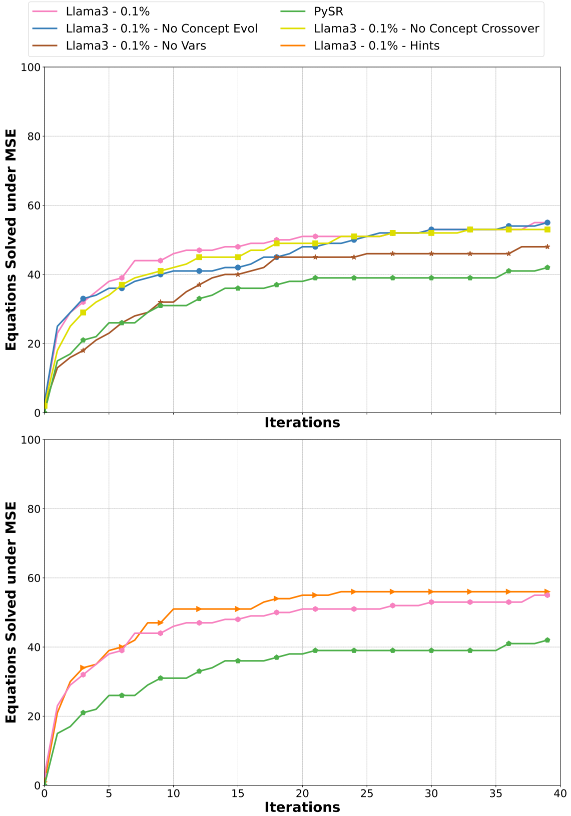


Figure 3: Evaluation results for ablations/extensions of LaSR. **Top**: We ablate three components of LaSR: Concept Evolution, Concept Crossover, and variable names and evaluate their MSE solve rate performance on the Feynman dataset over 40 iterations. We find that each component contributes to accelerating search at different stages in the search process. **Bottom**: We find that providing LaSR an initial concept library $\mathcal{C}_0$ in the form of user provided hints significantly increases the speed of solving equations.

and the input-output specification is serialized in the prompt and a code-generation foundation model (Li et al., 2023; Chen et al., 2021a; Brown et al., 2020) is leveraged to autoregressively generate candidate programs. This approach has been used to successfully synthesize programs in many areas including spreadsheet formula prediction (Devlin et al., 2017; Chen et al., 2021b), competitive programming (Li et al., 2022), and visual programming (Surís et al., 2023; Gupta & Kembhavi, 2023). LaSR is similar to work in this area in that the LLM Mutate, LLM Crossover, and LLM Initialization functions all follow the sequence prediction paradigm to synthesize mathematical equations, relying on guidance from the concept library.

Program Synthesis with Library Learning. Deploying classical program synthesizers in a new domain often necessitate hand-engineering DSLs to enable scalable synthesis. This severely limits the generality and practicality of such methods. An emerging direction of research – called library learning – attempts to learn the DSL and the programs simultaneously (Ellis et al., 2020; Bowers et al., 2023; Grand et al., 2023; Lake et al., 2015; Wong et al., 2021; Ellis et al., 2018; Shin et al., 2019; Zelikman et al., 2023). This is typically framed as a hierarchical Bayesian optimization problem over the space of programs and the space of library functions that generate those programs. Notably, (Grand et al., 2023) uses LLM guidance to assist in program induction and in auto-documenting learned library modules and (Wong et al., 2021) considers learning programs under a latent distribution over the space of natural language and the space of the DSL. LaSR shares a similar problem formulation to these works, but optimizes over the space of programs and over the space of natural language descriptions of these programs.

6. Conclusion

We have presented LaSR, a framework that uses zero-shot queries to an LLM to induce abstract, reusable concepts that can be used to accelerate SR. We have shown that LaSR outperforms state-of-the-art approaches on the standard Feynman equation task.

A key benefit of LaSR is that its capabilities are ultimately bottlenecked by those of the underlying LLM. LLMs are rapidly gaining capability and getting cheaper, and future versions of LaSR should be able to tap into this progress.

Many directions of research remain open. First, our strategy of accelerating evolutionary search with LLM-based concept induction may be applicable beyond the SR setting. Future research should explore such applications. Second, while our approach here was entirely based on in-context learning, it is worth exploring if finetuning improves the performance of the LLM. Finally, we evaluated the learned concept library exclusively on the downstream SR task. However, the library may also be valuable in other tasks such as clustering or explanation synthesis. Exploring these other tasks is an attractive topic for future work.

Limitations. The current instantiation of LaSR has several limitations. First, it cannot guarantee that the concepts it learns are correct or insightful. Even a concept that leads to strong performance in downstream SR tasks may do so because of quirks of the model and data, and end up misleading scientists using the method in a discovery process. Also, we do not currently have a way to ensure that the learned concepts are mutually consistent. Finally, our evaluation here was constrained by our compute budgets for LLMs and search. Whether the trends we see generalize to higher-compute regimes remains to be seen.

References

AI@Meta. Llama 3 model card. 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

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

Bowers, M., Olausson, T. X., Wong, L., Grand, G., Tenenbaum, J. B., Ellis, K., and Solar-Lezama, A. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chaudhuri, S., Ellis, K., Polozov, O., Singh, R., Solar-Lezama, A., Yue, Y., et al. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7 (3):158–243, 2021.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Chen, X., Maniatis, P., Singh, R., Sutton, C., Dai, H., Lin, M., and Zhou, D. Spreadsheetcoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*, pp. 1661–1672. PMLR, 2021b.

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

Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. In *Neural Information Processing Systems*, 2020.

Davis, B. L. and Jin, Z. Discovery of a planar black hole mass scaling relation for spiral galaxies. *The Astrophysical Journal Letters*, 956(1):L22, 2023.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A., and Kohli, P. Robustfill: Neural program learning under noisy I/O. In *ICML*, 2017.

Ellis, K., Morales, L., Sablé-Meyer, M., Solar-Lezama, A., and Tenenbaum, J. Learning libraries of subroutines for neurally–guided Bayesian program induction. In *Advances in Neural Information Processing Systems*, pp. 7805–7815, 2018.

Ellis, K., Wong, C., Nye, M., Sable-Meyer, M., Cary, L., Morales, L., Hewitt, L., Solar-Lezama, A., and Tenenbaum, J. B. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020.

Gerwin, D. Information processing, data inferences, and scientific generalization. *Behavioral Science*, 19(5):314–325, 1974.

Grand, G., Wong, L., Bowers, M., Olausson, T. X., Liu, M., Tenenbaum, J. B., and Andreas, J. Lilo: Learning interpretable libraries by compressing and documenting code. *arXiv preprint arXiv:2310.19791*, 2023.

Grundner, A., Beucler, T., Gentine, P., and Eyring, V. Data-driven equation discovery of a cloud cover parameterization. *Journal of Advances in Modeling Earth Systems*, 16 (3):e2023MS003763, 2024.

Gupta, T. and Kembhavi, A. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023.

Hernandez, A., Balasubramanian, A., Yuan, F., Mason, S. A., and Mueller, T. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *npj Computational Materials*, 5(1):112, 2019.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. ISBN 978-0-321-47617-3.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

La Cava, W., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Landajuela, M., Lee, C. S., Yang, J., Glatt, R., Santiago, C. P., Aravena, I., Mundhenk, T., Mulcahy, G., and Petersen, B. K. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.

Langley, P. Bacon: A production system that discovers empirical laws. In *International Joint Conference on Artificial Intelligence*, 1977. URL `https://api.semanticscholar.org/CorpusID:2320342`.

Lemos, P., Jeffrey, N., Cranmer, M., Ho, S., and Battaglia, P. Rediscovering orbital mechanics with machine learning. *Machine Learning: Science and Technology*, 4(4):045002, 2023.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

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

Merler, M., Dainese, N., and Haitsiukevich, K. In-context symbolic regression: Leveraging language models for function discovery. *arXiv preprint arXiv:2404.19094*, 2024.

Murali, V., Qi, L., Chaudhuri, S., and Jermaine, C. Neural sketch learning for conditional program generation. *ICLR*, 2018.

Organization, D. S. O. Srbench symbolic solution. `https://github.com/dso-org/deep-symbolic-optimization/blob/master/images/srbench_symbolic-solution.png`. Accessed: 2024-05-22.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

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

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

Schmidt, M. D. and Lipson, H. Age-fitness pareto optimization. In *Annual Conference on Genetic and Evolutionary Computation*, 2010. URL `https://api.semanticscholar.org/CorpusID:9975416`.

Shah, A., Zhan, E., Sun, J. J., Verma, A., Yue, Y., and Chaudhuri, S. Learning differentiable programs with admissible neural heuristics. In *Advances in Neural Information Processing Systems*, 2020.

Shin, R., Allamanis, M., Brockschmidt, M., and Polozov, O. Program synthesis and semantic parsing with learned code idioms. In *Advances in Neural Information Processing Systems*, pp. 10825–10835, 2019.

Shojaee, P., Meidani, K., Gupta, S., Farimani, A. B., and Reddy, C. K. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

Stephens, T. gplearn: Genetic programming in python, with a scikit-learn inspired api, 2024. URL `https://github.com/trevorstephens/gplearn`. Accessed: 2024-05-22.

Surís, D., Menon, S., and Vondrick, C. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Verstyuk, S. and Douglas, M. R. Machine learning the gravity equation for international trade. *Available at SSRN 4053795*, 2022.

Virgolin, M., Wang, Z., Alderliesten, T., and Bosman, P. A. Machine learning for the prediction of pseudorealistic pediatric abdominal phantoms for radiation dose reconstruction. *Journal of Medical Imaging*, 7(4):046501–046501, 2020.

Wigner, E. P. The unreasonable effectiveness of mathematics in the natural sciences. In *Mathematics and science*, pp. 291–306. World Scientific, 1990.

Wong, C., Ellis, K. M., Tenenbaum, J., and Andreas, J. Leveraging language to learn program abstractions and search heuristics. In *International conference on machine learning*, pp. 11193–11204. PMLR, 2021.

Zelikman, E., Huang, Q., Poesia, G., Goodman, N., and Haber, N. Parsel: Algorithmic reasoning with language models by composing decompositions. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=qd9qcbVAwQ`.

A. Appendix

A.1. Broader Societal Impacts

We have presented LASR: a symbolic regression framework that leverages concept guidance to accelerate symbolic regression. We hope that LASR helps accelerate the search for empirical laws in the broader scientific community. In this section, we discuss the broader societal impacts and ethical considerations of our work.

Potential for Misuse: As with other ML techniques, symbolic regression can be leveraged by bad actors to inflict societal harm. Our experiments show that LASR accelerates the search for empirical laws from raw observations. In our setting, we are restricted to observations about physical phenomena. However, a malicious actor could misuse LASR to find patterns in datasets that violate personal rights.

Privacy Concerns: As mentioned before, LASR enables finding patterns in raw observations. We hope that LASR is leveraged by scientists to explain physical phenomena. However, it is possible to use such models to learn behavioral profiles without the active knowledge or explicit consent of the subjects.

Bias and Fairness: LASR generates two artifacts: a hypothesis that maximizes a fitness function (represented as an equation) and a library of concepts that helped discover that hypothesis. LASR ensures fairness and lack of bias in the generated equation as long as the fitness function is free of biases as well. However, we leverage foundation models to induce our library of concepts which could be trained on biased data which may reflect in our concept library. Furthermore, we cannot directly evaluate the efficacy of the concept library and its factual correctness. This doesn't affect equation generation – since equations are quantitatively evaluated. However, a human analyzing the concepts LASR learns might misinterpret trends that the model picks up on.

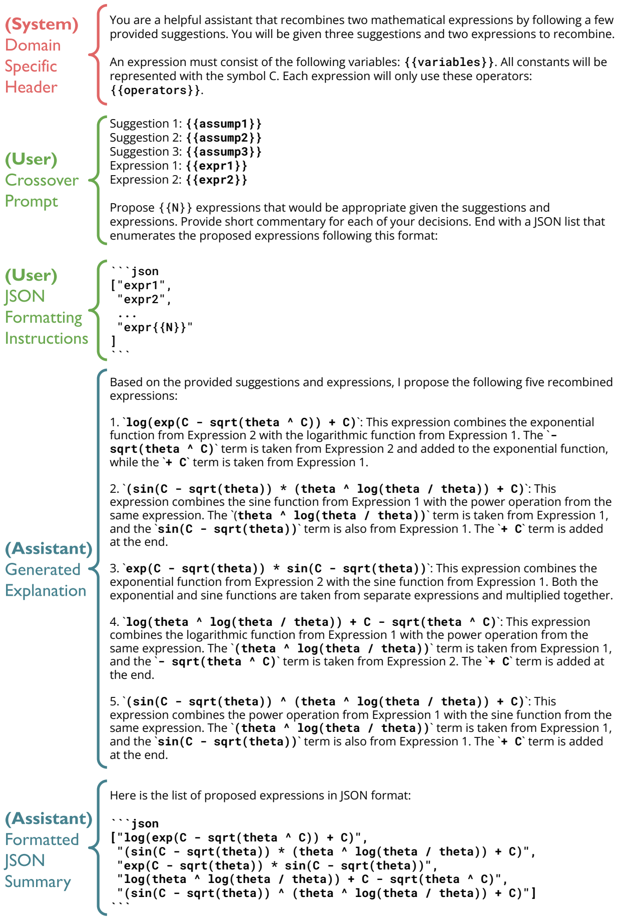
A.2. LLM Prompts

Note that in the prompts in Figures 4 and 5, we refer to our hypothesis as expressions and the concepts as hypotheses and suggestions. This prompting style was found to work best for the LLM.

A.3. Implementation Details

A.3.1. COMPUTE USAGE

We run all experiments on a server node with 8xA100 GPUs with 80 GB of VRAM each. However, our experiments can be reproduced with a GPU with 16 GB of VRAM. We were even able to run LASR on a laptop utilizing a quantized model hosted locally [2]. Moreover, certain models are hosted on external servers (such as `gpt-3-turbo-0125`) which allows running LASR on machines without GPUs. For this project, we chose to run `llama3-8b` using vLLM (Kwon et al., 2023). However, our framework is compatible with any LLM inference framework that allows hosting an OpenAI compliant RESTful server. For reference, each iteration makes around $60,000$ calls. Each call to the LLM is just under 1000 tokens. This gives an upper bound on total compute of $60,000,000$ tokens per iteration if $p = 100\%$. Hence, running our model at $p = 1\%$ for 40 iterations would result in just under 25M tokens for each equation.


(System)
Domain
Specific
Header

You are a helpful assistant that recombines two mathematical expressions by following a few provided suggestions. You will be given three suggestions and two expressions to recombine.

An expression must consist of the following variables: **{{variables}}**. All constants will be represented with the symbol C. Each expression will only use these operators: **{{operators}}**.

Suggestion 1: **{{assump1}}**
Suggestion 2: **{{assump2}}**
Suggestion 3: **{{assump3}}**
Expression 1: **{{expr1}}**
Expression 2: **{{expr2}}**

Propose {{N}} expressions that would be appropriate given the suggestions and expressions. Provide short commentary for each of your decisions. End with a JSON list that enumerates the proposed expressions following this format:

~~~
```json
["expr1",
 "expr2",
 ...
 "expr{{N}}"
]
```
~~~

Based on the provided suggestions and expressions, I propose the following five recombined expressions:

1. `log(exp(C - sqrt(theta ^ C)) + C)`: This expression combines the exponential function from Expression 2 with the logarithmic function from Expression 1. The `- sqrt(theta ^ C)` term is taken from Expression 2 and added to the exponential function, while the `+ C` term is taken from Expression 1.

2. `(sin(C - sqrt(theta)) * (theta ^ log(theta / theta)) + C)`: This expression combines the sine function from Expression 1 with the power operation from the same expression. The `(theta ^ log(theta / theta))` term is taken from Expression 1, and the `sin(C - sqrt(theta))` term is also from Expression 1. The `+ C` term is added at the end.

3. `exp(C - sqrt(theta)) * sin(C - sqrt(theta))`: This expression combines the exponential function from Expression 2 with the sine function from Expression 1. Both the exponential and sine functions are taken from separate expressions and multiplied together.

4. `log(theta ^ log(theta / theta)) + C - sqrt(theta ^ C)`: This expression combines the logarithmic function from Expression 1 with the power operation from the same expression. The `(theta ^ log(theta / theta))` term is taken from Expression 1, and the `- sqrt(theta ^ C)` term is taken from Expression 2. The `+ C` term is added at the end.

5. `(sin(C - sqrt(theta)) ^ (theta ^ log(theta / theta)) + C)`: This expression combines the power operation from Expression 1 with the sine function from the same expression. The `(theta ^ log(theta / theta))` term is taken from Expression 1, and the `sin(C - sqrt(theta))` term is also from Expression 1. The `+ C` term is added at the end.

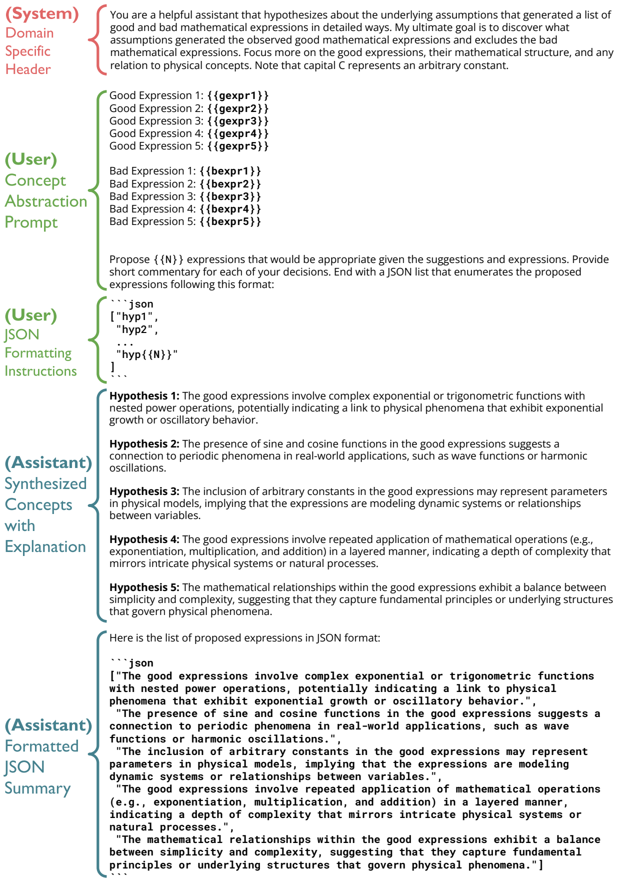
Here is the list of proposed expressions in JSON format:

~~~
```json
["log(exp(C - sqrt(theta ^ C)) + C)",
 "(sin(C - sqrt(theta)) * (theta ^ log(theta / theta)) + C)",
 "exp(C - sqrt(theta)) * sin(C - sqrt(theta))",
 "log(theta ^ log(theta / theta)) + C - sqrt(theta ^ C)",
 "(sin(C - sqrt(theta)) ^ (theta ^ log(theta / theta)) + C)"]
```
~~~

Figure 4: LLMCROSSOVER prompt with an example output. LLMMUTATION and LLMINIT follow the same structure but with slightly different wording and with one and no reference expressions, respectively. Variables within double braces are replaced with the instance specific arguments. These prompts are available in `prompts/*.txt` in the linked repository.

---

[2] `TheBloke/Mistral-7B-Instruct-v0.2-GGUF` using `llama.cpp`


(System)
Domain
Specific
Header

You are a helpful assistant that hypothesizes about the underlying assumptions that generated a list of good and bad mathematical expressions in detailed ways. My ultimate goal is to discover what assumptions generated the observed good mathematical expressions and excludes the bad mathematical expressions. Focus more on the good expressions, their mathematical structure, and any relation to physical concepts. Note that capital C represents an arbitrary constant.

Good Expression 1: **{{gexpr1}}**
Good Expression 2: **{{gexpr2}}**
Good Expression 3: **{{gexpr3}}**
Good Expression 4: **{{gexpr4}}**
Good Expression 5: **{{gexpr5}}**

Bad Expression 1: **{{bexpr1}}**
Bad Expression 2: **{{bexpr2}}**
Bad Expression 3: **{{bexpr3}}**
Bad Expression 4: **{{bexpr4}}**
Bad Expression 5: **{{bexpr5}}**

Propose {{N}} expressions that would be appropriate given the suggestions and expressions. Provide short commentary for each of your decisions. End with a JSON list that enumerates the proposed expressions following this format:

~~~
```json
["hyp1",
 "hyp2",
 ...
 "hyp{{N}}"
]
```
~~~

Hypothesis 1: The good expressions involve complex exponential or trigonometric functions with nested power operations, potentially indicating a link to physical phenomena that exhibit exponential growth or oscillatory behavior.

Hypothesis 2: The presence of sine and cosine functions in the good expressions suggests a connection to periodic phenomena in real-world applications, such as wave functions or harmonic oscillations.

Hypothesis 3: The inclusion of arbitrary constants in the good expressions may represent parameters in physical models, implying that the expressions are modeling dynamic systems or relationships between variables.

Hypothesis 4: The good expressions involve repeated application of mathematical operations (e.g., exponentiation, multiplication, and addition) in a layered manner, indicating a depth of complexity that mirrors intricate physical systems or natural processes.

Hypothesis 5: The mathematical relationships within the good expressions exhibit a balance between simplicity and complexity, suggesting that they capture fundamental principles or underlying structures that govern physical phenomena.

Here is the list of proposed expressions in JSON format:

~~~
```json
["The good expressions involve complex exponential or trigonometric functions
with nested power operations, potentially indicating a link to physical
phenomena that exhibit exponential growth or oscillatory behavior.",
 "The presence of sine and cosine functions in the good expressions suggests a
connection to periodic phenomena in real-world applications, such as wave
functions or harmonic oscillations.",
 "The inclusion of arbitrary constants in the good expressions may represent
parameters in physical models, implying that the expressions are modeling
dynamic systems or relationships between variables.",
 "The good expressions involve repeated application of mathematical operations
(e.g., exponentiation, multiplication, and addition) in a layered manner,
indicating a depth of complexity that mirrors intricate physical systems or
natural processes.",
 "The mathematical relationships within the good expressions exhibit a balance
between simplicity and complexity, suggesting that they capture fundamental
principles or underlying structures that govern physical phenomena."]
```
~~~

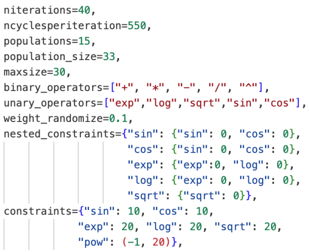
Figure 5: LLM Concept Abstraction prompt with an example output. The LLM Concept Crossover function follows a similar structure with a modified task description for crossover on concepts.

A.3.2. CONCEPT SAMPLING

In order to determine which concepts from the concept library we sample for the LLM Hypothesis Evolution, we randomly choose the top-K most recent concepts in the library. This ensures that we use the latest concepts, which are generally reflective of more informed hypotheses, and thus better to use. In practice, we set $K = 20$. Additionally, for Concept Evolution, we exclude the top-K most recent concepts from being used, and rather use older concepts. This is motivated by the desire to not have the concept library converge on a few ideas, rather we want diversity of thought. Our concepts are intended to be longer lasting than the hypotheses that generated them, similar to how observational data comes and goes, but the conclusions from them are more persistent.

A.3.3. HYPERPARAMETERS

Figure 6 showcases the hyperparameters used for all our experiments. Wherever possible, we use the default PySR parameters. Additionally, LASR introduces three new hyperparameters: (1) % of LLM calls, (2) List of user hints, and (3) a dictionary of parameters pertaining to backend LLM communication. Following other methods in SRBench, we utilize only a subset of the necessary operators for solving the Feynman equations, excluding special operators like $\arcsin$ and $\arctan$. These operators are seldom required, and removing them speeds up the search process. We generally set the number of iterations to 40. However, certain experiments may demand more or less iterations.



```
niterations=40,
ncyclesperiteration=550,
populations=15,
population_size=33,
maxsize=30,
binary_operators=["+", "*", "-", "/", "^"],
unary_operators=["exp","log","sqrt","sin","cos"],
weight_randomize=0.1,
nested_constraints={"sin": {"sin": 0, "cos": 0},
                    "cos": {"sin": 0, "cos": 0},
                    "exp": {"exp":0, "log": 0},
                    "log": {"exp": 0, "log": 0},
                    "sqrt": {"sqrt": 0}},
constraints={"sin": 10, "cos": 10,
             "exp": 20, "log": 20, "sqrt": 20,
             "pow": (-1, 20)},
```

Figure 6: The PySR hyperparameters used all experiments. Whenever possible, we use the default PySR parameters. Other hyperparameters are in `pysr_feynman.py` in the linked repository.

A.4. Dataset Details

A.4.1. FEYNMAN EQUATIONS

For the Feynman dataset, we took the equations and the bounds at which each variable was sampled at and generated our dataset. Then, we added additional noise of 0.001 to our target variable, following the noise formula detailed in the Appendix A.4 of (La Cava et al., 2021), as well as additional random noise variables with arbitrary names to force the model for proper feature selection. We then evaluate exact matches by looking at if the predicted equation symbolically simplifies into the ground truth equation. For the ablation graphs, we used the PySR hyperparameter "early_stop_condition" to check if there is a "solution" after N iterations.

A.4.2. SYNTHETIC DATASET

For the synthetic dataset, we ran a script that generates uncommon mathematical hypotheses that satisfy our constraints at random. Then, we ran PySR for 400 iterations and found all the equations that PySR performed poorly in, i.e. MSE loss greater than 1, while having a complexity less than 20. For these 41 remaining equations, we then compared LASR and PySR after 20 iterations using the average of their test set R^2 for each hypothesis.

A.5. Further Qualitative Analysis

LASR generates two artifacts: the best fit program, and the library of natural language concept that helped find that program. These artifacts provide a unique window into the inner workings of LASR. This section goes over a qualitative study of how LASR and PySR go about discovering Coulomb's law $F = \frac{q_1 q_2}{4\pi r^2 \epsilon}$ from data. Both methods are able to find an answer to this equation. However, their approach to finding the best fit equation as well as the form of the equation they discover differs significantly.

Setup: Coulomb's law is equation #10 in the Feynman equation dataset. It describes how the force between two point charges changes with respect to the distance between the charges, the magnitudes of the charges, and the permittivity of free space constant. The corresponding data for this equation has a target noise of 0.001 to simulate experimental errors.

By analyzing the form of the equation and relationships between variables in Coulomb's law, we can uncover several interesting properties: First, observe that this is an inverse square law (The force F varies inversely with the square of the distance r between the charged particles). Second, notice that the F is directly proportional to the magnitude of the charges q_1 and q_2. Third, observe that the resultant force is symmetric with respect to the magnitude of the charged particles (i.e.: The magnitude of the F doesn't change if the magnitude of the charged particles is swapped).

PySR Solution: PySR finds the following solution to this equation:

$$F = \frac{\left(\left(\left(\left(\left(\left(\left(\frac{q_2 \cdot 3.382}{r}\right) - \left(\frac{\sin\left(\frac{0.017}{\exp(B)}\right)}{\exp(C)}\right)\right)\right)/0.712\right) \cdot q_1\right) \cdot 0.087\right)/\epsilon\right) \cdot 0.191\right)}{r}$$

This equation has a complexity of 26 and achieves a loss of 2.191505×10^{-12} on the dataset. Obtaining a simplification of this solution is rather painstaking.

LASR's Solution: LASR finds the following solution to this equation. We also present three steps of simplification:

$$\begin{aligned}
F &= \frac{q_1}{\left(\frac{r}{q_2}\right)\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot 0.07957782 \\
&= \frac{q_1}{\left(\frac{r}{q_2}\right)\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot \frac{1}{4\pi} \\
&\qquad\qquad\qquad\qquad \text{(Substitute constant)} \\
&= \frac{q_1 q_2}{r\left(r + \frac{1.9181636\times10^{-5}}{q_2}\right)\epsilon} \cdot \frac{1}{4\pi} \\
&\qquad\qquad\qquad\qquad \text{(Simplify denominator)} \\
&\approx \frac{q_1 q_2}{r\,(r)\,\epsilon} \cdot \frac{1}{4\pi} \qquad \text{(Negligible. } \frac{1.9181636\times10^{-5}}{q_2} \approx 0)
\end{aligned}$$

This equation has a complexity of 15 and achieves a much lower loss of $4.6709058 \times 10^{-14}$ on the accompanying dataset. We can see with just three steps of simplification how this equation might be reduced to the ground truth.

Let's examine some essential concepts from various iterations in the search process. Keep in mind that an LLM operates on *tokens* in each concept. Consequently, even small relevant substrings can positively influence future LLM inference calls, despite full concepts appearing verbose to humans.

1. **Iteration 2** *The good mathematical expressions exhibit a clear and coherent relationship between the variables involved, with a focus on power functions and trigonometric functions that can be easily related to physical concepts.*

2. **Iteration 6** *The good mathematical expressions exhibit a certain level of symmetry or regularity in their form, possibly reflecting underlying patterns or relationships between the variables and constants.*

3. **Iteration 24**: *The good mathematical expressions have a clear and consistent structure involving the variables q1, q2, epsilon, C, and r, with a specific pattern of division and multiplication.*