An alternative and efficient method for Symbolic Regression

Massimo Bacchione

Department of Computer Science, Bioengineering, Robotics and System Engineering (DIBRIS)

University of Genova

Supervisor

Luca Oneto

Co-Supervisor

Damiano Verda

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Engineering

July 16, 2026

Abstract

Symbolic Regression (SR) is a machine learning technique that discovers mathematical expressions directly from data, without assuming any functional form. Unlike black-box models, SR produces interpretable equations that can be directly related to the underlying law being studied, making it particularly valuable in scientific domains such as physics. However, most state-of-the-art SR methods rely on evolutionary algorithms, which can be computationally expensive and non-deterministic.

This thesis proposes an alternative approach to SR, based on feature expansion and iterative linear regression. The method generates a large set of candidate mathematical features from the original input variables, and progressively builds a symbolic expression by selecting and combining the most relevant terms through a sequence of linear regression steps, either in additive or multiplicative way. The resulting expression is then simplified to produce a compact and interpretable formula.

The proposed method is compared against PySR running on datasets from the SRSD-Feynman benchmark, a collection of physical laws derived from the Feynman Lectures on Physics, using R^2, Mean Absolute Error (MAE), and execution time as evaluation metrics. Results show comparable accuracy between the two methods, with the proposed method achieving a lower average MAE and shorter execution

time. These results suggest that the proposed method is a controllable and lightweight alternative to evolutionary Symbolic Regression.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

In recent years, the exponential growth of available data across scientific and engineering domains has driven the development of increasingly sophisticated data-driven methods. Among these, Machine Learning (ML) has emerged as a powerful tool for extracting patterns and making predictions from complex datasets. However, a well-known limitation of most ML approaches is their lack of interpretability: models such as deep neural networks operate as black boxes, providing accurate predictions without offering any insight into the underlying relationships between variables [1].

This limitation becomes particularly critical in scientific contexts, where the goal is not only to predict outcomes but also to understand and validate the physical laws governing a system. In fields such as physics, chemistry, and materials science, having an interpretable and generalizable analytical expression is often as important as predictive accuracy itself [1].

Symbolic Regression (SR) addresses this gap by automatically discovering mathematical equations directly from data, without requiring any prior assump-

tion on the functional form of the model. Unlike conventional regression methods, which fix the structure of the equation and only optimize its coefficients, SR explores the full space of possible mathematical expressions simultaneously [1]. This makes it a particularly promising approach for scientific discovery, as it can recover human-readable equations that can be interpreted, validated, and generalized beyond the training data.

Despite its potential, SR has historically been limited by its high computational cost. Recent advances in hardware and algorithmic efficiency have made SR more accessible, leading to the development of tools such as PySR [2], a high-performance symbolic regression library available in Python and Julia. A state-of-the-art SR tool that leverages genetic programming to search for optimal symbolic expressions. The availability of standardized benchmarks, such as the Feynman Symbolic Regression Benchmark [3],has further enabled systematic evaluation and comparison of SR methods across a wide range of physical equations.

The potential of SR in scientific discovery is well illustrated by results already obtained in the literature. In particular, SR-based approaches have been successfully applied to rediscover Newton's law of gravitation and to recover conservation laws directly from trajectory data, without prior knowledge of the underlying physics [1]. These results demonstrate that SR is not merely a curve-fitting tool, but a method that can discover meaningful physical relationships from data.

It is within this context that the present thesis is motivated. Although SR methods have already been tested on benchmarks such as the SRSD-Feynman datasets [3], most of them are based on evolutionary algorithms, which tend to be computationally expensive and non-deterministic. This work proposes a simpler alternative, based on automated feature expansion and iterative linear regression, which does not rely on evolutionary search. The proposed method is

then compared with PySR [2] on the same benchmark datasets, to evaluate how it performs relative to a state-of-the-art SR tool.

1.2 Context of the Study

The present thesis is developed in collaboration with Rulex Innovation Labs [4; 5], within the framework of a research activity aimed at exploring the applicability of Symbolic Regression techniques to real-world data-driven problems.

The work is centered around the Feynman Symbolic Regression Benchmark, a well-established collection of datasets derived from the Feynman Lectures on Physics [3]. Each dataset corresponds to a physical law and consists of input variables and a target output computed by applying the corresponding formula. In this study, we consider a subset of 100 datasets, each containing 8000 samples, where each row is structured as a sequence of input variables $x_0, x_1, \ldots, x_{n-2}$ followed by the target value x_{n-1}, obtained by evaluating the underlying physical equation on the given inputs.

These datasets represent a benchmark for evaluating Symbolic Regression methods, as they cover a broad range of physical phenomena from simple multiplicative laws to more complex expressions involving trigonometric and exponential functions [3].

Among the most competitive SR methods evaluated on this benchmark, PySR [2] has demonstrated state-of-the-art performance, particularly in terms of solution rate and normalized edit distance (NED) [3]. For this reason, PySR is adopted in this work as a reference baseline, against which the results of the proposed method developed during this thesis are compared.

The method developed in this thesis follows a different strategy compared to PySR. Instead of using evolutionary algorithms to search for symbolic ex-

pressions, the proposed method automatically generates a large set of candidate mathematical features from the original input variables — applying transformations such as sin, cos, log, exp, and x^2, as well as combining pairs of variables through basic arithmetic operations (addition, subtraction, multiplication, and division). The resulting expression is then built step by step: at each iteration, the most promising candidate feature is selected and fitted using linear regression, either by adding or multiplying it to the current expression. The final result is simplified using SymPy to produce a compact and readable symbolic formula.

This approach is deterministic and computationally lightweight compared to genetic programming-based methods. Its performance on the SRSD-Feynman benchmark datasets is compared with PySR throughout this work.

1.3 Objectives and Contributions

The main objective of this thesis is to evaluate and compare the performance of two Symbolic Regression approaches on the SRSD-Feynman datasets: PySR, a state-of-the-art SR tool based on evolutionary algorithms, and and the proposed method developed during this work, based on automated feature expansion and iterative linear regression.

Both methods are tested on the same set of 100 datasets and the comparison is carried out using two evaluation metrics: the coefficient of determination (R^2) and the Mean Absolute Error (MAE), which together provide a measure of both the accuracy and the quality of the found expressions.

The main contributions of this thesis are:

- The implementation of an alternative Symbolic Regression method in Python, based on automated feature expansion and iterative linear regression, designed as an alternative to genetic programming-based approaches.

- A systematic benchmark comparison between the proposed method and PySR on 100 SRSD-Feynman datasets, showing that the two approaches achieve comparable or equivalent results on the majority of the datasets, particularly those involving simpler algebraic expressions.

1.4 Overview of the Thesis

To provide a roadmap of the research conducted, this thesis is structured into six chapters. Each chapter addresses specific aspects of the study, from the theoretical foundations to the final evaluation of the results.

Chapter 2: State of the Art A comprehensive review of the existing literature and current technologies related to the research topic. This chapter analyzes previous works, identifies current limitations in the field, and establishes the theoretical foundation upon which this thesis is built.

Chapter 3: Experimental Setup This chapter describes the datasets and tools used in this work. It introduces the datasets adopted for the experiments, covering both the synthetic datasets used in the preliminary phase and the SRSD-Feynman datasets It then describes the setup parameters of PySR, and presents the results of a preliminary experimental phase conducted to understand PySR's behavior under different conditions. .

Chapter 4: An Alternative Method This chapter describes the proposed Symbolic Regression method developed in this thesis. It explains all the steps involved in the proposed method used to progressively build symbolic expressions and the evaluation metrics adopted to assess its performance.

Chapter 5: Experimental Results This chapter presents the results of the comparison between PySR and the proposed method on the SRSD-Feynman

datasets. It includes a quantitative analysis of accuracy and execution time, as well as a set of illustrative examples that compare the symbolic expressions discovered by the two methods.

Chapter 6: Conclusions The final chapter summarizes the key findings and evaluates the extent to which the research objectives were met. It discusses the limitations of the current study and proposes potential directions for future work and further enhancements.

Chapter 2

State of the Art

Summary

This chapter provides an overview of the existing literature related to this thesis. It covers the foundations of Symbolic Regression, the role of Genetic Programming as the main algorithmic framework, and the most common SR methods proposed in the literature.

2.1 Symbolic Regression: Definition and Background

Symbolic regression (SR) is a regression method that uses machine learning (ML) and genetic programming principles to combine techniques and processes from different scientific fields and can generate analytical equations purely from data. Traditionally addressed through Genetic Programming, SR has recently attracted growing interest from the deep learning (DL) community as a tool for data-driven model discovery, achieving significant advances across a wide range of application domains, from fundamental to applied sciences [6].

Although such DL models are highly effective in making predictions, they are often difficult for humans to interpret since the mapping between inputs and outputs is not expressed in an explicit mathematical form[7]. These so-called black-box models [8] do not follow the principles of explainable and generalizable AI [1], making it difficult to understand why a model produces a given output.

In this sense, it is worth making a distinction between interpretability and explainability: while explainability refers to the use of additional methods to explain the predictions of an existing model, interpretability implies that the model itself is transparent and its internal logic can be directly understood because it is expressed in a clear and compact mathematical form [6; 9]. A linear regression model, for instance, is interpretable because the contribution of each input variable to the output can be directly read from its coefficients.

The absence of physical interpretation in black-box models has motivated the development of alternative approaches. SR addresses this limitation by generating symbolic expressions directly from data, without imposing prior constraints on the form of the equation, and provides a physics-inspired perspective on data-driven model discovery. [1].

This makes SR a rapidly growing subfield of machine learning aimed at inferring symbolic mathematical expressions directly from data [10; 11]. A key motivation behind SR is the growing demand for interpretable models [9].

This is particularly relevant in fields such as physics, where equations derived from data can be validated against known physical laws, or in critical domains such as healthcare, where non-interpretable models may not be suitable for deployment regardless of their predictive accuracy [6; 9].

Despite its advantages, SR presents some notable limitations. The main disadvantage is by no doubt the computational cost required to evaluate thousands of candidate expressions [12], which has historically been the main barrier to its

wide adoption. Recent advances in hardware, particularly the development of fast parallel architectures, have helped reduce this burden, opening the road to more efficient implementations [1]. For this reason, SR tends to perform better when the number of input variables is kept small [13], and a common strategy is to first apply feature selection methods to identify the most relevant variables before running SR [1].

Performance can also be affected by data quality. Issues may arise due to data scarcity or the presence of noise [14], although Bayesian approaches have been proposed to handle noisy datasets. On the other hand, SR has proven to be more effective than several ML models when working with small datasets [15], making it particularly suitable for scientific applications where data collection is expensive or limited.

Another known limitation is the so-called bloat effect, a common side effect of Genetic Programming in which the complexity of the generated expressions grows rapidly while the improvement in accuracy remains marginal [1; 16]. As a result, more complex expressions are not necessarily more accurate, and care must be taken when interpreting SR results, as oversimplified datasets and inadequate evaluation metrics may lead to misleading conclusions. Furthermore, the performance of SR methods tends to degrade significantly as the complexity of the target expression increases, with solution rates dropping close to zero on more challenging datasets[3].

2.2 Genetic Programming for Symbolic Regression

Genetic Programming (GP) is the algorithmic framework most commonly used to implement Symbolic Regression [1; 11]. It is an evolutionary method that searches

for mathematical expressions by iteratively combining and modifying candidate solutions, in a process inspired by natural selection. This section describes the foundations of GP, the procedure it follows to discover symbolic expressions, some used programming techniques and one of the most recent implementations, PySR.

2.2.1 Foundations of Genetic Programming

Genetic programming (GP) is a type of evolutionary algorithms (EA)[17], which at the same time is part of the wider field of Evolutionary Computing (EC) [18].

GP explores the space of mathematical expressions through an evolutionary process, and can be applied to most regression-based problems in science and engineering [1].

Symbolic expressions in GP are represented as tree structures, where each node contains either a primitive function or a terminal. Primitive functions can take any mathematical form, such as $+$, $-$, $\times$, $\div$, log, while terminal nodes correspond to input variables (e.g., x, y) and numeric constants [1]. This tree-based representation allows GP to construct arbitrarily complex mathematical expressions by combining these elements in different ways.

During the search process, GP constructs a large number of candidate symbolic expressions and evaluates each of them against accuracy and complexity criteria. Expressions that do not meet the required standards are discarded, while those that show potential are combined to form new and improved expressions. The most common way to visualize this process is through a tree structure, where nodes represent mathematical operations and branches connect them to their operands [1].

A key mechanism of GP is the crossover operation, through which sub-expressions are exchanged between two parent expressions to generate new ones [1]. Given two candidate expressions S_1 and S_2, both selected based on their fitness, a ran-

dom sub-configuration is chosen from each of them. These sub-configurations are then swapped, producing two new expressions S_3 and S_4, while the original parent expressions are removed from the process. The resulting expressions inherit parts from both parents, and therefore differ from them in both shape and depth. This exchange happens in parallel across thousands of expressions, iteratively combining the most promising candidates until a robust and accurate equation is obtained [1].

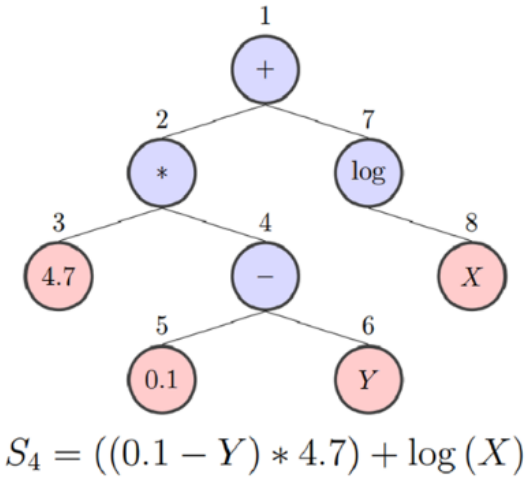

Figure 2.1: Example of symbolic expression tree structure in Genetic Programming [1].

The goal of GP is to find the combination of nodes and terminals that best fits the given dataset. To do so, GP explores an effectively infinite search space that includes all possible mathematical operators and numeric constants [1]. Each candidate expression is evaluated based on the error it produces on the training data, and the best are selected for further evolution.

There exist metrics used in GP-based SR such as the Sum of Squared Errors (SSE), defined as:

$$SSE = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2.1}$$

where n is the number of samples, Y_i is the true value and $\hat{Y}_i$ is the predicted value. Alternatively, the Mean Squared Error (MSE) is used to average the error over all instances:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2.2}$$

Throughout this iterative process, random mutations are also introduced to avoid the algorithm from becoming trapped in local minima. The process continues until a stopping criterion is met, such as reaching a maximum number of iterations or finding an expression whose error is less than a given threshold, at which point the final equation is exported [1].

2.2.2 SR Implementation Approaches

The evolution of SR implementations has been correlated with the availability of computational resources. For a long time, the cost of exploring large spaces of mathematical expressions limited SR to small-scale problems. The rapid development of modern parallel hardware has significantly changed this scenario, enabling the design of new SR algorithms that can handle larger datasets and more complex expressions in reasonable time frames.

Several different approaches have been proposed in the literature to implement SR. Some methods employ a Monte Carlo tree search (MCTS) algorithm [1], a matrix-based encoding process [19], or advanced pre-processing schemes applied before training [20] or before regression begins [21].

Others use nearest neighbor indexing [22] or non-evolutionary techniques such as the FFX algorithm [23], which, although extremely fast, tends to produce

non-interpretable equations. Additional formulations include Mixed-Integer Non-Linear Programming (MINLP) [24; 25], linguistic models [26], probabilistic frameworks [27], probabilistic grammars [28], and Bayesian approaches [29].

A common direction in GP-based SR research is improving the accuracy of the generated expressions [30], which is considered a key property for the practical applicability of ML models.

Several efforts have been made to modify and extend the basic GP-SR procedure in this direction [31; 32; 33].

One particularly interesting approach is the restriction of the search space by incorporating prior knowledge about the system, such as enforcing constraints like monotonicity or symmetry in the generated expressions [19; 28; 34; 35].

This strategy can increase accuracy and reduce computational cost while keeping the results aligned with known physical laws and empirical relations [36].

However, some researchers argue that GP-based procedures can sometimes lead to abstract mathematical formulas that lack physical meaning [22], highlighting the importance of carefully designing the search space and the constraints applied to it; constrained approaches also have limitations. Methods that restrict the formula to a fixed set of symbols can produce accurate results at low computational cost, but may fail on more complex cases involving periodicity or exponential terms [37].

To overcome these limitations, hybrid methods have been proposed, combining deterministic and GP-based approaches [38], or integrating neural networks to guide the search [21; 39].

Some studies have integrated SR within deep learning frameworks, where neural-network architectures are trained end-to-end through backpropagation to discover symbolic expressions. Others have adopted a reinforcement learning approach, using recurrent neural networks to search the space of mathematical

expressions through policy gradients, outperforming traditional GP-based methods on several benchmark problems [40; 41].

Promising results have also been obtained using Bayesian Neural Networks (BNN) [42]. Unlike conventional neural networks, BNNs treat model parameters as probability distributions rather than fixed values, and train through Bayesian inference [43]. This allows the model to quantify uncertainty and incorporate confidence intervals into its predictions, which makes it particularly suitable for noisy or incomplete datasets [43].

Nowadays, SR methods are available either as standalone tools or integrated into user-friendly platforms such as Eureqa and HeuristicLab [44; 45].

A significant step towards a systematic evaluation of SR methods was made by La Cava et al. research, who introduced SRBench, an open-source and reproducible benchmarking platform for symbolic regression. SRBench assesses 14 symbolic regression methods and 7 machine learning methods on a set of 252 regression problems, including both real-world datasets and ground-truth benchmark problems such as physics equations and systems of ordinary differential equations.

The study concludes that the best performing methods for real-world regression combine genetic algorithms with parameter estimation and semantic search, while deep learning and genetic algorithm-based approaches perform similarly in retrieving exact equations in the presence of noise [46].

2.2.3 PySR: A State of the Art SR Tool

Among the most popular modern implementations of GP-based Symbolic Regression, PySR [2] stands out as one of the most widely adopted tools in the scientific community. Built on the foundations of Genetic Programming described in the previous section, PySR extends the classic evolutionary approach with

some additions that improve results making it more suitable for real-world scientific applications. It is an open-source Python and Julia library developed with the explicit goal of democratizing and popularizing SR in the sciences, a type of machine learning whose objective is discovering human-interpretable symbolic models[2].

2.2.3.1 Base Algorithm

PySR's search algorithm is a multi-population evolutionary algorithm based on tournament selection. Starting from a population of randomly generated symbolic expressions, the algorithm iteratively selects individuals based on their fitness, applies mutations and crossovers to generate new expressions, and replaces members of the population with the new ones. As a first modification to this classic approach, PySR replaces the oldest individual in the population at each step rather than the worst-performing one, a strategy known as age-regularized evolution, which prevents the population from converging too quickly and getting stuck in local minima.

2.2.3.2 Algorithmic Improvements

In addition, developers have introduced three key modifications to this base GP-based algorithm. PySR applies simulated annealing [47], a probabilistic optimization algorithm, to the selection process, in which mutations that worsen the current solution are also accepted, encouraging a broad exploration of the expression space. As the search progresses, it becomes increasingly selective, accepting only mutations that improve the current solution. This strategy helps avoid getting stuck in suboptimal solutions and has been shown to significantly speed up the search process.

Second, the genetic algorithm is embedded inside a unique evolve-simplify-

optimize cycle: the "evolve" step applies tournament selection and genetic operations such as mutation and crossover; the "simplify" step reduces expressions to a canonical form using algebraic equivalencies; and the "optimize" step explicitly optimizes any numerical constants in the expressions using a classical optimization algorithm; this last step is particularly important for scientific applications, where equations often contain unknown real constants that must be discovered alongside the symbolic structure.

Third, an adaptive parsimony metric dynamically penalizes overly complex expressions, encouraging the search to explore both simple and complex functional forms. The concept of complexity here is defined as the number of nodes in an expression tree, regardless of the type of each node. However, this definition is fully configurable by the user, since the notion of "simplicity" is domain-dependent.

PySR also includes a set of additional features designed for scientific use, such as built-in support for noisy data through Gaussian process denoising, custom loss functions, user-defined mathematical operators specific to different scientific domains, automatic feature selection, and configurable constraints on the structure of discovered expressions.

2.2.3.3 Software Implementation

PySR's search algorithm is implemented in Julia under the library *SymbolicRegression.jl*. Julia combines a high-level interface with performance comparable to lower-level languages such as C++ and Rust, and its key advantage for PySR is its just-in-time compilation. This allows user-defined operators to be compiled directly into the core search functions and fused together into optimized SIMD (Single Input Multiple Data) kernels at runtime, resulting in significant speedups in expression evaluation. A simple Python frontend API, following the popular scikit-learn style [48], makes this high-performance Julia backend accessible to a

broader audience, making PySR usable without requiring knowledge of Julia.

As regards parallelization, PySR distributes independent populations of expressions to separate workers, which evolve asynchronously without slowing each other down. This approach is based on the large-scale parallelization strategy described in [49]. After each batch of iterations, the best expressions at each complexity level are recorded in a global "hall of fame", and expressions are randomly migrated between populations and from the hall of fame back into the populations; reintroducing hall of fame members can significantly speed up the search.

Additionally, while most SR benchmarks use Mean Squared Error (MSE) as the default learning objective, scientific applications often require more specific loss functions, as the underlying data distributions are frequently non-Gaussian. For this reason, PySR supports fully user-defined loss functions, passed directly as strings to the Julia backend, allowing researchers to define arbitrary regression objectives and custom likelihoods suited, for example, to a Bayesian context.

2.2.3.4 PySR Performances

To evaluate PySR's performances, the authors have introduced EmpiricalBench, a new benchmark consisting of 9 real empirical equations historically discovered by scientists from experimental data, such as Hubble's law, Kepler's Third Law and Newton's law of universal gravitation and others. Unlike existing benchmarks such as the Feynman datasets [21] or SRBench [46], which use synthetic data with simplified noise models, EmpiricalBench uses original experimental data, if available, with realistic noise and without physical constants, in such a way that the algorithm must discover them automatically, as the original scientists did in the past.

PySR was tested against five other methods — Operon [50], DSR [41], EQL

[51], QLattice [52], and SR-Transformer [53] — on EmpiricalBench, each method allowed up to one hour of computation on 8 CPU cores.

PySR successfully retrieved the correct equations in the majority of cases, outperforming all other tested methods. In particular, the two pure deep learning approaches, EQL and SR-Transformer, recovered the fewest expressions, suggesting that classic evolutionary methods still outperform deep learning-based approaches on real-world noisy data [2].

On the SRSD-Feynman benchmark [3], PySR also demonstrated state-of-the-art results in terms of solution rate and normalized edit distance, particularly on low and medium complexity datasets, while performance degraded on more algebraically complex expressions in the Hard set. For these reasons, PySR was adopted in this work as a reference baseline for comparison with the proposed alternative method.

2.2.4 SR Applications

Due to its ability to generate interpretable analytical expressions directly from data, SR has found application in a wide range of scientific and engineering domains. In materials science, SR has been applied to predict material properties, discover interatomic potentials, and generate constitutive models for mechanical behavior. Other applications include the generation of equations from scratch towards the prediction of Lennard–Jones fluid diffusion or the electrical conductivity of ionic liquids, others have exploited SR for the prediction of lattice thermal conductivity [54], the critical temperature in superconductors [55] and the yield strength of polycrystalline metals [56].

In engineering, SR has been used across multiple subfields. In civil and construction engineering, SR-derived models have been shown to outperform conventional formulas in estimating seismic response and concrete structural properties.

In chemical engineering, SR has been applied to model drag coefficients in gas-solid flows and to identify kinetic laws of chemical reactions.

In mechanical engineering, SR has been used to derive constitutive models for material behavior in aluminum alloys and to estimate calibration parameters in physics-based models.

In computer engineering, SR has been applied to construct fault detection mechanisms that outperform traditional methods such as support vector machines and pattern recognition neural networks [57]. SR has also been combined with reinforcement learning to handle dynamic tasks and techniques such as search space narrowing through a semantic cluster library have shown promising results[58]. Moreover, SR has been used to improve learning behavior models and to decompose complex problems into more manageable subproblems[59].

More recently, PySR has been successfully applied in astrophysics, where it has been used to rediscover Newton's law of gravitation from trajectory data[60], to discover new astrophysical scaling relations [61], and to identify interpretable population models of gravitational wave sources [62]. In climatology, PySR has been used to discover new analytical equations for cloud cover parameterization, achieving performance comparable to neural networks while remaining interpretable [63]. In economics, it has been applied to discover symbolic models of international trade [64]. In high-energy physics, SR has been used to derive compact analytical expressions for particle collision observables at the Large Hadron Collider (LHC) [65; 66]. Finally, in bioimaging, SR has been combined with deep learning to produce interpretable models of cell state classification [67]

These results confirm that SR, and of course PySR, represents a powerful and flexible tool for empirical equation discovery across diverse scientific fields.

Chapter 3

Experimental Setup

3.1 Datasets

3.1.1 Preliminary Datasets

A preliminary phase was conducted to gain familiarity with PySR and to understand its functionalities and behavior under different conditions. For this purpose, synthetic (toy) datasets were generated using Rulex Software [5] - a proprietary software platform - with a controlled structure: each dataset consisted of 1000 rows, 10 input variables (`Var_0` to `Var_9`) and one target whose value was computed from a predefined equation involving a subset of the input variables. Both linear and non-linear target equations were considered, as summarized in Tables 3.1 and 3.2.

3.1.2 SRSD-Feynman Datasets

The main experiments were conducted on a subset of the SRSD (Symbolic Regression Scientific discovery)-Feynman datasets [3], a collection of datasets derived from physical laws contained in the Feynman Lectures on Physics. Each dataset

corresponds to a specific physical equation and consists of 8000 samples, where each row contains the values of the input variables and the corresponding target value computed by applying the equation to those inputs. The datasets are organized into three difficulty categories based on the complexity of the underlying equation and the sampling ranges of the variables: Easy (30 datasets), Medium (40 datasets) and Hard (50 datasets); in this work, a subset of 100 datasets was considered, covering all categories. Each dataset is provided as a `.csv` file, where the columns represent the input variables $x_0, x_1, \ldots, x_{n-2}$ and the last column x_{n-1} contains the target value. The equations cover a wide range of physical domains, including classical mechanics, electromagnetism and thermodynamics, with a number of input variables ranging from 1 to 9, depending on the complexity of the underlying physical law. For example, the dataset includes the law of friction $F = \mu N_n$, the gravitational potential energy $U = mgz$, and the relativistic energy equation $E = mc^2/\sqrt{1 - v^2/c^2}$ [3].

A key feature of the SRSD-Feynman datasets, which distinguishes them from the original Feynman Symbolic Regression Database (FSRD) [68], is that the sampling ranges of the variables are designed to reflect physically realistic values in SI units [3].

In the original FSRD, variables were sampled from simplified ranges such as $[1, 5]$ regardless of their physical meaning. Furthermore, physical constants such as the speed of light c and the gravitational constant G, which were treated as variables in the original FSRD, are instead fixed to their known physical values in the SRSD datasets [3].

Both methods considered in this work were run on the entire set of 8000 samples available for each dataset. This is consistent with the nature of Symbolic Regression, where the goal is to retrieve the underlying symbolic expression, rather than to generalize to unseen data points, since when an expression is

found, it is compared to the true equation [2].

3.2 PySR Setup

3.2.1 Key Parameters

`binary_operators` **and** `unary_operators`

These parameters define the set of mathematical operators that PySR is allowed to use when constructing symbolic expressions. Binary operators take two arguments (e.g., $+$, $-$, $\times$, $\div$), while unary operators take one argument (e.g., sin, cos, log, exp). PySR offers a rich set of configurable parameters that allow the user to control both the search process and the complexity of the discovered expressions. The choice of operators directly determines the search space because a larger set increases the expressivity of the model but also the computational cost. Custom operators can also be defined by the user as Julia functions [2].

`niterations`

This parameter controls the total number of iterations that PySR will do: Each iteration consists of a full evolve-simplify-optimize cycle applied to all populations. A larger number of iterations generally leads to better results, at the cost of longer computation time. PySR can also be stopped manually when the user is satisfied with the discovered expressions.

`populations` **and** `population_size`

The `populations` parameter defines the number of independent islands that evolve in parallel, while `population_size` controls the number of symbolic expressions in each island. A larger number of populations increases the

diversity of the search, while a larger population size allows each island to explore a wider range of expressions. It is generally recommended to have more populations than available CPU cores [2].

`maxsize` **and** `maxdepth`

These two parameters control the complexity of the discovered expressions by limiting the size of the expression tree. `maxsize` defines the maximum total number of nodes in the tree, including operators, variables, and constants, regardless of their position. `maxdepth` instead limits the number of levels of the tree, controlling how deeply nested the expression can be. For example, the expression $\sin(x_0 + x_1 \times x_2)$ has a size of 6 nodes and a depth of 4 levels. Setting both parameters helps reduce the search space and encourages PySR to find simpler and more interpretable expressions [2].

`model_selection`

This parameter controls how PySR selects the final expression from the Pareto front of accuracy and complexity. The default value `'best'` selects an expression that balances accuracy and simplicity, while `'accuracy'` selects the most accurate expression regardless of its complexity [2].

`elementwise_loss`

By default, PySR uses Mean Squared Error (MSE) as the loss function. This parameter allows the user to define a custom loss function as a Julia string, making it possible to optimize for arbitrary regression objectives beyond MSE. For instance, an absolute error loss or a weighted loss [2]. This is particularly useful in scientific applications where the data may follow non-Gaussian distributions.

`batching` **and** `batch_size`

When enabled, PySR evaluates expressions on a random subset of the data during mutations and annealing, rather than on the entire dataset. The size of this subset is controlled by the `batch_size` parameter. This makes the evolutionary process faster for large datasets, while equations are still evaluated on the entire dataset at the end of each iteration to compare to the hall of fame.

`warm_start`

When set to `True`, PySR resumes the search from the previous run instead of starting from scratch. This is useful when the user wants to continue a search that was interrupted, or to refine the results of a previous run by increasing the number of iterations.

`turbo`

When enabled, PySR uses a more aggressive optimization strategy that can significantly speed up the search, at the cost of slightly less diversity in the explored expressions. It is recommended for large datasets or when computational resources are limited.

`deterministic` **and** `random_state`

By default, PySR uses a random seed, meaning that different runs may produce different results. Setting `deterministic=True` and specifying a fixed`random_state` ensures that the search is fully reproducible across runs.

`parallelism`

This parameter controls how PySR distributes the computation across multiple cores. The available options are `'multithreading'` for single-node execution using threads, `'multiprocessing'` for multi-node execution using separate processes, and `'serial'` for single-core execution without parallelization, which can be useful for debugging or when running on limited

hardware.

`verbosity`

This parameter controls the amount of information printed to the console during the search. Setting `verbosity=0` suppresses all output, while higher values print progressively more details about the search progress, including the best equations found at each iteration.

3.2.2 Console Output and Search Progress

During the search, PySR prints a real-time summary of the best expressions found so far, updated at the end of each iteration. Figure 3.1 shows an example of the console output produced by PySR at the end of a given run.

The output is organized in a table with three columns:

Complexity

The number of nodes in the expression tree, which represent the structural complexity of the equation.

Loss

The value of the loss function (by default MSE) achieved by the expression on the training data. Lower values indicate a better fit.

Score

A normalized measure that combines accuracy and complexity, used internally by PySR to rank the expressions on the Pareto front. Higher values indicate a better balance between fit quality and simplicity.

Equation

The symbolic expression discovered by PySR, written in human-readable form.

Each row of the table represents a point on the Pareto front between accuracy and complexity: for each complexity level, the expression with the lowest loss is shown, reflecting the trade-off between model simplicity and accuracy.

At the end of the run, PySR produces two additional outputs. First, it saves the full hall of fame to a `.csv` file for later inspection. Second, it prints the complete list of discovered equations as a pandas DataFrame (`PySRRegressor.equations_`), which includes additional columns with respect to the console table: `pick` (the selection score used by `model_selection`), `score` which measures how much the loss improves relative to the expression of lower complexity (a high score indicates a significant gain in accuracy for a small increase in complexity), `loss`, and `complexity`. The equation marked with `>>>>` in the `pick` column is the one selected as the best by the `model_selection` criterion.

```
Compiling Julia backend...
[ Info: Started!
Evolving for 100 iterations... 100%
[ Info: Final population:

Complexity  Loss       Score      Equation
1           3.623e+01  0.000e+00  y = 4.7267
2           3.264e+01  1.043e-01  y = exp(x₁)
3           3.053e+01  6.683e-02  y = x₁ + 4.6967
4           2.014e+01  4.162e-01  y = exp(x₁) * 2.3275
6           1.011e+01  3.443e-01  y = (x₄ - exp(x₁)) * -2.4585
8           9.491e+00  3.182e-02  y = (exp(x₁) + (0.43234 - x₄)) * 2.2559
9           9.340e+00  1.607e-02  y = ((exp(x₁) - x₄) * 2.3879) + cos(x₁)
10          6.106e+00  4.251e-01  y = (x₃ * x₃) + ((exp(x₁) - x₄) * 2.2696)
12          5.266e+00  7.393e-02  y = ((x₃ * x₃) + exp(x₁) - (x₄ * 1.5995))) * 1.8736
14          2.673e+00  3.390e-01  y = ((exp(x₁) - x₄) + ((x₂ * x₀) + (x₃ * x₃))) * 1.934
16          2.522e+00  2.922e-02  y = ((exp(x₁) - x₄) * 2.1181) + ((x₃ * x₃) - (x₄ - (x₀ * x₂)))
18          2.018e+00  1.114e-01  y = (((x₃ * 1.4789) * x₃) - (x₄ - (x₀ * x₂))) + ((exp(x₁) - x₄) * 2.0277)
20          1.599e+00  1.164e-01  y = ((exp(x₁) - x₄) * 2.0913) + (((x₃ * x₃) + (x₀ * x₂)) - (x₄ - (x₂ * x₀)))

[ Info: Results saved to:
  - outputs\20260113_111603_czEOnD\hall_of_fame.csv

Learned equations
PySRRegressor.equations_ = [
           pick     score                                           equation       loss  complexity
       0         0.000000                                          4.7266674  36.232616           1
       1         0.104318                                            exp(x1)  32.643360           2
       2         0.066837                                    x1 + 4.696743  30.532888           3
       3         0.416242                                  exp(x1) * 2.327542  20.137070           4
       4         0.344276                       (x4 - exp(x1)) * -2.4584894  10.114907           6
       5         0.031824             (exp(x1) + (0.4323432 - x4)) * 2.2559478   9.491170           8
       6         0.016068               ((exp(x1) - x4) * 2.3879075) + cos(x1)   9.339884           9
       7         0.425079              (x3 * x3) + ((exp(x1) - x4) * 2.2696257)   6.105653          10
       8         0.073935  ((x3 * x3) + (exp(x1) - (x4 * 1.5995054))) * 1...   5.266388          12
       9         0.339004  ((exp(x1) - x4) + ((x2 * x0) + (x3 * x3))) * 1...   2.673361          14
       10        0.029217  ((exp(x1) - x4) * 2.118093) + ((x3 * x3) - (x4...   2.521621          16
       11        0.111359  (((x3 * 1.4789315) * x3) - (x4 - (x0 * x2))) +...   2.018156          18
       12  >>>>  0.116424  ((exp(x1) - x4) * 2.091345) + (((x3 * x3) + (x...   1.598931          20
]

Equazione migliore (raw)
x0*x2 + x3*x3 + (-x4 + exp(x1))*2.091345 - (-x0*x2 + x4)

 Equazione semplificata
2*Var_0*Var_2 + Var_3**2 - 3.091345*Var_4 + 2.091345*exp(Var_1)
```

Figure 3.1: Example of PySR console output

The last two lines shown in Figure 3.1, are not part of PySR's output, but represent a post-processing step performed separately, in which the selected expression is simplified using SymPy to produce a more readable form.

3.3 Preliminary Experiments

The goal of this phase was exploring how PySR behaves when varying key parameters such as the number of iterations, the number of populations, the population size, and the noise level applied to the target variable.

Two main experimental conditions were explored: one in which only the variables actually involved in the target equation were provided as input to PySR, and one in which all available variables were included. The results showed that PySR was generally able to recover the correct symbolic structure of the target equation in both cases, particularly when noise was absent.

As shown in Table 3.1, several configurations were tested before reaching the correct expression. After some trials with various settings, PySR was able to recover the exact target equation, as shown in the last row of the table. Previous configurations produced structurally similar but incomplete expressions, often missing one or more variables.

As the noise factor increased from 0 to 1, the recovered coefficients deviated progressively from the true values, while the overall symbolic structure of the equation was found. This means that PySR is robust to moderate levels of noise in terms of structural recovery, but less in terms of numerical precision of the coefficients.

In the case of non-linear equations, PySR required a slightly higher number of iterations and larger population sizes to recover the correct structure, as shown in Table 3.2. In this case, the correct expression was found only after a little testing with several configurations.

Table 3.1: PySR results on linear equation $y = 2x_0 + 2x_1 - 3x_3 - 2x_4 + 7x_5 + 4x_8$

Configuration	Input Features	Noise	maxSize	maxDepth	Result
300 it, 10 pop, 800 size	all	0	20	5	$x_0 - 3.36x_3 - x_4 + 7.14x_5 + 3.36x_8$
100 it, 12 pop, 600 size	all	0	20	5	$x_0 - x_4 - 3.36x_3 + 7.14x_5 + 3.36x_8$
100 it, 15 pop, 500 size	all	0	20	5	$x_0 - x_4 + 7.14x_5 - 3.36x_3 + 3.36x_8$
300 it, 15 pop, 700 size	selected	0	20	5	$x_0 - 3.36x_3 + 7.14x_5 - x_4 + 3.36x_8$
200 it, 15 pop, 800 size	selected	0	20	5	$x_0 - 3.36x_3 - x_4 + 7.14x_5 + 3.36x_8$
200 it, 15 pop, 800 size	selected	0.5	20	5	$x_1 - 3.42x_3 - x_4 + 7.14x_5 + 3.42x_8$
100 it, 15 pop, 500 size	selected	0.5	20	5	$x_1 - 3.42x_3 - x_4 + 7.14x_5 + 3.42x_8$
100 it, 15 pop, 700 size	selected	0.5	20	5	$-2.46x_3 - 2.46x_4 + 7.11x_5 + 3.95x_8$
100 it, 13 pop, 500 size	selected	0.5	20	5	$x_1 - 3.42x_3 - x_4 + 7.14x_5 + 3.42x_8$
100 it, 13 pop, 500 size	selected	1	20	5	$-3.42x_3 + 7.13x_5 + 3.42x_8$
100 it, 15 pop, 500 size	selected	1	20	5	$x_1 - 3.44x_3 - x_4 + 7.14x_5 + 3.44x_8$
200 it, 15 pop, 600 size	selected	1	20	5	$x_1 - 3.44x_3 - x_4 + 7.14x_5 + 3.44x_8$
100 it, 10 pop, 500 size	all	0	20	5	$2x_0+2x_1-3x_3-2x_4+7x_5+4x_8$

Table 3.2: PySR results on non-linear equations.
Equation 1: $y = 2(x_0x_1) + 3x_2^2 - 6(x_4x_5)$
Equation 2: $y = 2(x_0x_2) + 1.5x_3^2 - 3x_4 + 2e^{x_1} + 0.8(x_5x_6)$
Equation 3: $y = 2(x_0x_2) + 1.5x_3^2 - 3x_4 + 2e^{x_1}$

Configuration	Noise	maxSize	maxDepth	Result
Target: $y = 2(x_0x_1) + 3x_2^2 - 6(x_4x_5)$				
100 it, 10 pop, 500 size	0	20	5	$2x_0x_1+3.05x_2^2-6.33x_4x_5+0.06x_0-0.07x_5$
100 it, 10 pop, 500 size	0	20	5	$2x_0x_1 + 3x_2^2 - 6x_4x_5$
Target: $y = 2(x_0x_2) + 1.5x_3^2 - 3x_4 + 2e^{x_1} + 0.8(x_5x_6)$				
300 it, 10 pop, 500 size	0	20	5	$2.02x_0(x_2-0.01)+1.47x_1+1.47x_3^2-2.98x_4 + 2.83$
200 it, 10 pop, 500 size	0	20	5	$2.09x_0x_2 + x_3^2 - 3.03x_4 + x_5x_6 + 2.09e^{x_1}$
100 it, 10 pop, 700 size	0	20	5	$2.00x_0x_2+1.49x_3^2-3.00x_4+2.00e^{x_1}-0.007$
Target: $y = 2(x_0x_2) + 1.5x_3^2 - 3x_4 + 2e^{x_1}$				
100 it, 10 pop, 600 size	0	20	5	$2x_0x_2 + 1.5x_3^2 - 3x_4 + 2e^{x_1}$

Chapter 4

An Alternative Method

Summary

This chapter presents a method for Symbolic Regression developed in this work. The proposed method builds a symbolic expression from data through automated feature expansion followed by an iterative linear regression procedure.

The chapter concludes with a description of the evaluation metrics used to assess and compare the performance of the proposed method against PySR on the SRSD-Feynman benchmark datasets: the coefficient of determination R^2 and the Mean Absolute Error (MAE).

4.1 The Proposed Method

4.1.1 Feature Expansion

Before the iterative search begins, the original input variables are expanded into a larger set of candidate features. The goal of this step is to provide the algorithm with a rich set of mathematical terms from which the symbolic expression can be constructed through linear combinations.

First, a constant column `const_1` is added to the dataset, filled with ones. This allows the algorithm to include constant terms in the expression without requiring a separate intercept parameter in the linear regression.

Starting from the original input variables $x_0, x_1, \ldots, x_{n-1}$, the following unary transformations are applied to each variable independently:

`sin`	$\sin(x)$
`cos`	$\cos(x)$
`square`	x^2
`sqrt`	$\sqrt{x}$
`log`	$\log(x)$
`exp`	e^x (clipped to avoid overflow)

Binary transformations are then computed with all pairs of variables. Commutative operations, such as addition and multiplication, are applied to all unique combinations of pairs, while non-commutative operations, such as subtraction and division, are applied to all ordered permutations:

$\texttt{add}(x_i, x_j)$	$x_i + x_j$
$\texttt{mul}(x_i, x_j)$	$x_i \cdot x_j$
$\texttt{sub}(x_i, x_j)$	$x_i - x_j$
$\texttt{div}(x_i, x_j)$	x_i / x_j

Note: the set of transformations used in this work reflects the operators configured for PySR in the experiments described in Chapter 5, to ensure the two methods have the same functional building blocks. However, the proposed approach is not limited to this set and like PySR, it can support a wider range of operators, including hyperbolic functions, special functions, and conditional operators [2]. Extending the feature expansion step with other operators represents a natural direction for future work, as discussed in Chapter 6.

4.1.2 Step 0: Initialization

The proposed algorithm begins with an initialization step, whose goal is to choose the best single term to start building the symbolic expression.

First of all, the Pearson correlation coefficient is computed between each candidate feature in the expanded dataset and the target variable y; then the feature with the highest absolute correlation is selected as the initial term.

If the selected term is not a simple input variable, i.e. if it is a transformed or combined feature, a secondary expansion is applied to it.

This consists of applying the same set of unary transformations described in Section 4.1.1 to the selected term, generating a set of further derived features. The correlation with y is then recomputed on this secondary set, and the best term among the original and the secondary ones is selected.

Once the best initial term t_0 is identified, a linear regression is fitted on it:

$$\hat{y} = a + c \cdot t_0 \tag{4.1}$$

where t_0 is the selected term, a is the intercept and c is the fitted coefficient. The proposed method allow to include or exclude the intercept during the fit. By default the intercept is disabled to limit the final expression's complexity. In any case, a fallback fit with intercept is performed when the resulting R^2 score is negative, which should not occur in practice thanks to the expanded feature space. The predicted target function $\hat{y}$ obtained from this first fit constitutes the starting point for the subsequent steps.

4.1.3 Step 1 to N: Iterative Search

Starting from the initial prediction $\hat{y}$ obtained in Step 0, the algorithm iteratively builds the symbolic expression by adding a new term at each step. The number

of steps is controlled by the configurable parameter `n_steps`.

At each step, two candidate branches are evaluated in parallel:

Additive branch

The difference between the current prediction and the target is computed:

$$y_{add} = y - \hat{y} \tag{4.2}$$

The Pearson correlation between each candidate feature and y_{add} is then computed, and the most correlated term is selected. If the selected term can be expanded further, a secondary expansion is applied and the best term among the original and secondary ones is chosen. A linear regression is performed on the selected term to obtain a new prediction $\hat{y}_{add}$, and the updated expression becomes:

$$\hat{y} \leftarrow \hat{y} + \hat{y}_{add} \tag{4.3}$$

Ratio branch

The ratio between the target and the current prediction is computed:

$$y_{ratio} = \frac{y}{\hat{y}} \tag{4.4}$$

Likewise, the most correlated term is selected, a secondary expansion is applied if possible, and a linear regression is performed to obtain the $\hat{y}_{ratio}$ term. In this case the updated expression becomes:

$$\hat{y} \leftarrow \hat{y} \cdot \hat{y}_{ratio} \tag{4.5}$$

4.1.4 Branch Selection

At the end of each step, the two candidate branches are compared and the one that produces the best result on the target expression is selected. The primary criterion for selection is the coefficient of determination R^2, computed on the full dataset:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.6}$$

where $\bar{y}$ is the mean of the target values. The branch with the higher R^2 is selected. When the R^2 values of the two branches are approximately equal, within a tolerance, `abs_tolerance`, the Mean Absolute Error (MAE) is used as a secondary criterion, and the branch with the lower MAE is preferred. In the current implementation, this selection strategy is fixed, though it could be made configurable as a direction for future work.

4.1.5 Early Stopping

To avoid unnecessary computation, the iterative procedure is stopped before reaching `n_steps` if the current expression already provides a sufficient accuracy. In particular, the proposed approach allows setting a predefined target values for the accuracy, `target_MAE` and `target_R2` and the search is completed when one of the target is reached. As default, the target is defined only for $R2$.

These thresholds were chosen to ensure that the recovered expression fits the data with very high accuracy, while avoiding overfitting caused by unnecessary terms. When early stopping is triggered, the current expression is passed directly to the post-processing step without further iterations.

4.1.6 Expression Selection and Simplification

At the conclusion of the step phase, the algorithm does not necessarily return the expression obtained at the last step. Instead, it analyzes the history of all expressions generated throughout the search and selects the one that offers the best trade-off between accuracy and complexity.

The selection is done as follows: the maximum R^2 achieved across all steps is identified. Then, all expressions whose R^2 score is within a threshold `R2_threshold` from the maximum are considered as candidates:

$$\Delta R^2 = R^2_{max} - R^2_{step} \leq \texttt{R2\_threshold} \tag{4.7}$$

Among these candidates, the one corresponding to the earliest step is selected in such a way that the simplest expression that achieves a performance close to the best one found is taken. This ensures that the final expression is not unnecessarily complex.

The selected expression is then passed to a simplification step based on SymPy. The simplification performs the following operations:

- Substitution of the constant column `const_1` with the value 1
- Collection of common factors
- Rounding of floating point coefficients
- Removal of terms with coefficients below a threshold of 10^{-10}, which are considered negligible
- Algebraic simplification to reduce the expression to a canonical form using `sympy.cancel`

4.1.7 A Step-by-Step Example

To show how the algorithm progressively builds the symbolic expression, the following tables show the complete search history for the corresponding dataset.

Example 1: I.14.4 — True equation: $U = \frac{1}{2}k_{spring}x^2$

Step	Expression	Branch	R^2	MAE	Selected
0	$1.888 \cdot k_{spring} \cdot x$	—	0.856	4.081	Yes
1	$1.888k_{spring}x + 0.038e^x$	ADD	0.881	3.760	No
1	$1.888k_{spring}x \cdot 0.265x$	RATIO	1.000	≈ 0	Yes

Table 4.1: Search history for dataset I.14.4

At Step 0, the algorithm identifies $k_{spring} \cdot x$ as the most correlated term, obtaining an initial approximation with $R^2 = 0.856$. At Step 1, the ADD branch only marginally improves the fit, while the RATIO branch multiplies the prediction by a linear term in x, effectively recovering the quadratic dependency on x. The search terminates due to early stopping, and the final expression, $1.888 \cdot 0.265 \cdot k_{spring}x^2 \approx 0.5 \cdot k_{spring} \cdot x^2$, correctly recovers the structure of the true equation after simplification.

Example 2: I.12.2 — True equation: $F = \dfrac{q_1 q_2}{4\pi\epsilon r^2}$

Step	Expression	Branch	R^2	MAE	Selected
0	$0.0234 \cdot (q_1/r)^2$	—	0.531	0.0292	Yes
1	$0.0234(q_1/r)^2 + 0.0185(q_2/\epsilon)$	ADD	0.625	0.0306	No
1	$0.0234(q_1/r)^2 \cdot 1.353(q_2/q_1)$	RATIO	0.733	0.0218	Yes
2	$0.0234(q_1/r)^2 \cdot 1.353(q_2/q_1) + 0.0268(1/\epsilon)$	ADD	0.752	0.0243	No
2	$0.0234(q_1/r)^2 \cdot 1.353(q_2/q_1) \cdot 2.511(1/\epsilon)$	RATIO	1.000	≈ 0	Yes

Table 4.2: Search history for dataset I.12.2

At each step, the RATIO branch outperforms the ADD branch, indicating that the true equation has a multiplicative structure. After two steps, the algorithm correctly recovers the full expression, equivalent to $F = q_1 q_2/(4\pi\epsilon r^2)$ after algebraic simplification.

Example 3: II.21.32 — True equation: $\phi = \dfrac{q}{4\pi\epsilon r(1 - v/c)}$

Step	Expression	Branch	R^2	MAE	Selected
0	$0.0449 \cdot (q/r)$	—	0.524	0.0220	Yes
1	$0.0449(q/r) + 0.0244(1/\epsilon)$	ADD	0.575	0.0228	No
1	$0.0449(q/r) \cdot 2.445(1/\epsilon)$	RATIO	0.931	0.0069	Yes
2	$0.0449(q/r) \cdot 2.445(1/\epsilon) - 0.0003\cos(v/c)$	ADD	0.931	0.0068	No
2	$[0.0449(q/r) \cdot 2.445(1/\epsilon)] \cdot 2.669(v/c)^2 + 0.794$	RATIO	0.9998	0.0003	Yes

Table 4.3: Search history for dataset II.21.32

Also in this case, the RATIO branch is selected at every step, progressively building a multiplicative expression that approximates the true equation.

Step	Expression	Branch	R^2	MAE	Selected
0	$27.322\sin(\theta)$	—	0.666	10.51	Yes
1	$27.322\sin(\theta) - 2.083\sin(m/\theta)$	ADD	0.669	10.53	No
1	$27.322\sin(\theta) \cdot 0.110(r \cdot v)$	RATIO	0.874	6.274	Yes
2	$27.322\sin(\theta) \cdot 0.110(r \cdot v) - 2.234\sin(m/\theta)$	ADD	0.878	6.291	No
2	$27.322\sin(\theta) \cdot 0.110(r \cdot v) \cdot 0.331m$	RATIO	1.000	≈ 0	Yes

Table 4.4: Search history for dataset I.18.14

In this example, the algorithm first identifies the angular dependency $\sin(\theta)$ as the most correlated term, then progressively multiplies it by the remaining variables (m, r, v) through the RATIO branch, fully recovering the structure of the angular momentum equation.

Across all four examples, as well as throughout the complete set of Feynman datasets, the RATIO branch was selected over the ADD branch at every step of the search process. This observation is consistent with the nature of the physical laws considered in this benchmark, which are mostly expressed as products or ratios of quantities, rather than as sums.

4.2 Evaluation Metrics

The performance of both methods is evaluated using two metrics: the coefficient of determination R^2 and the Mean Absolute Error (MAE), as defined in Section 4.1.4.

R^2, also known as the coefficient of determination, is a statistical measure used to assess the goodness of fit of a regression model. Intuitively, it compares the prediction error of the model against the error of a baseline that always predicts the mean of the target variable. Particularly, an R^2 of 1 indicates a perfect fit, while an R^2 of 0 means that the model performs no better than predicting the mean of the target variable for all samples.

Negative values of R^2 are also possible and indicate that the model performs worse than the mean predictor, which can occur when the discovered expression fails to understand the structure of the data. An R^2 value of 0.80, for instance, means that 80% of the variance in the target variable is explained by the retrieved expression.

MAE measures the average absolute difference between the predicted and true values, providing a direct and interpretable measure of the prediction error in the same scale as the target variable.

Unlike R^2, MAE is not normalized and depends on the scale of the target variable, making it more suitable for comparisons among the same dataset rather than across different ones.

Chapter 5

Experimental Results

Summary

This chapter presents the experimental results obtained by running both PySR and the proposed method on the SRSD-Feynman datasets.

5.1 Experimental Setup

For the experiments conducted on the SRSD-Feynman datasets, PySR was configured with the following parameters:

`niterations=100`

: The search was run for 100 iterations, each consisting of a full evolve-simplify-optimize cycle applied to all populations.

`populations=7, population_size=400`

: The search space was explored by 7 independent island populations, each containing 400 symbolic expressions.

`maxsize=10, maxdepth=5`

The maximum number of nodes in an expression tree was set to 10, with a maximum depth of 5 levels. These constraints limit the complexity of the discovered expressions, encouraging simpler and more interpretable results.

`binary_operators=["+", "-", "*", "/"]`

The four basic arithmetic operators were used as binary operators.

`unary_operators=["square", "exp", "log", "sin", "cos"]`

Four unary transformations were allowed: square, exponential, sine and cosine.

`elementwise_loss`

A custom loss function was defined as the squared difference between prediction and target, equivalent to Mean Squared Error (MSE).

`model_selection="accuracy"`

The expression with the highest accuracy on the training data was selected as the final result, regardless of its complexity.

`deterministic=True, random_state=1`

The search was made fully reproducible by fixing the random seed to 1 and enabling deterministic mode.

`warm_start=True`

Each run resumed from the previous state, allowing the search to build on previously discovered expressions.

`turbo=False, batching=False, parallelism="serial"`

The search was run in serial mode without turbo optimization or mini-batching, ensuring a controlled and reproducible experimental environment.

For the experiments performed using the proposed approach here are the configurations:

`n_steps=6`

The maximum number of iterative steps allowed to build the symbolic expression.

`fit_intercept=False`

Linear regression is performed without intercept by default.

`R2_threshold=0.01`

The tolerance used in the expression selection step: all expressions whose R^2 is within 1% of the maximum achieved are considered as candidates.

`target_R2=0.999`

The early stopping threshold: if the R^2 of the current expression exceeds 0.999, the search is terminated before reaching `n_steps`.

5.2 Overall Results

Table 5.1 summarizes the performance of both methods on the 100 SRSD-Feynman datasets in terms of R^2, MAE, and execution time.

Table 5.1: Overall results summary

Metric	Method	Mean	Median	Min / Max
R^2	PySR	0.921	0.999	0.025 / 1.000
	Proposed	0.920	0.996	0.029 / 1.000
MAE	PySR	2.297	0.004	≈ 0 / 124.21
	Proposed	0.485	0.013	≈ 0 / 8.04
Time (s)	PySR	636.3	524.5	372.8 / 1150.5
	Proposed	0.33	0.29	0.05 / 1.06

The results show that the two methods achieve comparable performance in terms of R^2, with a mean difference of only 0.0017. However, the proposed method achieves a significantly lower MAE on average (0.48 vs 2.30), suggesting that when both methods fail to recover the exact expression, the proposed method tends to produce expressions that are numerically closer to the true values.

The most noticeable difference between the two methods is the execution time. PySR required on average 636 seconds per dataset, around 10 minutes, with a maximum of 1150 seconds. The proposed method completed each run in an average of 0.33 seconds, making it thousands of times faster than PySR while achieving comparable accuracy.

It is worth noting that the results obtained by PySR in this work are broadly consistent with those reported by Matsubara et al.[3], who evaluated PySR on the same SRSD-Feynman benchmark using different metrics, in particular solution rate and normalized edit distance (NED). In their study, PySR achieved a solution rate of 60% on the Easy set and 30% on the Medium set, meaning that the tool is able to recover the exact symbolic expression for a significant portion of the datasets.

5.3 Comparative Analysis

5.3.1 R^2 Comparison

Figure 5.1 shows a direct comparison of the R^2 values obtained by the two methods on each of the 100 datasets. Each point represents a single dataset, with PySR's R^2 on the x-axis and the proposed method's R^2 on the y-axis. Points above the diagonal indicate datasets where the proposed method outperforms PySR, while points below the diagonal indicate the opposite.

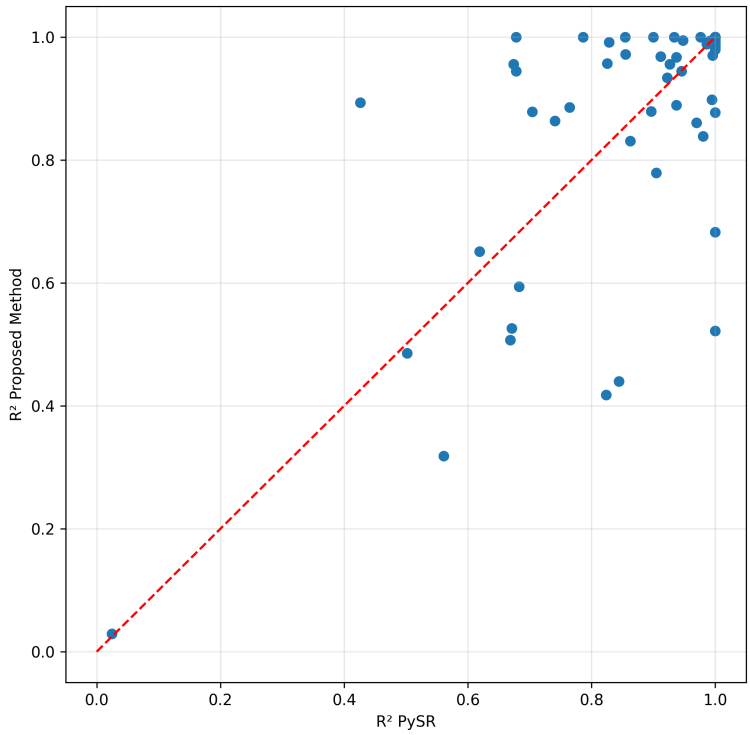

Figure 5.1: Comparison of R^2 values obtained by PySR and the proposed method.

Out of the 100 datasets, PySR achieves a higher R^2 on 40 datasets, the proposed method achieves a higher R^2 on 32 datasets, and the two methods perform equally on the remaining 28 datasets.

The majority of points are concentrated in the upper right region of the plot, indicating that both methods generally achieve high accuracy across most datasets.

5.3.2 MAE Comparison

Figure 5.2 shows the comparison of MAE values (on a logarithmic scale, due to the wide range of values observed) for the two methods across all datasets.

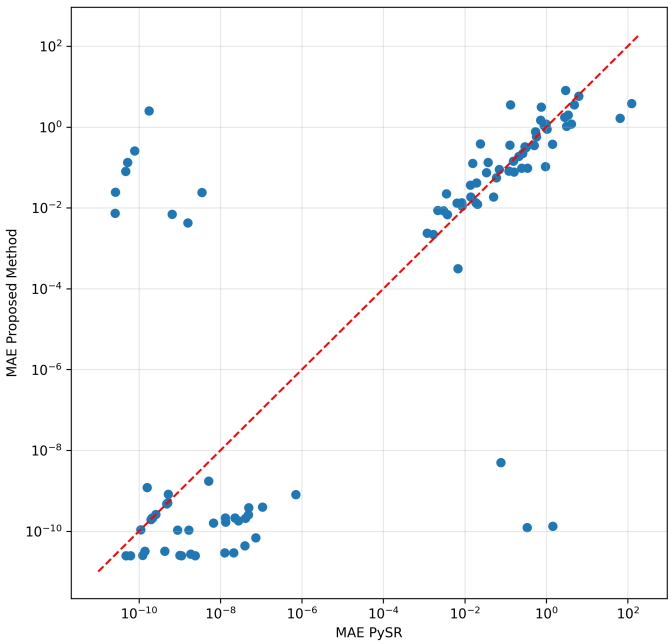

Figure 5.2: Comparison of MAE values (log scale) obtained by PySR and the proposed method.

While most points cluster around the diagonal, showing comparable performance, several points lie significantly below it, where the proposed method achieves a much lower MAE than PySR, contributing to its lower average MAE overall.

5.4 Execution Time

First of all, it is important to note that PySR and the proposed method were executed on a virtual machine hosted on Microsoft Azure (Windows Server 2022 Datacenter, Standard D4s v5, 4 vCPUs, 16 GiB RAM).

Despite running on the same hardware, the gap in execution time clearly

reflects a fundamental difference in computational complexity between the two approaches.

It is also worth noting that the PySR configuration adopted in this work (Section 5.1) is relatively modest compared to the default settings used by Cranmer [2], with a limited number of populations and a constrained maximum expression size (`maxsize=10`). This configuration was chosen to obtain acceptable results within a reasonable amount of time, given the need to run PySR independently on 100 different datasets. Despite this relatively lightweight setup, the average execution time remains much higher than that of the proposed method.

5.5 Illustrative Examples

To better illustrate the differences between the two methods, this section presents three representative examples.

Dataset I.18.12. True equation: $\tau = rF\sin\theta$

- PySR: $rF(5.88 \times 10^{-10} \cdot \theta + \sin\theta)$ $\quad (R^2 = 1.000)$
- Proposed method: $1.0000 \cdot rF\sin\theta$ $\quad (R^2 = 1.000)$

Dataset III.19.51. True equation: $E = -\dfrac{mq^4}{2(4\pi\epsilon)^2(h/2\pi)^2n^2}$

- PySR: $q^2(-q+h)/(n^2\epsilon^2)$ $\quad (R^2 = 0.678)$
- Proposed method: $-0.125\,mq^4/(h^2n^2\epsilon^2)$ $\quad (R^2 = 1.000)$

Dataset I.16.6. True equation: $w = \dfrac{u+v}{1+uv/c^2}$

- PySR: $c\cos\left(\dfrac{c-1.968}{v+u}\right)$ $\quad (R^2 = 0.945)$
- Proposed method: $0.0104\sqrt{c}\sin(1/c)\sin(c/v)\sin(c/u)\sin(u/v)+0.883$ $\quad (R^2 = 0.944)$

Dataset II.15.4. True equation: $U = -\mu B\cos\theta$, with $x_0 = \mu$, $x_1 = B$, $x_2 = \theta$.

- PySR: $-\mu(B\cos\theta - 3.04 \times 10^{-9}\theta)$ $\quad (R^2 = 1.000)$
- Proposed method: $-1.0000 \cdot \mu B\cos\theta$ $\quad (R^2 = 1.000)$

Dataset II.38.3. True equation: $F = \dfrac{YAx}{d}$

- PySR: $\dfrac{YAx}{d} - 2.25 \times 10^{-8}$ $\quad (R^2 = 1.000)$

- Proposed method: $1.0000 \cdot \dfrac{YAx}{d} \quad (R^2 = 1.000)$

Note: coefficients are rounded for readability; variable names are shown in place of $x_0, x_1, ..., x_n$ for clarity.

These examples show the overall behavior of the proposed method relative to PySR. In several cases, both methods recover expressions that are equivalent to the true equation, regardless of the number of variables or the presence of trigonometric terms.

In other cases, the two methods diverge significantly: the proposed method can sometimes find a simpler and more accurate structure than PySR, as observed for dataset III.19.51, while in other cases it produces noticeably more complex expressions with lower accuracy.

Finally, dataset I.16.6 shows that, even when the two methods achieve comparable metrics, neither of them is able to recover the correct structure of the true equation, converging instead to independent functional forms.

Table 5.2: Complete results on the 100 SRSD-Feynman datasets.

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
I.10.7	$m_0/\sqrt{1-v^2/c^2}$	$1.0\cdot(0.63924\cdot m_0\cdot v^2+0.952507\cdot c^2)/c^2$	0.9999	0.0085	$0.99873483\cdot m_0/\cos(v/(c-0.17883132))$	1.0000	0.0030
I.11.19	$x1\cdot y1+x2\cdot y2+x3\cdot y3$	$0.001515472142\cdot x1\cdot x2\cdot x3\cdot y1\cdot y2\cdot y3+0.0273156384\cdot x1\cdot x2\cdot y1\cdot y2+0.64487$	0.9561	1.1899	$x1\cdot y1+x3\cdot y3+8.934135$	0.6736	4.1093
I.12.1	$\mu\cdot Nn$	$\mu\cdot Nn$	1.0000	1.08e-10	$1.0\cdot\mu\cdot Nn-2.3125324e-10/\sin(0.7297853\cdot\mu)$	1.0000	1.65e-09
I.12.11	$q\cdot(Ef+B\cdot v\cdot\sin(\theta))$	$0.16803709772\cdot q\cdot Ef\cdot B+0.16803709772\cdot q\cdot Ef\cdot v+4.5604375372\cdot B\cdot\sin(\theta)-0.43725099798\cdot B\cdot\sin(\theta/q)-2.3425934244\cdot B\cdot\sin(\theta/B)-0.37837678022\cdot B\cdot\sin(\theta/v)+4.5604375372\cdot v\cdot\sin(\theta)-0.43725099798\cdot v\cdot\sin(\theta/q)-2.3425934244\cdot v\cdot\sin(\theta/B)-0.37837678022\cdot v\cdot\sin(\theta/v)$	0.8386	8.0410	$q\cdot(B\cdot v\cdot\sin(\theta)+3.0072086)$	0.9800	2.9818
I.12.2	$q1\cdot q2\cdot r/(4\cdot pi\cdot\varepsilon\cdot r^3)$	$0.079578\cdot q1\cdot q2/(\varepsilon\cdot r^2)$	1.0000	2.51e-11	$0.07957747\cdot q1\cdot q2/(\varepsilon\cdot r^2)$	1.0000	1.10e-09
I.12.4	$q1\cdot r/(4\cdot pi\cdot\varepsilon\cdot r^3)$	$1.0\cdot(-0.029373\cdot {q1}^2\cdot\log(q1+\varepsilon)+8.20701\cdot r^2)/r^2$	0.8776	0.0043	$0.079577465004792\cdot q1/(\varepsilon\cdot r^2)$	1.0000	1.57e-09
I.12.5	$q2\cdot Ef$	$q2\cdot Ef$	1.0000	1.08e-10	$1.0\cdot q2\cdot Ef+1.0788756e-10/\sin(0.71534497\cdot Ef)$	1.0000	8.76e-10
I.13.12	$G\cdot m1\cdot m2\cdot(1/r2-1/r1)$	$-0.29072263635\cdot {r1}^2\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}+0.29072263635\cdot r1\cdot r2\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}+0.29072263635\cdot r1\cdot G\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}+0.12741719045\cdot r1\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}\cdot e^{\cos(r1)}-0.07716471989\cdot r1\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}\cdot e^{\cos(r2)}-0.29072263635\cdot r2\cdot G\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}-0.12741719045\cdot G\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}\cdot e^{\cos(r1)}+0.07716471989\cdot G\cdot\sqrt{m1\cdot m2}\cdot\sqrt{G/r2}\cdot e^{\cos(r2)}+0.867296$	0.8636	1.7607	$0.468517432338938\cdot m2\cdot G\cdot(r1-r2)$	0.7409	2.8043
I.13.4	$1/2\cdot m\cdot(v^2+u^2+w^2)$	$1.0\cdot\sqrt{0.21116246023\cdot m^2\cdot v\cdot(u+w}\cdot\log(u+w)+0.008716163568\cdot m\cdot(u+w)^{2.5}\cdot\log(u+w))/v$	0.9675	3.5412	$m\cdot(2\cdot v+u\cdot w)$	0.9368	4.8146
I.14.3	$m\cdot g\cdot z$	$0.99999813952\cdot m\cdot g\cdot z$	1.0000	3.87e-10	$m\cdot g\cdot z-4.90000000002294e-8$	1.0000	4.90e-08
I.14.4	$1/2\cdot k_spring\cdot x^2$	$0.5000009777\cdot k_spring\cdot x^2$	1.0000	2.59e-10	$0.5\cdot k_spring\cdot x^2$	1.0000	2.57e-10
I.15.1	$m_0\cdot v/\sqrt{1-v^2/c^2}$	$1.0\cdot(0.642387\cdot m_0\cdot v^3+0.946276\cdot c^2)/c^2$	0.9999	0.0134	$m_0\cdot v/\cos((v+0.10903797)/c)$	0.9999	0.0186
I.15.3t	$(t-u\cdot x/c^2)/\sqrt{1-u^2/c^2}$	$0.996749\cdot t$	0.9903	0.0742	$(c^2\cdot t-u\cdot(x-t))/c^2$	0.9983	0.0340
I.15.3x	$(x-u\cdot t)/\sqrt{1-u^2/c^2}$	$1.0\cdot(-0.08094\cdot x\cdot {u}^2\cdot e^{u/x}\cdot\sin(u\cdot t)+0.08094\cdot {u}^2\cdot t\cdot e^{u/x}\cdot\sin(u\cdot t)+0.977172\cdot c^2)/c^2$	0.9888	0.1329	$x+x/c^2-u\cdot t$	0.9978	0.0370

Continued on next page

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
I.16.6	$(u+v)/(1+u\cdot v/c^2)$	$0.010351\cdot\sqrt{c}\cdot\sin(c\cdot\cdot(-1.0))\cdot\sin(c/v)\cdot\sin(c/u)\cdot\sin(u/v)+0.883412$	0.9445	0.1907	$c\cdot\cos((c-1.9676447)/(v+u))$	0.9452	0.2091
I.18.12	$r\cdot F\cdot\sin(\theta)$	$1.00000484304\cdot r\cdot F\cdot\sin(\theta)$	1.0000	1.70e-10	$r\cdot F\cdot(5.875961e-10\cdot\theta+\sin(\theta))$	1.0000	1.34e-08
I.18.14	$m\cdot r\cdot v\cdot\sin(\theta)$	$1.00000426717682\cdot m\cdot r\cdot v\cdot\sin(\theta)$	1.0000	8.19e-10	$m\cdot r\cdot v\cdot\sin(\theta)$	1.0000	5.21e-10
I.18.4	$(m1\cdot r1+m2\cdot r2)/(m1+m2)$	$3.3e-5\cdot r1\cdot\sin(r2/m1)\cdot\sin(m1/r1)\cdot\sin(r1/m2)\cdot\sin(m2/r2)+3.3e-5\cdot r2\cdot\sin(r2/m1)\cdot\sin(m1/r1)\cdot\sin(r1/m2)\cdot\sin(m2/r2)+1.06724$	0.9562	0.1427	$r2+(r1-r2)\cdot\sin(1.3009794/m2)$	0.9264	0.1548
I.24.6	$1/2\cdot m\cdot(\omega^2+omega_0^2)\cdot 1/2\cdot x^2$	$0.212045365994842\cdot m\cdot\omega\cdot omega_0\cdot x^3+0.099822134034\cdot\omega^3\cdot x$	0.8980	3.1269	$m\cdot x^2\cdot(\omega+omega_0-1.8340527)$	0.9946	0.7467
I.25.13	q/C	q/C	1.0000	3.19e-11	$q\cdot\cos(q)/(C\cdot\cos(q)-1.674387e-11)$	1.0000	1.39e-10
I.26.2	$\arcsin(n\cdot\sin(theta2))$	$0.829106\cdot\sqrt{e^n}\cdot\sin(n)\cdot\sin(theta2)$	0.9953	0.0188	$\sin(n)\cdot\sin(theta2)/\sin(n+13.985465)$	0.9973	0.0139
I.27.6	$1/(1/d1+n/d2)$	$2.37525e-7\cdot n^4\cdot\sqrt{d1\cdot d2}\cdot\sqrt{d2/n}\cdot e^{\cos(d2)}-3.36315e-7\cdot n^4\cdot e^{\cos(d2)}\cdot\log(d2+n)+1.0418$	0.9906	0.0245	$d1\cdot d2/(d1\cdot n+d2)$	1.0000	2.65e-11
I.29.16	$\sqrt{x1^2+x2^2-2\cdot x1\cdot x2\cdot\cos(theta1-theta2)}$	$-0.409318503228\cdot e^{\cos(x1)}\cdot\log(x1+x2)\cdot\sin(theta1/theta2)+0.231364614578\cdot e^{\cos(x1)}\cdot\log(x1+x2)-0.140660921273\cdot e^{\cos(x1)}\cdot\cos(theta1/theta2)+0.877569$	0.5072	1.0878	$x1-2.238741\cdot\cos(theta1-theta2)+1.120845$	0.6684	0.8773
I.29.4	ω/c	ω/c	1.0000	3.19e-11	$\omega/c+2.3748198e-10/(c-1.0127088)$	1.0000	4.32e-10
I.30.3	$Int_0\cdot\sin(n\cdot\theta/2)^2/\sin(\theta/2)^2$	$1.0\cdot(-0.111583903298\cdot Int_0^2\cdot\sin(\theta-1.0\cdot n)-0.003645187042\cdot\theta^2\cdot e^{\theta}\cdot\sin(\theta-1.0\cdot n)+0.003221784776\cdot\theta^2\cdot e^{\theta/n}\cdot\sin(\theta-1.0\cdot n)-0.074055525922\cdot\theta^2\cdot\sin(Int_0/\theta)\cdot\sin(\theta-1.0\cdot n)-0.069131652135\cdot\theta^2\cdot\sin(\theta-1.0\cdot n)\cdot\sin(\sin(n))+0.976883\cdot\theta^2)/\theta^2$	0.4401	1.4890	$Int_0\cdot(e^{\cos(\theta)}-\cos(\theta\cdot n))$	0.8443	0.7255
I.30.5	$\arcsin(lambd/(n\cdot d))$	$1.010867\cdot lambd/(d\cdot n)$	0.9960	0.0022	$(lambd-0.046574626)/(d\cdot n-0.26197016)$	0.9994	0.0017
I.32.17	$(1/2\cdot\varepsilon\cdot c\cdot Ef^2)\cdot(8\cdot pi\cdot r^2/3)\cdot(\omega^4/(\omega^2-omega_0^2)^2)$	$1.0\cdot(16.638675\cdot\varepsilon\cdot c\cdot Ef^2\cdot r^2\cdot\omega^6+0.571182\cdot omega_0^6)/omega_0^6$	0.9918	0.1055	$Ef^2\cdot\omega^4/(r-omega_0)^2$	0.8280	0.9406
I.32.5	$q^2\cdot a^2/(6\cdot pi\cdot\varepsilon\cdot c^3)$	$0.017904\cdot a^2\cdot e^{q/c}/(\varepsilon\cdot c)$	0.8791	0.0776	$a\cdot(q-c)/(\varepsilon\cdot c^2)$	0.8964	0.1598
I.34.1	$omega_0/(1-v/c)$	$1.0\cdot\sqrt{0.965888\cdot c^2-1.9e-5\cdot v^2\cdot omega_0\cdot(omega_0/c\cdot(c-1.0\cdot omega_0)^2\cdot\sqrt{e^{omega_0}}-1.9e-5\cdot v^3\cdot\sqrt{omega_0/c}\cdot(c-1.0\cdot omega_0)^2\cdot\sqrt{e^{omega_0}})/c^2}$	0.9861	0.1330	$c\cdot omega_0/(1.0\cdot c-v)$	1.0000	5.26e-11

Continued on next page

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
I.34.14	$(1+v/c)/\sqrt{1-v^2/c^2}\cdot omega_0$	$0.287749\cdot v\cdot\log(omega_0/c)+0.287749\cdot omega_0\cdot\log(omega_0/c)+1.19377$	0.9867	0.1265	$omega_0\cdot(c+0.3710055\cdot v)/(c-0.6289945\cdot v)$	0.9998	0.0154
I.34.27	$(h/(2\cdot pi))\cdot\omega$	$0.159155\cdot\omega\cdot h$	1.0000	2.91e-11	$0.15915494\cdot\omega\cdot h+2.54878282021642e-8$	1.0000	1.27e-08
I.34.8	$q\cdot v\cdot B/p$	$0.999995\cdot q\cdot v\cdot B/p$	1.0000	2.13e-10	$q\cdot v\cdot B/p$	1.0000	2.11e-10
I.37.4	$I1+I2+2\cdot\sqrt{I1\cdot I2}\cdot\cos(delta)$	$3.35141571215\cdot\log(I1+I2)\cdot\cos(delta)+3.4860932095\cdot\log(I1+I2)$	0.9704	0.3601	$(I1+I2)\cdot(0.963656\cdot\cos(delta)+1.0002778)$	0.9955	0.1262
I.38.12	$4\cdot pi\cdot\varepsilon\cdot(h/(2\cdot pi))^2/(m\cdot q^2)$	$0.31831\cdot\varepsilon\cdot(h/q)^2/m$	1.0000	4.39e-11	$0.31830987\cdot h^2\cdot\varepsilon/(m\cdot q^2)$	1.0000	3.99e-08
I.39.1	$3/2\cdot pr\cdot V$	$1.5\cdot pr\cdot V$	1.0000	1.61e-10	$1.5\cdot pr\cdot V+6.70000000011634e-9$	1.0000	6.70e-09
I.39.11	$1/(\gamma-1)\cdot pr\cdot V$	$1.0\cdot(0.06777932642481\cdot\gamma^2\cdot V\cdot\cos(pr/\gamma)-0.079616220720498\cdot\gamma^2\cdot V+0.045603053595\cdot\gamma^2\cdot\log(\gamma+pr)+1.06678\cdot pr^2)/pr^2$	0.9830	0.2574	$pr\cdot V/(\gamma-1.0)$	1.0000	7.81e-11
I.39.22	$n\cdot kb\cdot T/V$	$0.999999\cdot n\cdot T\cdot kb/V$	1.0000	2.10e-10	$n\cdot T\cdot kb/V$	1.0000	2.09e-10
I.40.1	$n_0\cdot e^{-m\cdot g\cdot x/(kb\cdot T)}$	$0.000677792203\cdot n_0\cdot T^2\cdot kb+0.004918602758\cdot T\cdot kb\cdot\sqrt{T/x}+0.051420502672\cdot T\cdot kb\cdot\sqrt{kb/x}-0.06433120698\cdot T\cdot kb\cdot\log(m+g)+0.13755446195\cdot T\cdot kb$	0.5942	0.2237	$(n_0-x+T)/(m\cdot g)$	0.6826	0.2612
I.41.16	$h/(2\cdot pi)\cdot\omega^3/(pi^2\cdot c^2\cdot(e^{(h/(2\cdot pi)\cdot\omega/(kb\cdot T))}-1))$	$0.095467\cdot T\cdot kb\cdot(\omega/c)^2$	0.9947	0.0969	$T\cdot kb\cdot(\omega-0.86162007)\cdot e^{-c}$	0.9478	0.3429
I.43.16	$mu_drift\cdot q\cdot Volt/d$	$1.000002\cdot mu_drift\cdot q\cdot Volt/d$	1.0000	2.14e-10	$mu_drift\cdot q\cdot Volt/d$	1.0000	2.14e-10
I.43.31	$mob\cdot kb\cdot T$	$1.00000139616\cdot mob\cdot T\cdot kb$	1.0000	4.00e-10	$mob\cdot(T\cdot kb+3.55999999999967e-8)$	1.0000	1.07e-07
I.43.43	$1/(\gamma-1)\cdot kb\cdot v/A$	$1.0\cdot(0.001703\cdot kb\cdot e^{\gamma\cdot\cdot(-1.0)}\cdot e^v\cdot\cos(v/\gamma)+1.0364\cdot A)/A$	0.9929	0.0807	$kb\cdot v/(A\cdot(\gamma-1.0))$	1.0000	4.70e-11
I.44.4	$n\cdot kb\cdot T\cdot ln(V2/V1)$	$1.0\cdot(-0.107344\cdot n\cdot kb\cdot T\cdot V1^2+0.107344\cdot n\cdot kb\cdot T\cdot V1\cdot V2+0.107344\cdot n\cdot T\cdot V1^3-0.107344\cdot n\cdot T\cdot V1^2\cdot V2+0.613533\cdot V2)/V2$	0.9682	1.9959	$-n\cdot(kb-\sin(T))\cdot(V1-V2)$	0.9116	3.4210
I.47.23	$\sqrt{\gamma\cdot pr/\rho}$	$0.999998\cdot\sqrt{\gamma}\cdot\sqrt{pr/\rho}$	1.0000	5.04e-09	$(\gamma+pr)\cdot(0.15576868\cdot\rho+0.35294208)/\rho$	0.9758	0.0768
I.48.20	$m\cdot c^2/\sqrt{1-v^2/c\cdot\cdot c}$	$1.043418\cdot m\cdot c^2$	0.9977	3.5478	$m\cdot c^2/\cos(v/c)$	1.0000	0.1308
I.50.26	$x1\cdot(\cos(\omega\cdot t)+\alpha\cdot\cos(\omega\cdot t)^2)$	$0.0679419193970282\cdot x1\cdot\omega\cdot t\cdot\log(\omega\cdot\alpha)+0.0679419193970282\cdot x1\cdot t\cdot\alpha\cdot\log(\omega\cdot\alpha)$	0.3183	1.1923	$e^{exp(0.26988286\cdot x1\cdot\cos(\omega\cdot t))}$	0.5607	0.9774
I.6.2	$e^{-(\theta/\sigma)^2/2}/(\sqrt{2\cdot pi}\cdot\sigma)$	$-3.465519e-5\cdot\sqrt{\theta}\cdot e^{\sigma\cdot\theta}-5.1313e-7\cdot e^{\theta/\sigma}\cdot e^{\sigma\cdot\theta}+2.449832e-6\cdot e^{\sigma\cdot\theta}\cdot\sin^2(\sigma)+1.967936e-6\cdot e^{\sigma\cdot\theta}\cdot\cos(\sigma\cdot\theta)+7.0235372e-5\cdot e^{\sigma\cdot\theta}+0.979616$	0.8894	0.0111	$(0.13926756\cdot\sigma+0.11557541\cdot\cos(\theta)-0.040707866007732)/(\sigma-0.2922997)$	0.9369	0.0085
I.6.2a	$e^{-\theta^2/2}/\sqrt{2\cdot pi}$	$1.0\cdot(1.04774\cdot\theta+0.162279\cdot\cos(\theta))/\theta$	0.9965	0.0024	$-0.014836193\cdot(\theta-3.5520906)^3$	0.9996	0.0012

Continued on next page

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
I.6.2b	$e^{-((\theta-theta1/\sigma)^2/2)/(\sqrt{2\cdot pi}\cdot\sigma)}$	$1.0\cdot(-4.207287e{-}6\cdot\theta^2\cdot e^{theta1/\theta}\cdot\sin(\theta/theta1)\cdot\sin(\log(\sigma))\cdot\cos(\theta/theta1)+6.527997e{-}6\cdot\theta^2\cdot e^{theta1/\theta}\cdot\sin(\theta/theta1)\cdot\cos(\theta/theta1)+0.973186\cdot theta1^2)/theta1^2$	0.8788	0.0124	$0.4494567\cdot\sin(\theta)/((\sigma+0.33979473)\cdot\sin(\theta)+0.065476425)$	0.7043	0.0203
I.8.14	$\sqrt{(x2-x1)^2+(y2-y1)^2}$	$-0.000579939426\cdot y1^2\cdot y2^2\cdot\log(x1+y1)-4.3557527997684e{-}5\cdot y1^2\cdot y2^2\cdot\sin(x1/x2)\cdot\sin(y1/y2)\cdot\sin(\cos(y1))\cdot\cos(x1)-0.0001725698861982\cdot y1^2\cdot y2^2\cdot\sin(x1/x2)\cdot\sin(y1/y2)\cdot\cos(x1)+1.0795$	0.4857	0.5723	$-\cos(y1-y2)+\cos(\cos(x1))+1.487447$	0.5018	0.5735
I.9.18	$x2\cdot m1\cdot x1/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)$	$0.173752545525\cdot m2\cdot G\cdot\sqrt{x1}\cdot y2\cdot z2\cdot\log(y2+z2)\cdot\sin(m1/y1)-0.25145733145\cdot\sqrt{x1}\cdot\log(x2-1.0\cdot z1)\cdot\log(y2+z2)+3.23246$	0.9446	0.0186	$0.29475754\cdot m1\cdot m2\cdot G/x1$	0.6783	0.0508
II.10.9	$sigma_den/\varepsilon\cdot1/(1+chi)$	$1.0\cdot(-0.202532\cdot sigma_den\cdot\log(chi)+1.76755\cdot\varepsilon)/\varepsilon$	0.9982	0.0074	$sigma_den/(\varepsilon\cdot(chi+1))$	1.0000	2.59e-11
II.11.17	$n_0\cdot(1+p_d\cdot Ef\cdot\cos(\theta)/(kb\cdot T))$	$2.42222\cdot\cos(\theta)+2.00455$	0.5262	0.7717	$n_0+(1.4832397-\theta)\cdot(p_d+\cos(T))$	0.6712	0.5405
II.11.20	$n_rho\cdot p_d^2\cdot Ef/(3\cdot kb\cdot T)$	$0.333333\cdot n_rho\cdot p_d^2\cdot Ef/(kb\cdot T)$	1.0000	1.32e-10	$n_rho\cdot p_d\cdot Ef/(kb\cdot T)$	0.8545	1.4505
II.11.27	$n\cdot\alpha/(1-(n\cdot\alpha/3))\cdot\varepsilon\cdot Ef$	$0.829789399754\cdot\varepsilon\cdot Ef\cdot e^{n\cdot\alpha}$	0.9889	0.0558	$1.24153727841074\cdot n\cdot\alpha\cdot\varepsilon\cdot Ef$	0.9863	0.0597
II.11.28	$1+n\cdot\alpha/(1-(n\cdot\alpha/3))$	$0.978621\cdot e^{n\cdot\alpha}$	0.9904	0.0222	$e^{0.8973903\cdot(n+0.014456269}\cdot(\alpha+0.014503286))$	0.9998	0.0035
II.11.3	$q\cdot Ef/(m\cdot(omega_0^2-\omega^2))$	$-0.179019946521\cdot q\cdot Ef\cdot\log(m+omega_0)+0.37001850444\cdot q\cdot Ef$	0.8607	0.0366	$q\cdot Ef\cdot e^{-0.64516217\cdot omega_0}/m$	0.9696	0.0137
II.13.17	$1/(4\cdot pi\cdot\varepsilon\cdot c^2)\cdot2\cdot I/r$	$1.0\cdot(-4.0546095e{-}5\cdot\varepsilon^2\cdot c^2\cdot I^2+0.000219084174\cdot\varepsilon^2\cdot c^4\cdot\log(\varepsilon+r)+7.8554436e{-}5\cdot\varepsilon^2\cdot c^4\cdot\log(c+I)+1.22015\cdot c^2)/c^2$	0.6826	0.0069	$0.15915495\cdot I/(\varepsilon\cdot c^2\cdot r)$	1.0000	6.45e-10
II.13.23	$\rho_{c0}/\sqrt{1-v^2/c^2}$	$1.0\cdot(0.640765\cdot\rho_{c0}\cdot v^2+0.952906\cdot c^2)/c^2$	0.9999	0.0087	$\rho_{c0}/\cos((v-0.0769222)/(c-0.31712487))$	1.0000	0.0022
II.13.34	$\rho_{c0}\cdot v/\sqrt{1-v^2/c^2}$	$1.0\cdot(0.640474\cdot\rho_{c0}\cdot v^3+0.946605\cdot c^2)/c^2$	0.9999	0.0133	$\rho_{c0}\cdot v/\cos(v/(c-0.17109266))$	1.0000	0.0064
II.15.4	$-mom\cdot B\cdot\cos(\theta)$	$-0.9999989664\cdot mom\cdot B\cdot\cos(\theta)$	1.0000	1.80e-10	$-mom\cdot(B\cdot\cos(\theta)-3.0418097e{-}9\cdot\theta)$	1.0000	2.75e-08
II.15.5	$-p_d\cdot Ef\cdot\cos(\theta)$	$-1.00000148562\cdot p_d\cdot Ef\cdot\cos(\theta)$	1.0000	2.14e-10	$-p_d\cdot Ef\cdot(\cos(\theta)+2.8292029e{-}9)+1.4558873e{-}8$	1.0000	1.31e-08
II.2.42	$kappa\cdot(T2-T1)\cdot A/d$	$1.0\cdot(-1.000001\cdot kappa\cdot T1\cdot A+1.000001\cdot kappa\cdot T2\cdot A)/d$	1.0000	1.21e-09	$kappa\cdot A\cdot(-T1+T2)/d$	1.0000	1.57e-10
II.21.32	$q/(4\cdot pi\cdot\varepsilon\cdot r\cdot(1-v/c))$	$1.0\cdot(0.293277\cdot q\cdot v^2+0.793874\cdot\varepsilon\cdot r\cdot c^2)/(\varepsilon\cdot r\cdot c^2)$	0.9998	0.0003	$0.1152073\cdot q\cdot\sin(v)/(\varepsilon\cdot r)$	0.9336	0.0068
II.24.17	$\sqrt{\omega^2/c^2-pi^2/d^2}$	$1.0\cdot(-0.225939\cdot\omega\cdot c+1.05599\cdot d^2)/d^2$	0.9960	0.0416	$1.0586737\cdot\omega/c-1.151929/d$	0.9990	0.0194
II.27.16	$\varepsilon\cdot c\cdot Ef^2$	$0.999998\cdot\varepsilon\cdot(c\cdot Ef)^2/c$	1.0000	1.75e-09	$\varepsilon\cdot c\cdot Ef^2-4.99999999736822e{-}9$	1.0000	5.09e-09

Continued on next page

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
II.27.18	$\varepsilon \cdot Ef^2$	$0.9999998069 \cdot \varepsilon \cdot Ef^2$	1.0000	5.08e-10	$\varepsilon \cdot Ef^2$	1.0000	5.08e-10
II.3.24	$Pwr/(4 \cdot pi \cdot r^2)$	$0.079577 \cdot Pwr/r^2$	1.0000	2.51e-11	$0.0795774675741507 \cdot Pwr/r^2$	1.0000	2.39e-09
II.34.11	$g \cdot q \cdot B/(2 \cdot m)$	$0.499998 \cdot g \cdot q \cdot B/m$	1.0000	1.09e-10	$0.5 \cdot g \cdot q \cdot B/m$	1.0000	1.09e-10
II.34.2	$q \cdot v \cdot r/2$	$0.50000064636 \cdot q \cdot v \cdot r$	1.0000	1.97e-10	$0.5 \cdot q \cdot v \cdot r$	1.0000	1.96e-10
II.34.29a	$q \cdot h/(4 \cdot pi \cdot m)$	$0.079577 \cdot q \cdot h/m$	1.0000	2.55e-11	$0.079577471271841 \cdot q \cdot h/m$	1.0000	1.00e-09
II.34.29b	$g \cdot mom \cdot B \cdot Jz/(h/(2 \cdot pi))$	$1.0 \cdot (1.273543 \cdot g \cdot Jz \cdot mom \cdot B \cdot \log(B) + 1.273543 \cdot g \cdot Jz \cdot mom \cdot \log(B) + 0.675553 \cdot h)/h$	0.9998	1.6660	$Jz \cdot mom \cdot (g_+ B)^2/h$	0.7860	65.0050
II.34.2a	$q \cdot v/(2 \cdot pi \cdot r)$	$0.159155 \cdot q \cdot v/r$	1.0000	2.75e-11	$0.159154942569214 \cdot q \cdot v/r$	1.0000	1.87e-09
II.35.18	$n_0/(e^{mom \cdot B/(kb \cdot T)} + e^{-mom \cdot B/(kb \cdot T)})$	$0.109029 \cdot kb \cdot \sqrt{n_0 \cdot T} \cdot \sqrt{n_0/mom} - 0.109029 \cdot B \cdot \sqrt{n_0 \cdot T} \cdot \sqrt{n_0/mom}$	0.8860	0.0811	$0.14627653 \cdot n_0 \cdot (T + \sin(kb - mom))$	0.7641	0.1203
II.35.21	$n_r ho \cdot mom \cdot \tanh(mom \cdot B/(kb \cdot T))$	$-0.061908 \cdot n_r ho \cdot mom \cdot kb \cdot \log(mom + B) - 0.061908 \cdot n_r ho \cdot mom \cdot T \cdot \log(mom + B) + 1.88041$	0.9342	0.8973	$n_r ho \cdot (mom \cdot B - 0.61953235 \cdot kb)/B$	0.9224	1.0413
II.36.38	$mom \cdot H/(kb \cdot T) + (mom \cdot \alpha)/(\varepsilon \cdot c^2 \cdot kb \cdot T) \cdot M$	$1.0 \cdot \sqrt{-0.010429 \cdot mom \cdot H \cdot \alpha^2 \cdot M \cdot (H \cdot \alpha + 1.39426 \cdot kb \cdot T \cdot \varepsilon \cdot c^2)/(kb \cdot T \cdot \varepsilon \cdot c^2)}$	0.9718	0.0960	$mom \cdot (H + \sin(c))/(kb \cdot T)$	0.8550	0.2502
II.37.1	$mom \cdot (1 + chi) \cdot B$	$1.000002 \cdot mom \cdot B \cdot chi + 1.000002 \cdot mom \cdot B$	1.0000	4.77e-10	$mom \cdot B \cdot (chi + 1)$	1.0000	4.77e-10
II.38.14	$Y/(2 \cdot (1 + \sigma))$	$0.047182 \cdot Y \cdot \sqrt{Y/\sigma} - 0.047182 \cdot \sigma \cdot \sqrt{Y/\sigma} + 0.888834$	0.9798	0.0240	$Y/(2 \cdot \sigma + 2.000000056)$	1.0000	3.52e-09
II.38.3	$Y \cdot A \cdot x/d$	$1.000004 \cdot Y \cdot A \cdot x/d$	1.0000	2.13e-10	$Y \cdot A \cdot x/d - 2.2459004e-8$	1.0000	2.25e-08
II.4.23	$q/(4 \cdot pi \cdot \varepsilon \cdot r)$	$0.079577 \cdot q/(\varepsilon \cdot r)$	1.0000	2.48e-11	$0.079577471666119 \cdot q/(\varepsilon \cdot r)$	1.0000	6.20e-11
II.6.11	$1/(4 \cdot pi \cdot \varepsilon) \cdot p_d \cdot \cos(\theta)/r^2$	$1.0 \cdot (-0.00022237072 \cdot p_d^2 \cdot \theta^2 \cdot r^2 \cdot \cos(\varepsilon/p_d) - 0.00979393349 \cdot \theta^2 \cdot \cos(\varepsilon/p_d) - 0.00332059354 \cdot r^2 \cdot \log(\sin(\theta)) \cdot \cos(\varepsilon/p_d) + 0.00890658879 \cdot r^2 \cdot \sin(\theta + r) \cdot \cos(\varepsilon/p_d) + 0.01397621357 \cdot r^2 \cdot \cos(\theta/\varepsilon) \cdot \cos(\varepsilon/p_d) + 0.829448 \cdot r^2)/r^2$	0.7793	0.0069	$\sin(0.158481675239905/r^2) \cdot \cos(\theta)/\varepsilon$	0.9043	0.0037
II.6.15a	$p_d/(4 \cdot pi \cdot \varepsilon) \cdot 3 \cdot z/r^5 \cdot \sqrt{x^2 + y^2}$	$1.0 \cdot (0.0367921497790842 \cdot r^2 \cdot e^{p_d/\varepsilon} \cdot e^{z/r} \cdot \cos(r/\varepsilon) - 0.09507477692 \cdot r^2 \cdot \sin(r)^2 \cdot \cos(r/\varepsilon) + 0.419730527714674 \cdot e^{p_d/\varepsilon} \cdot \cos(r/\varepsilon))/r^2$	0.8307	0.0894	$p_d \cdot z/(\varepsilon \cdot r^8)$	0.8628	0.0703
II.6.15b	$p_d/(4 \cdot pi \cdot \varepsilon) \cdot 3 \cdot \cos(\theta) \cdot \sin(\theta)/r^3$	$0.045595 \cdot \cos(\theta/r) - 0.028992$	0.4177	0.0135	$\sin(2.029414 \cdot \theta)/(\varepsilon + r^2)^2$	0.8233	0.0084
II.8.31	$\varepsilon \cdot Ef^2/2$	$0.50000082373 \cdot \varepsilon \cdot Ef^2$	1.0000	2.56e-10	$\varepsilon \cdot (0.5 \cdot Ef^2 + 1.6274726e-8)$	1.0000	4.88e-08
II.8.7	$3/5 \cdot q^2/(4 \cdot pi \cdot \varepsilon \cdot d)$	$0.047747 \cdot q^2/(\varepsilon \cdot d)$	1.0000	2.51e-11	$0.047746483 \cdot q^2/(\varepsilon \cdot d)$	1.0000	1.24e-10
III.10.19	$mom \cdot \sqrt{Bx^2 + By^2 + Bz^2}$	$1.0 \cdot (0.001481 \cdot mom \cdot Bx^2 \cdot By \cdot \log(Bz/Bx) + 0.001481 \cdot mom \cdot Bx^2 \cdot Bz \cdot \log(Bz/Bx) + 0.999012 \cdot By \cdot Bz)/(By \cdot Bz)$	0.9939	0.3498	$0.60340136 \cdot mom \cdot (Bx + By + Bz)$	0.9912	0.5070
III.12.43	$n \cdot (h/(2 \cdot pi))$	$0.159155 \cdot n \cdot h$	1.0000	2.93e-11	$h \cdot (0.15915495 \cdot n - 2.2390224849525e-8)$	1.0000	2.09e-08

Continued on next page

Dataset	True Expression	Proposed Method Equation	R^2	MAE	PySR Equation	R^2	MAE
III.13.18	$2 \cdot E_n \cdot d^2 \cdot k/(h/(2 \cdot pi))$	$1.0 \cdot (2.550267 \cdot E_n \cdot d^2 \cdot k \cdot \log(E_n) + 2.550267 \cdot d^2 \cdot k \cdot \log(E_n) + 0.675439 \cdot h)/h$	0.9999	3.7986	$47.402508 \cdot E_n \cdot d \cdot k/h$	0.8997	124.2108
III.14.14	$I_0 \cdot (e^{q \cdot Volt/(kb \cdot T)} - 1)$	$1.0 \cdot (-0.181357 \cdot I_0 \cdot q \cdot Volt \cdot e^{Volt/kb} \cdot \cos(q/T) + 0.181357 \cdot I_0 \cdot q \cdot kb \cdot e^{Volt/kb} \cdot \cos(q/T) - 0.181357 \cdot I_0 \cdot Volt \cdot T \cdot e^{Volt/kb} \cdot \cos(q/T) + 0.181357 \cdot I_0 \cdot kb \cdot T \cdot e^{Volt/kb} \cdot \cos(q/T))/T$	0.9573	0.3757	$I_0 \cdot q^2 \cdot Volt^2/(kb \cdot T^2)$	0.8253	1.4159
III.15.12	$2 \cdot U \cdot (1 - \cos(k \cdot d))$	$0.478370612412 \cdot U \cdot \cos(k \cdot d) \cdot \cos(k + d) + 0.00170059554 \cdot U^2 \cdot k^2 \cdot \cos(k \cdot d) + 0.000135494604 \cdot e^k \cdot e^d \cdot \cos(k \cdot d) - 0.01747143006 \cdot e^{U/d} \cdot \cos(k \cdot d) + 0.946122$	0.8934	1.0502	$2.05741318477603 \cdot U + \sin(k \cdot d - 1.548729)$	0.4258	3.1066
III.15.14	$(h/(2 \cdot pi))^2/(2 \cdot E_n \cdot d^2)$	$0.012665 \cdot h^2/(E_n \cdot d^2)$	1.0000	2.48e-11	$0.012665148 \cdot h^2/(E_n \cdot d^2)$	1.0000	4.80e-11
III.15.27	$2 \cdot pi \cdot \alpha/(n \cdot d)$	$6.283188 \cdot \alpha/(n \cdot d)$	1.0000	6.89e-11	$6.28318515443566 \cdot \alpha/(n \cdot d)$	1.0000	7.38e-08
III.17.37	$\beta \cdot (1 + \alpha \cdot \cos(\theta))$	$-0.0125102592 \cdot \beta^2 \cdot \alpha \cdot \theta^2 - 0.007450822656 \cdot \beta^2 \cdot \theta^2 \cdot \sin(\alpha/\theta) + 0.05358882816 \cdot \beta^2 \cdot \theta^2 \cdot \cos(\theta) - 0.0125102592 \cdot \beta^2 \cdot \theta^2$	0.5221	2.5037	$\beta \cdot (\alpha \cdot \cos(\theta) + 1.0)$	1.0000	1.78e-10
III.19.51	$-m \cdot q^4/(2 \cdot (4 \cdot pi \cdot \varepsilon)^2 \cdot (h/(2 \cdot pi))^2) \cdot (1/n^2)$	$-0.125 \cdot m \cdot q^4/(h^2 \cdot n^2 \cdot \varepsilon^2)$	1.0000	1.24e-10	$q^2 \cdot (-q + h)/(n^2 \cdot \varepsilon^2)$	0.6781	0.3407
III.21.20	$-\rho_{c0} \cdot q \cdot A_{vec}/m$	$-1.000002 \cdot \rho_{c0} \cdot q \cdot A_{vec}/m$	1.0000	2.09e-10	$-\rho_{c0} \cdot (q \cdot A_{vec} + 1.3760186e-8 \cdot m)/m$	1.0000	4.11e-08
III.4.32	$1/(e^{(h/(2 \cdot pi)) \cdot \omega/(kb \cdot T)} - 1)$	$5.94265 \cdot kb \cdot T/(h \cdot \omega)$	0.9972	0.3246	$6.116637 \cdot kb \cdot T/(h \cdot \omega)$	0.9987	0.3002
III.4.33	$(h/(2 \cdot pi)) \cdot \omega/(e^{(h/(2 \cdot pi)) \cdot \omega/(kb \cdot T)} - 1)$	$0.9413 \cdot kb \cdot T$	0.9910	0.3865	$-\omega \cdot \sin(0.07585779 \cdot h) + kb \cdot T$	0.9999	0.0239
III.7.38	$2 \cdot mom \cdot B/(h/(2 \cdot pi))$	$12.566363 \cdot mom \cdot B/h$	1.0000	8.10e-10	$12.56637080888 \cdot mom \cdot B/h$	1.0000	7.13e-07
III.8.54	$\sin(E_n \cdot t/(h/(2 \cdot pi)))^2$	$0.093795221872 \cdot \sin(h/t) \cdot \cos(E_n \cdot \cdot (-1.0)) + 0.184080076304 \cdot \cos(E_n \cdot \cdot (-1.0)) + 0.599254$	0.0289	0.3107	$0.59368783 - 0.26843357 \cdot h \cdot e^{-t}/E_n$	0.0245	0.3117
III.9.52	$(p_d \cdot Ef \cdot t/(h/(2 \cdot pi))) \cdot \sin((\omega - omega_0) \cdot t/2)^2/((\omega - omega_0) \cdot t/2)^2$	$1.0 \cdot \sqrt{-0.634807 \cdot p_d \cdot \sqrt{Ef} \cdot \omega^2 \cdot (Ef \cdot t \cdot \sin(\omega/omega_0) + 1.13368 \cdot h \cdot omega_0^2)/(h \cdot omega_0^2)}$	0.6513	5.8640	$12.091013 \cdot p_d/h + 12.091013 \cdot \cos(\omega - omega_0)$	0.6189	6.2027

Chapter 6

Conclusions

This thesis presented an alternative approach to Symbolic Regression, based on automated feature expansion and iterative linear regression, and compared it against PySR, a state-of-the-art evolutionary SR tool, on the SRSD-Feynman benchmark. Both methods were evaluated on 100 datasets, each corresponding to a physical law, using R^2, MAE, and execution time as evaluation metrics.

The two methods achieve comparable R^2, with the proposed method achieving a lower average MAE. The main difference between the two approaches is execution time: the proposed method completed each run in a fraction of a second, much faster than PySR on average, while remaining competitive in terms of accuracy.

The proposed method is deterministic and computationally lightweight, and does not require the configuration of evolutionary parameters such as population size or number of generations.

A limitation is its greedy nature: at each step, the algorithm selects the branch that appears most promising at that moment, without considering how this choice may constrain future steps.

Once a term is selected and incorporated into the expression, the search must continue from that point, regardless of whether a different choice might have led

to a better overall result and this does not guarantee a globally optimal solution.

A possible direction to mitigate this limitation would be to maintain multiple active branches simultaneously, rather than selecting a single one at each step. Each branch would be expanded independently for a number of steps, allowing a branch that appears suboptimal at step t to potentially outperform others at step $t + n$, if its subsequent expansion leads to a more accurate expression with respect to other branches. This approach would increase the diversity of the search and reduce the risk of getting stuck in locally optimal but globally suboptimal solutions.

A related direction concerns the stopping criterion. While increasing `n_steps` allows the algorithm to explore longer expressions, it also increases execution time proportionally. A more efficient alternative would be to introduce a patience parameter, defining the maximum number of consecutive steps without a meaningful improvement in accuracy. This would allow the search to continue only when there is evidence of progress, stopping early otherwise, and therefore balancing exploration depth with computational efficiency.

Moreover, a known limit of Symbolic Regression regards the interpretability of the discovered expressions: this is also confirmed by the proposed method: in some examples, a high accuracy is achieved but the related expressions are difficult to relate to the true physical law.

Another possible improvement would be extending the post-processing step with a check based on dimensional analysis: since each physical variable has a unit of measurement, such as length, time and charge, terms that are not dimensionally consistent could be identified and discarded.

This would guide the simplification process towards expressions that are not only numerically accurate, but also physically meaningful and easier to interpret.

Overall, this work shows that a simple, deterministic approach can be a practi-

cal alternative to evolutionary methods for Symbolic Regression, especially when speed matters.

Adding physical constraints, such as dimensional consistency, could improve the quality of the results while keeping the method fast and simple.

References

[1] D. Angelis, F. Sofos, and T. E. Karakasidis, "Artificial intelligence in physical sciences: Symbolic regression trends and perspectives," *Archives of Computational Methods in Engineering*, vol. 30, pp. 3845–3865, 2023. vi, 1, 2, 8, 9, 10, 11, 12

[2] M. Cranmer, "Interpretable machine learning for science with pysr and symbolicregression.jl," *arXiv preprint arXiv:2305.01582*, 2023. 2, 3, 14, 15, 18, 22, 23, 31, 45

[3] Y. Matsubara, N. Chiba, R. Igarashi, and Y. Ushiku, "Rethinking symbolic regression datasets and benchmarks for scientific discovery," *Journal of Data-centric Machine Learning Research*, 2024. 2, 3, 9, 18, 20, 21, 42

[4] Rulex, "People-first decision intelligence platform," 2024. Accessed: 2026-06-26. 3

[5] C. Muselli, D. Verda, E. Ferrari, C. T. Gaggiotti, and M. Muselli, "Rulex platform: leveraging domain knowledge and data-driven rules to support decisions in the fintech sector through explainable ai models.," in *xAI (Late-breaking Work, Demos, Doctoral Consortium)*, pp. 305–312, 2024. 3, 20

[6] N. Makke and S. Chawla, "Interpretable scientific discovery with symbolic regression: a review," *Artificial Intelligence Review*, vol. 57, no. 1, p. 2, 2024. 7, 8

[7] K. Papastamatiou, F. Sofos, and T. E. Karakasidis, "Machine learning symbolic equations for diffusion with physics-based descriptions," *AIP Advances*, vol. 12, no. 2, 2022. 8

[8] M. Z. Asadzadeh, H.-P. Gänser, and M. Mücke, "Symbolic regression based hybrid semiparametric modelling of processes: an example case of a bending process," *Applications in Engineering Science*, vol. 6, p. 100049, 2021. 8

[9] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature machine intelligence*, vol. 1, no. 5, pp. 206–215, 2019. 8

[10] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009. 8

[11] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994. 8, 9

[12] K. Danai and W. G. La Cava, "Controller design by symbolic regression," *Mechanical Systems and Signal Processing*, vol. 151, p. 107348, 2021. 8

[13] D. Wadekar, F. Villaescusa-Navarro, S. Ho, and L. Perreault-Levasseur, "Modeling assembly bias with machine learning and symbolic regression," *arXiv preprint arXiv:2012.00111*, 2020. 9

[14] W. Gilpin, "Chaos as an interpretable benchmark for forecasting and data-driven modelling," *arXiv preprint arXiv:2110.05266*, 2021. 9

[15] C. Wilstrup and J. Kasak, "Symbolic regression outperforms other models for small data sets," *arXiv preprint arXiv:2103.15147*, 2021. 9

[16] G. F. Bomarito, P. E. Leser, N. C. Strauss, K. M. Garbrecht, and J. D. Hochhalter, "Bayesian model selection for reducing bloat and overfitting in genetic programming for symbolic regression," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 526–529, 2022. 9

[17] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996. 10

[18] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015. 10

[19] T. Tohme, D. Liu, and K. Youcef-Toumi, "Gsr: A generalized symbolic regression approach," *arXiv preprint arXiv:2205.15569*, 2022. 12, 13

[20] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo, "Neural symbolic regression that scales," in *International conference on machine learning*, pp. 936–945, Pmlr, 2021. 12

[21] S.-M. Udrescu and M. Tegmark, "Symbolic pregression: Discovering physical laws from distorted video," *Physical Review E*, vol. 103, no. 4, p. 043307, 2021. 12, 13, 17

[22] R. K. McRee, "Symbolic regression using nearest neighbor indexing," in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pp. 1983–1990, 2010. 12, 13

[23] T. McConaghy, "Ffx: Fast, scalable, deterministic symbolic regression technology," in *Genetic Programming Theory and Practice IX*, pp. 235–260, Springer, 2011. 12

[24] V. Austel, S. Dash, O. Gunluk, L. Horesh, L. Liberti, G. Nannicini, and B. Schieber, "Globally optimal symbolic regression," *arXiv preprint arXiv:1710.10720*, 2017. 13

[25] M. R. Engle and N. V. Sahinidis, "Deterministic symbolic regression with derivative information: General methodology and application to equations of state," *AIChE Journal*, vol. 68, no. 6, p. e17457, 2022. 13

[26] M. Valipour, B. You, M. Panju, and A. Ghodsi, "Symbolicgpt: A generative transformer model for symbolic regression," *arXiv preprint arXiv:2106.14131*, 2021. 13

[27] C. Gong, J. Bryan, A. Furcoiu, Q. Su, and R. Grobe, "Evolutionary symbolic regression from a probabilistic perspective," *SN Computer Science*, vol. 3, no. 3, p. 209, 2022. 13

[28] J. Brence, L. Todorovski, and S. Džeroski, "Probabilistic grammars for equation discovery," *Knowledge-Based Systems*, vol. 224, p. 107077, 2021. 13

[29] D. Vázquez, R. Guimerà, M. Sales-Pardo, and G. Guillen-Gosalbez, "Automatic modeling of socioeconomic drivers of energy consumption and pollution using bayesian symbolic regression," *Sustainable Production and Consumption*, vol. 30, pp. 596–607, 2022. 13

[30] M. Virgolin, E. Medvet, T. Alderliesten, and P. A. Bosman, "Less is more: A call to focus on simpler models in genetic programming for interpretable machine learning," *arXiv preprint arXiv:2204.02046*, 2022. 13

[31] J. Kubalík, E. Derner, and R. Babuška, "Multi-objective symbolic regression for physics-aware dynamic modeling," *Expert Systems with Applications*, vol. 182, p. 115210, 2021. 13

[32] B. He, Q. Lu, Q. Yang, J. Luo, and Z. Wang, "Taylor genetic programming for symbolic regression," in *Proceedings of the genetic and evolutionary computation conference*, pp. 946–954, 2022. 13

[33] H. Zhang, A. Zhou, H. Qian, and H. Zhang, "Ps-tree: A piecewise symbolic regression tree," *Swarm and Evolutionary Computation*, vol. 71, p. 101061, 2022. 13

[34] P. A. Reinbold, L. M. Kageorge, M. F. Schatz, and R. O. Grigoriev, "Robust learning from noisy, incomplete, high-dimensional experimental data via physically constrained symbolic regression," *Nature communications*, vol. 12, no. 1, p. 3219, 2021. 13

[35] G. Kronberger, F. O. de França, B. Burlacu, C. Haider, and M. Kommenda, "Shape-constrained symbolic regression—improving extrapolation with prior knowledge," *Evolutionary computation*, vol. 30, no. 1, pp. 75–98, 2022. 13

[36] T. E. Karakasidis, F. Sofos, and C. Tsonos, "The electrical conductivity of ionic liquids: numerical and analytical machine learning approaches," *Fluids*, vol. 7, no. 10, p. 321, 2022. 13

[37] D. Rivero, E. Fernandez-Blanco, and A. Pazos, "Dome: A deterministic technique for equation development and symbolic regression," *Expert Systems with Applications*, vol. 198, p. 116712, 2022. 13

[38] I. Icke and J. C. Bongard, "Improving genetic programming based symbolic regression using deterministic machine learning," in *2013 IEEE Congress on Evolutionary Computation*, pp. 1763–1770, IEEE, 2013. 13

[39] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen, "Symbolic regression via neural-guided genetic programming population seeding," *arXiv preprint arXiv:2111.00053*, 2021. 13

[40] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić, "Integration of neural network-based symbolic regression in deep learning for scientific discovery," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 9, pp. 4166–4177, 2020. 14

[41] B. K. Petersen, M. Landajuela, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," *arXiv preprint arXiv:1912.04871*, 2019. 14, 17

[42] M. Cranmer, D. Tamayo, H. Rein, P. Battaglia, S. Hadden, P. J. Armitage, S. Ho, and D. N. Spergel, "A bayesian neural network predicts the dissolution of compact planetary systems," *Proceedings of the National Academy of Sciences*, vol. 118, no. 40, p. e2026053118, 2021. 14

[43] R. McElreath, *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018. 14

[44] R. Dubčáková, "Eureqa: software review," 2011. 14

[45] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, *et al.*, "Architecture and design of the heuristiclab optimization environment," in *Advanced methods and applications in computational intelligence*, pp. 197–261, Springer, 2014. 14

[46] W. La Cava, B. Burlacu, M. Virgolin, M. Kommenda, P. Orzechowski, F. O. de França, Y. Jin, and J. H. Moore, "Contemporary symbolic regression methods and their relative performance," *Advances in neural information processing systems*, vol. 2021, no. DB1, p. 1, 2021. 14, 17

[47] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983. 15

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011. 16

[49] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*, vol. 1, pp. 69–93, Elsevier, 1991. 17

[50] B. Burlacu, G. Kronberger, and M. Kommenda, "Operon c++ an efficient genetic programming framework for symbolic regression," in *Proceedings of the 2020 genetic and evolutionary computation conference companion*, pp. 1562–1570, 2020. 17

[51] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *International conference on machine learning*, pp. 4442–4450, Pmlr, 2018. 18

[52] K. R. Broløs, M. V. Machado, C. Cave, J. Kasak, V. Stentoft-Hansen, V. G. Batanero, T. Jelen, and C. Wilstrup, "An approach to symbolic regression using feyn," *arXiv preprint arXiv:2104.05417*, 2021. 18

[53] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10269–10281, 2022. 18

[54] C. Loftis, K. Yuan, Y. Zhao, M. Hu, and J. Hu, "Lattice thermal conductivity prediction using symbolic regression and machine learning," *The Journal of Physical Chemistry A*, vol. 125, no. 1, pp. 435–450, 2020. 18

[55] S. Xie, Y. Quan, A. Hire, B. Deng, J. DeStefano, I. Salinas, U. Shah, L. Fanfarillo, J. Lim, J. Kim, *et al.*, "Machine learning of superconducting critical temperature from eliashberg theory," *npj Computational Materials*, vol. 8, no. 1, p. 14, 2022. 18

[56] L. Jiang, H. Fu, H. Zhang, and J. Xie, "Physical mechanism interpretation of polycrystalline metals' yield strength via a data-driven method: A novel hall–petch relationship," *Acta Materialia*, vol. 231, p. 117868, 2022. 18

[57] W. T. Hale, E. Safikou, and G. M. Bollas, "Inference of faults through symbolic regression of system data," *Computers & Chemical Engineering*, vol. 157, p. 107619, 2022. 19

[58] H. Jeong, J. H. Kim, S.-H. Choi, S. Lee, I. Heo, and K. S. Kim, "Semantic cluster operator for symbolic regression and its applications," *Advances in Engineering Software*, vol. 172, p. 103174, 2022. 19

[59] S. S. M. Astarabadi and M. M. Ebadzadeh, "A decomposition method for symbolic regression problems," *Applied Soft Computing*, vol. 62, pp. 514–523, 2018. 19

[60] P. Lemos, N. Jeffrey, M. Cranmer, S. Ho, and P. Battaglia, "Rediscovering orbital mechanics with machine learning," *Machine Learning: Science and Technology*, vol. 4, no. 4, p. 045002, 2023. 19

[61] D. Wadekar, L. Thiele, J. C. Hill, S. Pandey, F. Villaescusa-Navarro, D. N. Spergel, M. Cranmer, D. Nagai, D. Anglés-Alcázar, S. Ho, *et al.*, "The sz flux-mass (y–m) relation at low-halo masses: improvements with symbolic regression and strong constraints on baryonic feedback," *Monthly Notices of the Royal Astronomical Society*, vol. 522, no. 2, pp. 2628–2643, 2023. 19

[62] K. W. Wong and M. Cranmer, "Automated discovery of interpretable gravitational-wave population models," *arXiv preprint arXiv:2207.12409*, 2022. 19

[63] A. Grundner, T. Beucler, P. Gentine, and V. Eyring, "Data-driven equation discovery of a cloud cover parameterization," *Journal of Advances in Modeling Earth Systems*, vol. 16, no. 3, p. e2023MS003763, 2024. 19

[64] S. Verstyuk and M. R. Douglas, "Machine learning the gravity equation for international trade," *Available at SSRN 4053795*, 2022. 19

[65] M. Morales-Alvarado, D. Conde, J. Bendavid, V. Sanz, and M. Ubiali, "Symbolic regression for precision LHC physics," in *Postponed: Machine Learning and the Physical Sciences: Workshop at NeurIPS 2024*, 12 2024. 19

[66] A. Butter, T. Plehn, N. Soybelman, and J. Brehmer, "Back to the formula-lhc edition," *SciPost Physics*, vol. 16, no. 1, p. 037, 2024. 19

[67] C. J. Soelistyo and A. R. Lowe, "Discovering interpretable models of scientific image data with deep learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6884–6893, 2024. 19

[68] S.-M. Udrescu and M. Tegmark, "Ai feynman: A physics-inspired method for symbolic regression," *Science advances*, vol. 6, no. 16, p. eaay2631, 2020. 21