InceptionSR: Recursive Symbolic Regression for Equation Synthesis

Edward Gu[1], Simon Alford[1], Omar Costilla-Reyes[2], Miles Cranmer[3], Kevin Ellis[1]

[1]Cornell University
[2]Massachusetts Institute of Technology
[3]University of Cambridge

elg227@cornell.edu, sca63@cornell.edu, costilla@mit.edu, mc2473@cam.ac.uk, kellis@cornell.edu

Abstract

Symbolic regression (SR) algorithms generate equations that fit a dataset. We propose a symbolic regression variant which we call InceptionSR. Motivated by gradient boosting, our algorithm iteratively runs a base SR algorithm, freezes partial equations, and seeds the next round of SR using those frozen equations as features. Our algorithm can also be viewed as a simple form of "library learning", a technique common in the related field of program synthesis. We evaluate our algorithm on toy problems as well as a real world scientific discovery problem of generating equations to predict planetary instability. Results show that our method consistently improves SR performance, and may also generate more interpretable equations. **Keywords:** symbolic regression; equation discovery; library learning, recursion; boosting

Introduction

The discovery of mathematical equations that describe physical phenomena has been foundational to scientific progress. While neural networks and other machine learning approaches have achieved impressive predictive performance across many domains, they often lack interpretability and fail to provide scientific insights in the form of closed-form mathematical expressions. Symbolic regression (SR) algorithms address this limitation by automatically discovering interpretable equations from data, offering a promising approach for scientific discovery that combines the automation of machine learning with the interpretability of analytical expressions. The field of symbolic regression has seen significant advances since its inception in the 1970s, with methods ranging from evolutionary algorithms to neural network-based approaches (Udrescu and Tegmark 2020; Schmidt and Lipson 2009; Makke and Chawla 2023; Fong, Wongso, and Motani 2023). PySR, a recent open-source symbolic regression system, has emerged as a powerful tool for equation discovery in scientific applications (Cranmer 2023). PySR employs a multi-objective optimization approach, simultaneously minimizing prediction error and equation complexity while maintaining a Pareto front of candidate expressions. This allows scientists to explore the trade-off between predictive accuracy and interpretability - a crucial consideration in scientific applications where understanding the underlying relationships is as important as predictive performance.

In this paper, we propose InceptionSR, a symbolic regression algorithm that builds upon PySR by incorporating ideas from gradient boosting and library learning. Motivated by the observation that complex equations often contain reusable subexpressions, our algorithm iteratively runs symbolic regression, freezes promising partial equations, and uses these frozen expressions as features in subsequent iterations. This approach shares similarities with gradient boosting (Friedman 2001), where successive models are trained on residuals, but differs in that we maintain interpretability by working with symbolic expressions rather than black-box predictors. Our method can also be viewed as a form of automated library learning, where useful mathematical building blocks are discovered and reused throughout the search process (Ellis et al. 2020; Bowers et al. 2023).

We evaluate InceptionSR on both a synthetic dataset, rediscovering Heron's formula, and a challenging real-world problem: predicting the instability time of planetary systems. The dynamics of multi-planet systems represent a complex n-body problem that has resisted analytical characterization for systems involving more than two planets (Tamayo et al. 2020; Petit, Laskar, and Boué 2018; Petit et al. 2020). While recent work (Cranmer et al. 2021) has shown success using neural networks to predict planetary instability, these models remain black boxes that offer limited scientific insight. By applying InceptionSR to this problem, we aim to discover interpretable equations that not only predict instability but also advance our understanding of the underlying physical mechanisms.

Our work demonstrates that recursive symbolic regression can consistently outperform vanilla PySR in terms of both prediction accuracy and equation interpretability. By breaking down complex relationships into simpler, reusable components, InceptionSR makes progress toward automated scientific discovery that yields not just predictive models, but mathematical insights that can inform theory development.

Our work makes the following two contributions:

1. We introduce InceptionSR, a recursive symbolic regression algorithm which outperforms traditional approaches by iteratively discovering and reusing subexpressions.
2. We show that InceptionSR's core principle of expression reuse may be applied with other methods of library learn-

ing (Stitch), creating powerful hybrid symbolic regression methods for equation synthesis.

Related Work

Symbolic regression. Research into methods for symbolic regression — discovering of equations that fit data — goes back to the 1970's, with increasing popularity and applicability over time (Gerwin 1974; Udrescu and Tegmark 2020; Langley 1977; Schmidt and Lipson 2009). Symbolic regression algorithms often use evolutionary search, but more recently, a wide variety of techniques have been used. (Cranmer et al. 2020) extract physical laws from a neural network trained on the data. (Rivero, Fernandez-Blanco, and Pazos 2022) deterministically grows expressions to fit data, avoiding the need for evolutionary search. (Holt, Qian, and van der Schaar 2023) uses deep generative modeling to discover equations. (Fong, Wongso, and Motani 2023) improves robustness of equations by alternating fitness functions used in evolutionary search over succcessive generations. For a review of using symbolic regression, see (Makke and Chawla 2023).

We build our algorithm using PySR as a starting block (Cranmer 2023). PySR is a popular open source symbolic regression algorithm that uses an evolutionary algorithm to discover equations. In principle, our approach can interface with any symbolic regression algorithm that outputs a Pareto frontier of the best equation at each complexity.

Gradient boosting and library learning. Our proposed algorithm shares similarities to the technique of gradient boosting (Friedman 2001). InceptionSR also takes inspiration from an area of research closely related to symbolic regression: program synthesis, searching or otherwise synthesizing programs that satisfy a specification (Gulwani, Polozov, and Singh 2017). In particular, we take inspiration from the technique of *library learning*, where a synthesized program is constructed out of a library of interpretable subprograms discovered during search (Ellis et al. 2020; Bowers et al. 2023; Grayeli et al. 2024).

Gradient boosting for symbolic regression. (Sipper and Moore 2021) shares some similarities to the present work. In it, the authors propose using gradient boosting directly for symbolic regression, using a symbolic regression algorithm as the weak learner. Our algorithm instead reruns symbolic regression for multiple iterations by providing previous rounds' discovered equations as frozen feature inputs to the next round. This allows greater flexibility in how the residuals are defined and how previous equations are used in the next round's prediction. By comparing the performance of equations at equal levels of complexity, our method also provides a fair comparison. (Sipper and Moore 2021), in contrast, achieve higher performance through boosting by increasing the final complexity of the discovered equations compared to no boosting.

Predicting planetary instability. Our work continues research of prior work building models to predict the instability of planetary systems (Tamayo et al. 2020; Cranmer et al. 2021). These prior works seek to shed light on the long studied problem (Chambers, Wetherill, and Boss 1996; Petit et al. 2020; Obertas, Van Laerhoven, and Tamayo 2017) of understanding dynamics behind the evolution of planetary systems by building machine learning models that predict the instability of a system given features of the planetary system configuration as inputs. By applying symbolic regression to this problem, we hope to discover equations for predicting instability which are more interpretable than machine learning models, potentially leading to new insights into the area of research. (Alford et al., *in prep.*) discover equations for predicting instability by distilling a modified form of the neural network from (Cranmer et al. 2021).

Methods

InceptionSR Algorithm

Our approach leverages a black-box symbolic regression system that produces a Pareto frontier of equations that optimally trades off between expression size (complexity) and data fit. PySR is a multi-population evolutionary algorithm that learns optimal mathematical equations approximating the relationship between a dataset of features and a target. It uses an evolve-simplify-optimize loop in order to discover expressions containing unknown scalar constants. This loop is comprised of the evolve phase, where multiple rounds of tournament selection, mutations, and crossovers are performed; the simplify phase, where algebraic simplification rules are applied to reduce expression complexity; and the optimize phase, which optimizes numerical constants and parameters with respect to the loss function. It is important to note that complexity in PySR is defined as "the number of nodes in an expression tree, regardless of each node's content" (Cranmer 2023). The design of PySR considers the challenges of symbolic regression in scientific discovery problems, which makes it an ideal candidate for our algorithm.

There are several key differences between vanilla PySR and InceptionSR. Firstly, vanilla PySR is a single-pass learning algorithm where all expressions are discovered in one evolutionary process, whereas InceptionSR is a multi-pass learning algorithm where each pass discovers new relationships using previously discovered patterns. This is similar to how human scientists build new theories upon established mathematical relationships between variables. Secondly, vanilla PySR maintains a fixed feature space during the evolutionary process, whereas InceptionSR has a dynamic feature space that incorporates derived features, successful expressions from prior runs, alongside the original features. Thirdly, vanilla PySR searches through all possible expressions that may be constructed from the original features and operators, as opposed to InceptionSR which uses a hierarchical search space where each subsequent run searches through an augmented space that includes composite functions. Lastly, only a single complexity measure based on expression tree size is used for vanilla PySR, while InceptionSR allows for the complexity of input features into PySR to be modified. Reused equations from previous runs maintain their original complexities, allowing the algorithm to build upon simpler building blocks while controlling overall expression complexity. We hypothesize that unlike InceptionSR, no matter how long vanilla PySR is run for, it will

Algorithm 1: InceptionSR

0: **function** INCEPTIONSR(X, y, prevModel, targetComplexities)
0: Initialize feature space and tracking variables
0: Load equations from previous model
0: Select equations matching target complexities from previous model
0: **for all** selected equation **do**
0: Evaluate equation on current features
0: Augment feature matrix with new feature and its complexity
0: **end for**
0: Train new symbolic regression model on augmented feature space
0: **return** model
0: **end function**=0

not be able to discover key subexpressions to reuse that make the generated equations most accurate in modeling the data.

We experiment with different sets of equations to reuse, including: top k elbow (k number of equations from the Pareto frontier may be included), maximum complexity (only the highest complexity equation is included, which is around complexity 30), and all (the entire Pareto frontier is included). When we reuse the entire Pareto frontier, we give PySR complete autonomy in figuring out the best subexpressions to use in the new equations. The automated selection process for topk equations is that we choose k number of equations that have the steepest 'elbow' in its Pareto frontier. Generally these are equations with low complexity, containing only 1 to 3 variables and a similar number of constants or coefficients, and compared to other equations they have a relatively steep drop-off in loss from the next lesser complexity equation. These low-complexity equations are more interpretable and are thus reused as a feature for the next iteration of PySR. Another option of equation selection would be to have a domain expert select physically feasible equations they are able to interpret, allowing for more intelligent subexpression selection.

Mathematical Interpretation

Feature Space Construction Vanilla PySR operates on a static feature space $X \in \mathbb{R}^{N \times D}$, where N is the number of samples and D is the dimensionality of the input space. The algorithm searches for functions $f : \mathbb{R}^D \to \mathbb{R}$ within this fixed dimensional space.

In contrast, InceptionSR dynamically expands the feature space through iterative augmentation. At iteration k, the feature space becomes:

$$X_k = [X|\Phi(X)] \in \mathbb{R}^{N \times (D+K)} \tag{1}$$

where $\Phi(X) = [\phi_1(X), \phi_2(X), ..., \phi_K(X)]$ represents the evaluation of K selected equations from previous iterations.

Complexity Measure Formulation Vanilla PySR employs a straightforward complexity measure based on expression tree size:

$$C(E) = |V_E| + |O_E| \tag{2}$$

where $|V_E|$ is the number of variables/constants and $|O_E|$ is the number of operators in expression E.

Meanwhile, InceptionSR introduces a hierarchical complexity measure:

$$C_{\text{res}}(E) = \sum_{i=1}^{D} w_i + \sum_{j=1}^{K} C(\phi_j) \tag{3}$$

where $w_i = 1$ for original features and $C(\phi_j)$ is the complexity of reused equation ϕ_j.

Search Space Topology The search space in Vanilla PySR forms a connected graph $G = (V, E)$ where vertices V represent expressions and edges E represent valid mutations or crossovers. The diameter of this graph is bounded by the maximum allowed expression size.

On the other hand, InceptionSR creates a layered search space hierarchy $\{G_k\}_{k=1}^{K}$ where:

$$G_k = (V_k, E_k) \supset G_{k-1} \tag{4}$$

Each layer k incorporates some or all expressions from previous layers as atomic operations, effectively creating shortcuts in the search space.

Fitness Function Construction Vanilla PySR uses a standard regularized fitness function:

$$F(E) = L(E(X), y) + \lambda C(E) \tag{5}$$

where L is a loss function and λ balances accuracy vs complexity.

However, InceptionSR employs a component-wise fitness function:

$$F_{\text{res}}(E) = L(E(X_k), y) + \sum_i w_i C(\text{component}_i(E)) \tag{6}$$

where $w_i = 1$ for original input features, and for reused expressions, w_i equals the complexity of the expression when it was first discovered. This formulation explicitly accounts for the hierarchical nature of reused expressions by maintaining their original complexity costs through weighted component-wise penalties, ensuring that the total complexity accurately reflects both the new operations and the complexity of any reused subexpressions.

Population Dynamics Vanilla PySR maintains independent populations $\{P_i\}_{i=1}^{n_p}$ evolving in parallel with occasional migration. The evolution follows a Markov chain

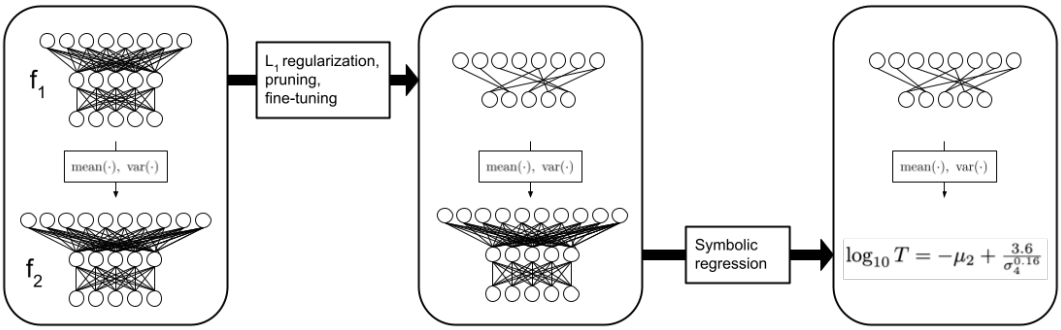

Figure 1: Overview of equation distillation approach from (Alford et al., *in prep.*). We experiment replacing the use of PySR in the second stage of equation distillation with our InceptionSR algorithm.

where the transition probabilities depend only on the current state.

InceptionSR introduces conditional dependencies between successive runs through the augmented feature space. The population dynamics can be described by a conditional probability:

$$P(E_k|\Phi_{k-1}) = P(E_k|X_k) \cdot P(X_k|\Phi_{k-1}) \tag{7}$$

where Φ_{k-1} represents the set of equations selected from previous iterations.

Migration Mechanism Vanilla PySR uses a simple migration probability between its parallel populations:

$$P(\text{migrate}) = \begin{cases} \alpha_h & \text{for Hall of Fame} \\ \alpha_m & \text{for Population Memory} \end{cases} \tag{8}$$

where the Hall of Fame maintains the Pareto frontier of best-performing equations across all runs, and Population Memory preserves diversity by sharing promising solutions between parallel populations.

InceptionSR extends this migration mechanism with equation tracking:

$$P(\text{migrate}, \phi) = P(\text{migrate}) \cdot P(\phi|C(\phi)) \tag{9}$$

where $P(\phi|C(\phi))$ represents the probability of selecting equation ϕ based on its complexity, allowing the algorithm to preferentially migrate equations that balance performance and complexity.

Predicting Instability of Planetary Systems

To show the potential for our algorithm to assist in scientific discovery on a real-world problem, we apply it to the problem of generating equations that predict the time to instability of a planetary system, a long studied problem (Chambers, Wetherill, and Boss 1996; Petit et al. 2020; Obertas, Van Laerhoven, and Tamayo 2017). As a general N-body problem, predicting the dynamics of a planetary system is fundamentally chaotic and unpredictable. While analytic characterization is known for two-planet systems (Petit, Laskar, and Boué 2018; Hadden and Lithwick 2018), the general case of three or more planets is not fully understood (Petit et al. 2020). To shed light on the problem, (Tamayo et al. 2020; Cranmer et al. 2021) build machine learning models to predict instability of a planetary system. Building off these works, concurrent work (Alford et al., *in prep.*) learns equations by distilling a neural network trained to predict instability time based on the architecture of the model from (Cranmer et al. 2021). As part of their approach, they use PySR to find equations that imitate a neural network that predicts the instability time given a set of input features. We experiment with replacing the PySR symbolic regression algorithm as used in their approach with our recursive SR algorithm, and evaluate the extent to which our method can improve the performance and interpretability of the found equations.

Let us briefly overview the techniques of (Alford et al., *in prep.*) for discovering equations for predicting instability. They start with a neural network trained on a dataset of simulations to predict the instability time of a planetary system. The neural network takes as input the orbital elements of the planetary system over 100 timesteps taken across the first 10^4 orbits of the simulation, and predicts a instability time of the system. Then they distill equations for predicting instability from this neural network. The distillation approach first simplifies the neural network architecture and applies regularization and sparsification to the network to extract a transformed set of input features. These features are sparse linear combinations of the original orbital element input features. Second, the approach distills the neural network trained to predict instability given the mean and standard deviation of these transformed features across the input timesteps, using PySR to imitate the neural network with equations. The inputs to PySR are a set of 20 means and variances of the transformed feature set, and the targets are the instability time predicted by the neural network. The binary operators provided to PySR are $+$, $*$, $\div$, $-$, and exponent. The only unary operator included is the sin function, which they deem to be sufficient for this problem. For all training, we use one GPU. See Figure 1 for an overview of their approach.

PySR + Stitch

One of our motivations for our InceptionSR algorithm is the idea that freezing and reusing initially discovered equations

Table 1: Average Pareto Frontier MSE for PySR and InceptionSR

Total Hours	PySR	InceptionSR (All)	InceptionSR (Max Complex)	InceptionSR (Top 5 Elbow)
1	1.764 ± 0.027	-	-	-
2	1.729 ± 0.016	1.691 ± 0.044	1.681 ± 0.045	1.698 ± 0.039

Table 2: % Improvement Compared to PySR (1 Hour)

Method	Improvement (%)
PySR (2 Hours)	1.98 ± 0.02
InceptionSR (All)	4.14 ± 0.11
InceptionSR (Max Complex)	4.71 ± 0.13
InceptionSR (Top 5 Elbow)	3.74 ± 0.09

works as a form of "library learning", where en route to discovering a best-performing equation, we assemble a library of smaller, more interpretable subexpressions out of which the best-performing equation is composed. However, InceptionSR only selects subexpressions which are good approximations of the downstream task. We are also curious as to whether a more sophisticated library, where subexpressions that are reused multiple times in a single equation, or that occur multiple complexities across the Pareto frontier of results, are used as building blocks for the next round of SR. To this end, we experiment integrating Stitch, a Python package for library learning, into our SR algorithm (Bowers et al. 2023). Stitch takes as input a set of lambda calculus programs, and outputs one or more functions chosen to reduce the description length of the set of programs when rewritten with the functions. After running one iteration of search, we pass the Pareto frontier of discovered equations into Stitch, with some processing to convert equation representation into lambda expressions. Then we look at the resulting abstractions discovered, and manually convert them into the most similar corresponding expression. Then we provide that expression as a feature for the next round of symbolic regression.

Results

Predicting Planetary Instability

In Table 1, we compare the performance of vanilla PySR against three variants of InceptionSR by reporting their Mean Squared Error (MSE) on the training set of the planetary instability prediction problem. For each run of PySR and InceptionSR, the losses of equations across the entire Pareto frontier are averaged. To ensure fairness in comparing InceptionSR with PySR, we control for equation complexity and total computational runtime, since both higher complexity and longer runtime improve performance. It is fair to compare the results of running InceptionSR with running PySR for the same amount of total time, which is why we compare two hours of PySR runtime with two hours of InceptionSR runtime. The two hours of InceptionSR runtime is broken into one hour of the first iteration of PySR followed by one hour of the second iteration of PySR with equation reuse. Because InceptionSR involves at least two iterations of running vanilla PySR (so that subexpressions have the chance of being reused), there are no values for running InceptionSR for one hour. The choice of running each iteration of PySR for one hour comes from the original PySR paper (Cranmer 2023). The maximum complexity equation that may be generated in the Pareto frontier is capped to complexity 30 for a total runtime of one hour, complexity 60 for a total runtime of two hours, and complexity 90 for a total runtime of three hours. We double the amount of complexity for each subsequent hour of runtime to observe how reusing previous high-complexity equations would affect the performance of equations in subsequent iterations.

We find that all 3 InceptionSR variants modestly but consistently outperform running PySR in the same amount of time (shown in Table 1). Table 2 demonstrates that all InceptionSR variants achieve greater percentage improvements over 1-hour PySR runs than simply running PySR for an additional hour. This suggests that InceptionSR's recursive approach is more effective than extending PySR's computation time. In Figure 2, we plot the MSE against equation complexity for the average Pareto frontier across 5 seeds of running PySR for 2 hours compared to running InceptionSR, reusing the entire Pareto frontier, for the same amount of time. At almost all complexities, InceptionSR achieves a superior MSE. Similarly in Figure 3, InceptionSR that reuses the 5 equations with the steepest loss drop-off (Top 5 Elbow) outperforms PySR at nearly all complexities as well. For all 5 seeds, the lowest complexity equation reused was 3 and the highest complexity equation reused was 17. Figure 4 shows that when InceptionSR incorporates the complexity-30 equation as a building block, its performance notably diverges from vanilla PySR. At precisely the complexity level where this equation is introduced, InceptionSR begins to achieve lower loss values than PySR for equations of equivalent complexity.

Table 3 compares the MSE of the highest complexity equation from the previous iteration (equation with complexity 30 ± 1), between PySR and InceptionSR variants. It is interesting to note that the longer PySR is run, the worse the MSE becomes for this equation. The results from all InceptionSR variants outperforms that of PySR, with reusing the Top 5 Elbow equations performing the best. As expected, when the highest complexity equation was reused, the lowest average MSE was achieved. This is because the highest complexity equation would have the lowest loss out of the whole Pareto frontier, and whenever it was reused in Pareto frontier of InceptionSR, it would dramatically reduce loss. This improvement is evident in Table 4, where all iterations of InceptionSR show a significant reduction in loss after incorporating the complexity-30 equations as building blocks.

Table 3: MSE Equation Complexity 30 Comparison

Time (Hours)	PySR	InceptionSR (All)	InceptionSR (Max Complexity)	InceptionSR (Top 5 Elbow)
1	1.725±0.044	-	-	-
2	1.746±0.040	1.711±0.043	1.719±0.042	1.692±0.038

Table 4: InceptionSR Max Complexity

Complexity	Equation	Loss
27	$y_0 = ((4.57/(s14^{0.13})) + (0.22 * (((-1.22 * ((sin(m1) - s13) + s16)) - m1) - (sin(m19) + (m19 + s15))))) - s1$	1.824
32	$y_1 = prev - s5$	1.701
30	$prev = (((6.39 - ((1.66^{m1}) + s19)) - (0.02 * (-2.39/(s14 + (0.06 * s7))))) - sin((m19 - ((m13 + 0.36) * s3)) * 0.18)) + 0.42$	
31	$y_0 = ((((6.98 - sin(s14^{-0.31})) - ((s1 + (((m1 + m19) + 0.83) * 0.28)) + s1)) - (sin(s7) + ((s19 - 0.42)/0.52))) - 0.32) - s5$	1.825
32	$y_1 = prev - 1.00$	1.647
30	$prev = ((((s14 - (s7 * (-0.06/0.84)))^{-0.34}) + 4.35) * 1.07) - ((1.84^{m1})^{0.81})) - sin(0.14 * (((m19/0.61) - m13) - 1.29))$	

Table 4 shows two different seeds where y_0 represents the last pareto front equation without subexpression reuse, y_1 shows how the subexpression 'prev' is reused in the subsequent equation, and the 'prev' equation displays the reused subexpression itself. The complexity and loss for each equation are provided. Each y_1 equation that reuses the previous iteration's expression achieves a lower loss than its predecessor y_0, demonstrating the effectiveness of reusing discovered expressions. Higher complexity equations after y_1 all build upon their respective reused expression, further decreasing the loss. This suggests that recursively applying PySR first discovers a simpler relationship between features, and subsequently uses this discovery to find the rest of the equation.

Although one might expect that an expanded feature space would slow down evolutionary symbolic regression, we observe that reused subexpressions actually provide a more refined basis for equation construction. This enriched starting point leads to more efficient synthesis of high-quality equations whenever these subexpressions are incorporated. These results suggest that InceptionSR's recursive approach to equation discovery discovers better equations for predicting the instability of 3-body planetary systems, whenever previous subexpressions are reused, compared to vanilla PySR in the same amount of time.

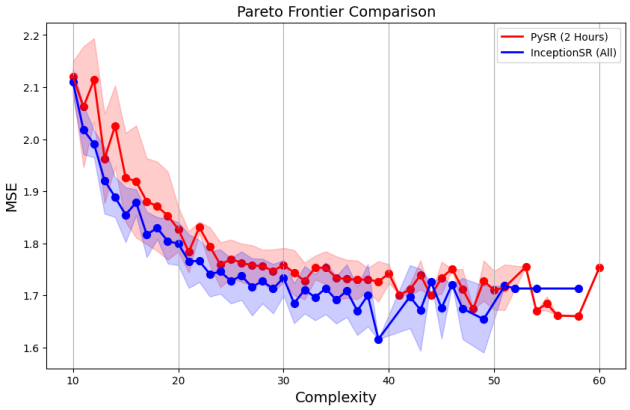

Figure 2: PySR vs InceptionSR All Complexities Reuse

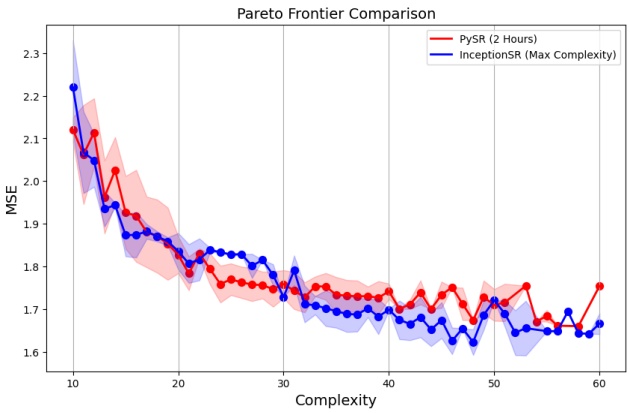

Figure 3: PySR vs InceptionSR Complexity 30 Reuse

Heron's Formula Rediscovery

We also explored a modified version of InceptionSR, where, instead of recursively reusing PySR, we include the library-learning method Stitch in between PySR runs, in order to aid in the discovery and reuse of key subexpressions (Bowers et al. 2023). We validate this method on Heron's formula, which expresses the area of a triangle in terms of the lengths of its sides (a, b, c).

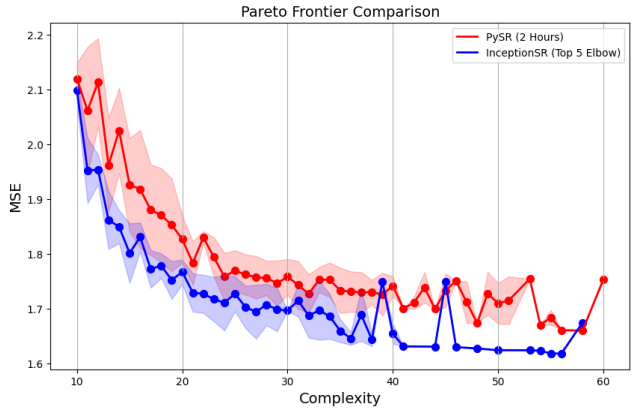

Figure 4: PySR vs InceptionSR Top 5 Elbow Complexities Reuse

$$A = \sqrt{s(s-a)(s-b)(s-c)} \tag{10}$$

$$s = \frac{a+b+c}{2} \tag{11}$$

We created a synthetic dataset of 10,000 data points from random values of input features a, b, and c. Each data point had a target label of area A, computed from the values of the input features. Running a symbolic regression algorithm may achieve lower losses, but the generated equations themselves may not appear reminiscent of the form of Heron's formula. The goal is to find a variant of the subexpression s, which represents the semiperimeter of the triangle, and to reuse it multiple times correctly in an equation.

Running PySR for multiple hours was unsuccessful in discovering a subexpression close to s, and therefore we never reused any variation of a summation between a, b, and c, in any equation in the Pareto frontier. Recursively running PySR, where all equations are reused in subsequent iterations, for multiple hours also failed to discover the functional form of Heron's formula. However, directly running PySR for 1 hour produces equations that contain $a + b + c$ as a component in the overall equation. By applying Stitch to the Pareto frontier generated by PySR, we successfully identified the function $f(x) = x-(a+b+c)$, which encapsulates the crucial subexpression underlying Heron's formula. Although the exact output of Stitch cannot be included as an operator in the next iteration of PySR, we use the output of Stitch as an oracle and extract the feature $-a-b-c$ to reuse as s. Subsequently, the second iteration of PySR (1 hour) recognized the importance of s and reused it three times in all equations on the Pareto frontier after complexity 20. Here is one such equation synthesized (coefficients removed):

$$A = \frac{s}{a} - a - \frac{abc}{s} + s + 1 \tag{12}$$

PySR demonstrates particular effectiveness in discovering summative relationships, suggesting that logarithmic transformation of Heron's formula could better facilitate its discovery. Furthermore, simplifying the search space by eliminating division operations and numerical coefficients could also be helpful. While our approach did not fully reconstruct Heron's formula, we demonstrated that integrating library learning methods enables the identification and strategic reuse of critical subexpressions from symbolically regressed equations. This integration enables the systematic reuse of important mathematical components, an ability that existing symbolic regression methods struggle with. We believe that longer runtimes and utilizing more powerful symbolic regression methods would lead to a more accurate Heron's formula rediscovery.

Conclusion

We have shown the efficacy of recursively running symbolic regression in order to discover equations that better model input data. Through the planets instability problem, we demonstrate that InceptionSR enables the use of a superior basis for synthesizing equations, through the reincorporation of subexpressions from the previous iteration of regression. Recursively incorporating the output equations of each iteration of symbolic regression into the feature space of the next iteration of symbolic regression, is reminiscent of gradient boosting and more powerful than continuously running symbolic regression for longer durations. As a library learning method, we have demonstrated that combining PySR with Stitch is potentially an effective way to discover key subexpressions that may be reused multiple times within a larger equation, such as the variable s in Heron's formula.

Future Work Due to the flexibility of InceptionSR, we may recursively use any other symbolic regression method (e.g. AI Feynman) in place of PySR, in order to recursively search through the space of possible equations in a more informed way. It is also interesting to see how hybrid symbolic regression and library learning methods may be used in tandem to iteratively refine a useful library of mathematical relations between features in a dataset. Elements in such a library can be used to compose higher-complexity equations to model more complicated phenomenon. We want to validate our approach on existing benchmarks such as SRBench, as well as predict equations in other scientific domains. Future work on the planets instability prediction problem would involve running more iterations of InceptionSR and comparing those results with vanilla PySR. Having a physicist interpret the physical meaning behind the synthesized equations at each iteration is crucial for advancing scientific theory, and leveraging LLMs for equation interpretation is a promising direction as well.

References

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

Chambers, J.; Wetherill, G.; and Boss, A. 1996. The stability of multi-planet systems. *Icarus*, 119(2): 261–268.

Cranmer, M. 2023. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. arXiv:2305.01582.

Cranmer, M.; Tamayo, D.; Rein, H.; Battaglia, P.; Hadden, S.; Armitage, P. J.; Ho, S.; and Spergel, D. N. 2021. A Bayesian neural network predicts the dissolution of compact planetary systems. *Proceedings of the National Academy of Sciences*, 118(40): e2026053118.

Cranmer, M. D.; Sanchez-Gonzalez, A.; Battaglia, P. W.; Xu, R.; Cranmer, K.; Spergel, D. N.; and Ho, S. 2020. Discovering Symbolic Models from Deep Learning with Inductive Biases. *CoRR*, abs/2006.11287.

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

Fong, K. S.; Wongso, S.; and Motani, M. 2023. Evolutionary Symbolic Regression: Mechanisms from the Perspectives of Morphology and Adaptability. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, 21–22. New York, NY, USA: Association for Computing Machinery. ISBN 9798400701207.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

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

Grayeli, A.; Sehgal, A.; Costilla-Reyes, O.; Cranmer, M.; and Chaudhuri, S. 2024. Symbolic Regression with a Learned Concept Library. arXiv:2409.09359.

Gulwani, S.; Polozov, A.; and Singh, R. 2017. *Program Synthesis*, volume 4. NOW.

Hadden, S.; and Lithwick, Y. 2018. A criterion for the onset of chaos in systems of two eccentric planets. *The Astronomical Journal*, 156(3): 95.

Holt, S.; Qian, Z.; and van der Schaar, M. 2023. Deep Generative Symbolic Regression. arXiv:2401.00282.

Langley, P. W. 1977. BACON: a production system that discovers empirical laws. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, 344. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Makke, N.; and Chawla, S. 2023. Interpretable Scientific Discovery with Symbolic Regression: A Review. arXiv:2211.10873.

Obertas, A.; Van Laerhoven, C.; and Tamayo, D. 2017. The stability of tightly-packed, evenly-spaced systems of Earth-mass planets orbiting a Sun-like star. *Icarus*, 293: 52–58.

Petit, A. C.; Laskar, J.; and Boué, G. 2018. Hill stability in the AMD framework. *Astronomy & Astrophysics*, 617: A93.

Petit, A. C.; Pichierri, G.; Davies, M. B.; and Johansen, A. 2020. The path to instability in compact multi-planetary systems. *Astronomy & Astrophysics*, 641: A176.

Rivero, D.; Fernandez-Blanco, E.; and Pazos, A. 2022. DoME: A deterministic technique for equation development and Symbolic Regression. *Expert Systems with Applications*, 198: 116712.

Schmidt, M.; and Lipson, H. 2009. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923): 81–85.

Sipper, M.; and Moore, J. H. 2021. Symbolic-regression boosting. *Genetic Programming and Evolvable Machines*, 22(3): 357–381.

Tamayo, D.; Cranmer, M.; Hadden, S.; Rein, H.; Battaglia, P.; Obertas, A.; Armitage, P. J.; Ho, S.; Spergel, D. N.; Gilbertson, C.; Hussain, N.; Silburt, A.; Jontof-Hutter, D.; and Menou, K. 2020. Predicting the long-term stability of compact multiplanet systems. *Proceedings of the National Academy of Sciences*, 117(31): 18194–18205.

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