

Article

FastSymbolicGP: A Lightweight Python Library for Efficient Symbolic Regression and Classification

Nikola Anđelić 

Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Vukovarska 58, 51000 Rijeka, Croatia; nikola.andjelic@riteh.uniri.hr

Abstract

Symbolic regression and symbolic classification generate explicit mathematical expressions that combine predictive modelling with direct model interpretability. However, symbolic learning based on genetic programming can be computationally expensive because large populations of candidate expressions must be repeatedly evaluated over multiple generations. This paper presents FastSymbolicGP, a lightweight Python library for symbolic regression, binary classification, and multiclass classification through a compact, scikit-learn-compatible interface. The library implements tree-based genetic programming, protected mathematical operators, tournament selection, subtree crossover, subtree, hoist, and point mutation, elitism, validation-aware model selection, adaptive parsimony, expression complexity analysis, and Numba-compiled postfix evaluation. FastSymbolicGP was evaluated through 940 successful benchmark runs covering real-world scientific regression, binary and multiclass classification, physical law recovery, dynamical system identification, parameter sensitivity, ablation, and scalability. Across four real-world scientific regression datasets, FastSymbolicGP achieved the highest mean test R^2 on every dataset and obtained significantly better pooled paired results than gplearn and PySR under the evaluated configurations. These results are specific to the selected hyperparameters, primitive sets, stopping criteria, and computational budgets, and should not be interpreted as evidence of universal superiority. In binary classification, it achieved a mean balanced accuracy of 0.8507, compared with 0.7458 for gplearn, while validation-based threshold optimization and class weighting further improved performance under severe class imbalance. FastSymbolicGP also achieved competitive multiclass and dynamical system results while generally producing substantially simpler models than gplearn. In the scalability experiment with 50,000 samples, FastSymbolicGP was approximately 6.20 times faster than PySR and 1.63 times faster than gplearn, while obtaining predictive performance nearly identical to PySR. physical law experiments showed that nondimensionalization increased the dimensional validity rate of recovered FastSymbolicGP expressions from 24% to 96%, while reducing runtime and expression complexity. Analysis of the stored equation pools further showed that near-optimal model selection reduced symbolic complexity by an average of 28.6% when a simpler candidate was available, with negligible predictive degradation. These results indicate that FastSymbolicGP provides a practical balance of predictive performance, computational efficiency, task coverage, and symbolic interpretability for reproducible scientific and machine learning applications.



Academic Editor: Francesco Camastra

Received: 16 June 2026

Revised: 25 July 2026

Accepted: 1 August 2026

Published: 4 August 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: symbolic regression; symbolic classification; genetic programming; interpretable machine learning; evolutionary computation; Python library; scikit-learn; model complexity; computational efficiency

1. Introduction

Symbolic machine learning aims to discover explicit mathematical expressions that describe relationships between input variables and target outputs. In contrast to many black-box machine learning models, symbolic models can be inspected, simplified, exported, and reused as analytical formulas. This property makes symbolic regression and symbolic classification particularly attractive in scientific machine learning, engineering, biomedical analysis, and other application domains in which model transparency is important [1,2].

Genetic programming (GP) is one of the most established approaches for symbolic model discovery [3,4]. In tree-based GP, candidate mathematical expressions are represented as hierarchical program structures composed of operators, functions, variables, and constants. A population of candidate expressions is iteratively improved through fitness evaluation, selection, crossover, mutation, reproduction, and elitism. The resulting model is therefore not only a numerical predictor, but also an explicit mathematical structure that can be directly examined and reported.

Despite these advantages, genetic programming-based symbolic learning is often computationally demanding. During every generation, each individual in the population must be evaluated over the available training samples. In simplified form, the main computational burden can be expressed as

$$C_{GP} \propto P \times G \times N \times C_{\text{expr}}, \quad (1)$$

where P denotes the population size, G is the number of generations, N is the number of training samples, and C_{expr} represents the cost of evaluating a symbolic expression. Consequently, runtime increases not only with the evolutionary search budget, but also with dataset size, expression depth, primitive-set complexity, and the number of repeated experiments.

This computational cost becomes particularly important in modern experimental workflows. Symbolic learning studies commonly require repeated executions across several datasets, random seeds, train–test splits, preprocessing strategies, fitness functions, and hyperparameter configurations. For example, an experiment involving ten datasets, five random seeds, twenty parameter configurations, and two compared methods already requires 2000 independent model-training runs. Even when an individual run is moderately expensive, the complete experimental study can therefore become time-consuming.

The computational limitation is also relevant to practical applications involving high-dimensional, noisy, imbalanced, or large-scale datasets. In such settings, repeated evaluation of complex symbolic trees can restrict the population size and number of generations that can be explored within an acceptable time budget. It can also make hyperparameter optimization, robustness analysis, and symbolic ensemble construction difficult. Therefore, improvements in expression evaluation and evolutionary execution can directly increase the number and quality of experiments that researchers are able to conduct.

Python has become a dominant environment for machine learning research because libraries such as scikit-learn provide consistent estimator interfaces, reproducible workflows, preprocessing utilities, evaluation metrics, and model-selection tools [5,6]. Symbolic learning libraries that follow similar conventions can be more easily integrated into existing data-analysis pipelines. Existing tools such as Distributed Evolutionary Algorithms in Python (DEAP), gplearn, and PySR provide important functionality for evolutionary computation and symbolic regression [7–9]. However, their design goals differ. General evolutionary frameworks often require users to implement substantial parts of the symbolic learning pipeline, while more advanced symbolic regression systems may introduce addi-

tional dependencies or focus primarily on regression. There remains a practical need for a lightweight Python library that combines regression, binary classification, multiclass classification, compact installation, estimator-style usage, and efficient repeated experimentation.

This paper presents FastSymbolicGP, a lightweight Python library for symbolic regression and classification based on genetic programming. The library provides a compact, scikit-learn-compatible interface and supports regression, binary classification, and multiclass classification within a unified framework. Its design emphasizes efficient vectorized expression evaluation, protected mathematical operations, reproducibility, compact symbolic expressions, and straightforward benchmarking.

The evaluation of FastSymbolicGP considers four complementary dimensions:

1. Predictive performance on regression, binary classification, and multiclass classification tasks;
2. Computational efficiency measured through fitting time, prediction time, and scalability with increasing dataset size and dimensionality;
3. Symbolic interpretability assessed through explicit equation reporting, expression size, tree depth, variable use, nonlinear-operator count, and related complexity indicators;
4. Scientific validity examined through physical law recovery, governing-equation identification, dimensional-consistency analysis, and comparison with known analytical relationships.

The main contributions of this work are as follows:

- A lightweight Python library that supports symbolic regression, binary classification, and multiclass classification through a unified, scikit-learn-compatible estimator interface.
- A detailed tree-based genetic programming implementation incorporating ramped population initialization, tournament selection, subtree crossover, subtree mutation, hoist mutation, point mutation, reproduction, elitism, protected numerical operators, duplicate control, and several mechanisms for limiting expression growth.
- An efficient symbolic-expression execution mechanism based on vectorized numerical evaluation and a Numba-compiled postfix backend, designed to reduce the computational cost of repeatedly evaluating large populations of candidate expressions.
- Explicit mechanisms for addressing the principal practical challenges of symbolic classification, including class weighting, validation-based decision-threshold optimization, probability transformation, one-versus-rest (OVR) multiclass classification, and shared symbolic representations with a softmax output layer.
- Validation-aware and complexity-aware model selection using internal validation performance, adaptive parsimony, hall-of-fame storage, Pareto equation pools, and selection of the smallest near-optimal expression.
- A reproducible benchmarking framework using matched data partitions, random seeds, preprocessing procedures, independent test sets, and transparent experimental configurations for FastSymbolicGP and the compared symbolic learning methods.
- An extensive empirical evaluation comprising 940 successful runs across real-world scientific regression, binary and multiclass classification, physical law recovery, dynamical system identification, parameter-sensitivity analysis, component ablation, and scalability experiments.
- A joint analysis of predictive performance, computational efficiency, symbolic complexity, dimensional consistency, and the trade-off between predictive accuracy and equation simplicity.

The experimental results demonstrate that FastSymbolicGP provides a favorable balance among predictive performance, computational efficiency, task coverage, and symbolic

model complexity. In binary classification, FastSymbolicGP achieved a mean balanced accuracy of 0.8507, compared with 0.7458 for gplearn, while reducing the mean fitting time from 27.30 to 18.23 s, corresponding to an approximately 1.50-fold speedup. In multiclass classification, the one-versus-rest FastSymbolicGP strategy achieved a mean macro-F1 score of 0.9385, compared with 0.8492 for the gplearn one-versus-rest baseline, and reduced the mean fitting time from 149.6 to 92.9 s, corresponding to an approximately 1.61-fold speedup. Across the four real-world scientific regression datasets, FastSymbolicGP achieved the highest mean test R^2 on every dataset and was approximately 2.10 times faster than gplearn on average. In the scalability experiment with 50,000 samples, FastSymbolicGP was approximately 6.20 times faster than PySR and 1.63 times faster than gplearn, while achieving predictive performance nearly identical to PySR. Furthermore, nondimensionalization increased the dimensional-validity rate of the recovered physical law expressions from 24% to 96%, and complexity-aware selection reduced symbolic complexity by an average of 28.6% whenever a simpler near-optimal candidate was available. These findings indicate that FastSymbolicGP is particularly suitable for reproducible symbolic learning studies in which predictive performance, computational cost, explicit equation reporting, and model interpretability must be considered jointly.

The remainder of this paper is organized according to the journal template. Section 1.1, within the Introduction, reviews symbolic regression, symbolic classification, genetic programming, and existing Python symbolic learning libraries. Section 2 presents the FastSymbolicGP library design and the complete experimental methodology, including the evolutionary operators, prediction mechanisms, datasets, experimental profiles, baseline configurations, evaluation metrics, and reproducibility settings. Section 3 reports the predictive, runtime, scientific-validity, and interpretability results. Section 4 discusses the practical implications, strengths, and limitations of the library. Finally, Section 5 summarizes the principal findings and outlines future development directions.

1.1. Related Work

1.1.1. Symbolic Regression and Scientific Model Discovery

Symbolic regression is a data-driven modelling approach in which both the structure and parameters of a mathematical expression are inferred from observations. Unlike conventional regression methods, which estimate the parameters of a predefined model form, symbolic regression searches a space of candidate analytical expressions composed of variables, constants, arithmetic operators, and mathematical functions [2–4,10]. The result is an explicit model that can be inspected, simplified, differentiated, integrated, and implemented independently of the software used to discover it.

The ability to obtain explicit analytical expressions makes symbolic regression particularly relevant to scientific discovery. Schmidt and Lipson demonstrated that symbolic search can recover free-form natural laws from experimental measurements without requiring the final equation structure to be specified in advance [1]. Subsequent studies extended data-driven equation discovery to dynamical systems, partial differential equations, astrophysics, fluid mechanics, and other scientific domains [11–13]. Physics-informed methods such as AI Feynman exploit properties including dimensional consistency, separability, symmetry, and compositional structure to reduce the search space and recover equations from scientific data [14]. Shape-constrained and physics-aware symbolic regression additionally incorporate prior knowledge such as monotonicity, positivity, convexity, or known physical relationships [15,16].

The principal advantage of symbolic regression is therefore not limited to prediction. A discovered expression may also provide a compact hypothesis about the data-generating process. This differentiates symbolic models from post hoc explanations of black-box

predictors because the symbolic expression is itself the predictive model. Nevertheless, interpretability is not guaranteed merely because an expression is symbolic. Deep trees, repeated subexpressions, many numerical constants, and excessive use of nonlinear operators may produce equations that are technically explicit but difficult for a human reader to understand [17–19].

Symbolic regression is also computationally difficult because the algorithm must search simultaneously over model structures, operators, variables, and numerical constants. Virgolin and Pissis formally showed that symbolic regression is NP-hard under the investigated formulation [20]. Consequently, practical symbolic regression systems rely on heuristic, evolutionary, probabilistic, deterministic, or neural-guided search procedures rather than exhaustive enumeration of the complete expression space.

Genetic programming remains one of the most established approaches to this problem [3,4]. However, alternative methods have also been proposed. Fast Function Extraction (FFX) uses a deterministic pathwise search to generate sparse nonlinear models [21]. AI Feynman uses recursive decomposition and physics-inspired transformations [14]. Deep symbolic regression employs recurrent neural networks and risk-seeking policy gradients to generate candidate expressions [22], while transformer-based systems attempt to predict complete expressions directly from numerical observations [23]. These methods demonstrate the diversity of modern symbolic regression, but genetic programming remains highly relevant because it directly searches expression structures, supports arbitrary primitive sets, and does not require a large pretraining corpus.

1.1.2. Genetic Programming for Symbolic Model Discovery

In tree-based genetic programming, each individual represents an executable symbolic expression. Internal nodes contain functions or operators, whereas terminal nodes contain input variables and numerical constants [3,4]. A population of such expressions is evaluated using a task-specific fitness function and improved over successive generations through selection, crossover, mutation, reproduction, and elitism.

The behaviour of a genetic programming system depends strongly on its design choices. Important components include the population-initialization strategy, primitive and terminal sets, parent-selection mechanism, genetic operators, fitness function, tree-depth limits, stopping conditions, and bloat-control strategy [3,4]. Tournament selection is frequently used because it is simple and allows selection pressure to be adjusted through the tournament size. Subtree crossover exchanges portions of two parent trees, while mutation may replace a subtree, modify a node, alter a constant, or insert a newly generated structure. Although these operations provide substantial structural flexibility, they also make the evolutionary trajectory stochastic and sensitive to parameter settings.

A major limitation of genetic programming is the cost of repeatedly evaluating candidate expressions. For a population of size P , evolved for G generations over N samples, the computational burden is approximately proportional to

$$\mathcal{O}(PGN\bar{C}_{\text{expr}}), \quad (2)$$

where $\bar{C}_{\text{expr}}$ denotes the average cost of evaluating an expression. This cost increases with tree size, primitive complexity, dataset size, and the number of random seeds or hyperparameter configurations. Numerical constant optimization can improve predictive performance, but it can also add substantial optimization overhead to every candidate model [24].

Expression bloat represents an additional challenge. Bloat occurs when program trees grow without a corresponding improvement in fitness. Large expressions require more time to evaluate, consume more memory, complicate genetic operations, and reduce model

readability [18,25,26]. Common control mechanisms include depth constraints, size limits, lexicographic parsimony, fitness penalties, dynamic parsimony pressure, multi-objective optimization, and post-evolution simplification [25–27]. However, minimizing the number of nodes alone does not necessarily guarantee generalization or human interpretability, because expressions with equal sizes may differ considerably in nonlinear structure and cognitive difficulty [17,19].

Recent work has provided direct evidence of inefficiency in conventional genetic programming search. Kronberger et al. showed that GP may explore only a small part of the space of semantically unique expressions and may repeatedly evaluate expressions that are algebraically or semantically equivalent [28]. This result is important because repeated evaluation of redundant expressions consumes computational resources without increasing meaningful search coverage. Expression caching, canonicalization, duplicate detection, vectorized execution, and optimized internal representations are therefore important directions for improving practical runtime.

The scalability problem becomes more significant in modern experimental workflows. A single study may involve multiple datasets, random seeds, cross-validation folds, evolutionary profiles, primitive sets, and parameter configurations. Large-scale benchmark studies have shown that symbolic regression can be competitive with established machine learning methods, but that its runtime is often among the highest of the evaluated approaches [29,30]. Earlier work on time-constrained symbolic regression similarly emphasized the difficulty of applying conventional GP configurations to large regression and classification problems [31]. These findings motivate lightweight implementations that reduce per-expression overhead and make repeated experimentation more feasible.

1.1.3. Symbolic Classification

Genetic programming has also been widely applied to classification [32]. In symbolic classification, an evolved expression usually produces a continuous score that must be converted into a discrete class label. For binary classification, this conversion can be performed using a fixed threshold, a sign function, or a probabilistic transformation such as the logistic sigmoid. The fitness function may optimize accuracy, balanced accuracy, area under the receiver operating characteristic curve, F1-score, or another classification-oriented criterion.

Symbolic classifiers offer an important interpretability advantage because the decision score is represented by an explicit mathematical expression. Consequently, the variables, interactions, transformations, and decision boundary used by the classifier can be examined directly. Genetic programming has also been used for interpretable feature construction, in which evolved expressions transform the original input variables before classification [33,34]. Such transformations can reduce dimensionality and expose nonlinear relationships that are difficult to identify using the original feature space.

Multiclass symbolic classification is more difficult because a single scalar expression does not naturally represent several classes. Common approaches include one-vs-rest decomposition, one-vs-one decomposition, multiple co-evolved trees, class-specific symbolic scores, interval-based output mapping, and multidimensional symbolic representations [32,35]. In a one-vs-rest strategy, one binary symbolic classifier is trained for each class and the final class is selected from the resulting decision scores. Although this strategy is easy to implement, its computational cost grows with the number of classes because multiple independent GP searches are required.

La Cava et al. proposed multidimensional genetic programming for multiclass classification, in which evolved programs construct an adaptive feature space and class assignment is performed in that space [35]. This work showed that multiclass GP need not be restricted

to a collection of isolated binary classifiers. Nevertheless, multiclass support remains less common in general-purpose symbolic learning libraries than symbolic regression or binary classification. This creates a practical gap for users who require regression, binary classification, and multiclass classification through a consistent estimator interface.

1.1.4. Symbolic Learning Libraries and Implementations

The availability of reusable software has been essential to the development of symbolic machine learning. DEAP is a general-purpose Python framework for evolutionary computation that emphasizes explicit algorithms and transparent data structures [7]. It provides the components needed to construct genetic programming systems, but users must define substantial parts of the representation, fitness evaluation, genetic operators, and evolutionary workflow. This flexibility is valuable for evolutionary computation research, although it requires more implementation effort than a task-specific estimator.

The gplearn library provides genetic programming estimators through an interface inspired by scikit-learn [8]. It includes a symbolic regressor, a binary symbolic classifier, and a symbolic transformer. Its familiar `fit`, `predict`, and `transform` methods make it convenient for Python machine learning workflows. However, its classifier is designed for binary targets, and multiclass problems therefore require an external decomposition strategy or an alternative implementation. As a predominantly Python-oriented implementation, repeated evaluation of large populations can also become costly when many datasets, seeds, and experimental configurations are considered.

DEAP and gplearn benefit from integration with the wider scientific Python ecosystem. Scikit-learn established estimator conventions that support pipelines, preprocessing, cross-validation, model selection, and standardized evaluation [5,6]. Compatibility with these conventions is particularly useful for benchmarking because compared models can share identical data partitions, preprocessing procedures, and evaluation metrics.

Operon is a C++ genetic programming framework developed for efficient symbolic regression [36]. Its implementation emphasizes efficient execution, modern optimization techniques, and numerical-parameter estimation. The existence and benchmark performance of Operon demonstrate that internal representation and execution-engine design can have a major influence on GP runtime. However, Operon focuses primarily on symbolic regression rather than a unified regression and classification application programming interface (API).

PySR provides a Python interface to the Julia-based `SymbolicRegression.jl` backend [9]. It combines multi-population evolutionary search, expression simplification, numerical constant optimization, and parallel execution. Its backend can exploit compiled operators, vectorized instructions, and distributed computation, making it suitable for demanding scientific symbolic regression tasks. These capabilities provide strong performance but require an additional Julia runtime and primarily address symbolic regression rather than general symbolic classification.

Other systems focus on different aspects of the problem. FFX emphasizes fast and deterministic sparse model construction [21]. AI Feynman uses physics-inspired decomposition [14]. Transformation–interaction representations restrict the expression space to obtain compact models [37]. Feature Engineering Automation Tool (FEAT) evolves feature representations combined with conventional parameter optimization [34]. Neural symbolic methods generate expressions using learned sequence models [22,23]. These approaches provide important alternatives, but they differ in task support, dependencies, representation, computational requirements, and compatibility with standard machine learning interfaces.

Table 1 summarizes representative symbolic learning tools and highlights the position of FastSymbolicGP. The table is not intended to identify a universally superior system. Instead, it illustrates that existing tools pursue different objectives and that no single implementation simultaneously emphasizes a lightweight Python installation, scikit-learn-style regression, binary classification, native multiclass classification, and fast repeated benchmarking.

Table 1. Comparison of representative symbolic learning tools. Task support refers to the principal functionality documented for each system.

Tool	Primary Backend	Regression	Binary Class	Native Multiclass	Estimator-Style API	Primary Emphasis or Practical Consideration
DEAP	Python	Custom	Custom	Custom	No	General evolutionary framework; the GP workflow and task-specific components must be configured by the user.
gplearn	Python/NumPy	Yes	Yes	No	Yes	Accessible scikit-learn-style symbolic regression, binary classification, and feature transformation.
PySR	Python/Julia	Yes	No	No	Partial	High-performance scientific symbolic regression with a compiled and parallel Julia backend.
Operon	C++	Yes	No	No	No	Efficient GP framework focused on symbolic regression and numerical parameter optimization.
AI Feynman	Python	Yes	No	No	No	Physics-inspired equation discovery using separability, symmetry, and recursive decomposition.
FastSymbolicGP	Python/NumPy	Yes	Yes	Yes	Yes	Lightweight unified interface for fast symbolic regression, binary classification, multiclass classification, and repeated benchmarking.

1.1.5. Benchmarking, Interpretability, and Research Gap

Reliable evaluation of symbolic learning methods requires more than reporting the best result from a single evolutionary run. Because GP is stochastic, experiments should use repeated random seeds and identical data partitions. Comparisons should also account for predictive performance, computational time, model complexity, and robustness to noise. The Penn Machine Learning Benchmark (PMLB) provides standardized datasets for broad machine learning evaluation [38], while SRBench introduced a reproducible benchmark specifically for contemporary symbolic regression methods [30].

SRBench evaluated symbolic and conventional machine learning methods on a large collection of real-world and synthetic problems. Its findings showed that modern GP-based methods can achieve strong predictive performance, particularly when evolutionary search is combined with parameter estimation or semantic search [30]. At the same time, the results confirmed that accuracy must be considered together with model complexity and computational cost. A method that produces a small improvement in prediction while generating a substantially larger expression or requiring considerably longer execution may not provide the most useful symbolic model.

Interpretability assessment remains less standardized than predictive evaluation. Expression length and tree depth are commonly used because they are objective and straightforward to calculate [17,18]. However, two expressions with the same number of nodes may differ in the number of variables, nonlinear functions, nested transformations, constants, and repeated subexpressions. More detailed analysis may therefore consider:

- The total number of expression nodes;
- The maximum tree depth;
- The number of operator and function nodes;
- The number of distinct input variables;

- The number of nonlinear or transcendental operations;
- The number of fitted numerical constants;
- The position of the model on the predictive-performance–complexity Pareto front.

Aldeia and de França emphasized that symbolic form alone is insufficient as a complete definition of interpretability and evaluated symbolic models using multiple explanatory criteria [19]. Similarly, multi-objective and Bayesian model-selection approaches have been proposed to balance prediction error against expression complexity [17,27]. Consequently, a symbolic learning library should expose both the final expression and measurable properties of its structure.

The reviewed literature reveals four practical gaps. First, conventional tree-based GP can be computationally expensive because large numbers of expressions are repeatedly evaluated, including structurally or semantically redundant candidates [28,29]. Second, general evolutionary frameworks provide flexibility but require substantial task-specific implementation [7]. Third, lightweight scikit-learn-compatible libraries commonly focus on regression and binary classification, while native multiclass symbolic classification is less frequently available [8,35]. Fourth, symbolic model evaluations often emphasize prediction and runtime without reporting sufficiently detailed expression-complexity indicators [18,19].

FastSymbolicGP is developed to address these gaps. It provides a lightweight Python implementation with a unified estimator-style interface for symbolic regression, binary classification, and multiclass classification. The library emphasizes efficient expression evaluation and reproducible repeated experimentation while retaining access to the explicit evolved model. The evaluation in this study therefore considers predictive performance, fitting time, runtime speedup, expression size, and tree depth. This joint analysis positions FastSymbolicGP not as a replacement for every existing symbolic learning system, but as a compact research tool for experiments in which computational efficiency, task coverage, reproducibility, and symbolic interpretability are required simultaneously.

2. Materials and Methods

2.1. FastSymbolicGP Library Design

FastSymbolicGP is a Python library for symbolic regression, binary symbolic classification, and multiclass symbolic classification. The library uses tree-based genetic programming to search for explicit mathematical expressions while exposing the resulting models through an estimator-oriented interface compatible with common scikit-learn workflows. Its principal design objective is to combine four properties that are not always simultaneously available in symbolic learning software: a compact application programming interface, support for several supervised learning tasks, efficient repeated evaluation of candidate expressions, and direct access to the analytical structure of the fitted model.

The library is based specifically on genetic programming, but genetic programming is not the only possible approach to symbolic regression. Alternative families include deterministic feature construction, as used by Fast Function Extraction; sparse identification of nonlinear dynamical systems; physics-inspired decomposition methods such as AI Feynman; neural symbolic regression; transformer-based equation generation; and hybrid evolutionary systems such as PySR [9,11,14,21–23]. FastSymbolicGP does not attempt to replace all of these approaches. Its methodological scope is a lightweight and extensible implementation of tree-based genetic programming for users who require explicit expressions, classification support, estimator-style use, and controllable computational cost.

The overall design follows the established principles of genetic programming introduced by Koza and subsequently developed in the evolutionary-computation literature [3,4]. Candidate solutions are represented as symbolic trees and are improved through population initialization, fitness evaluation, parent selection, crossover, mutation, elitism,

and final model selection. The public interface follows the estimator conventions used throughout the Python machine-learning ecosystem [5,6]. Consequently, a model can be constructed, fitted, evaluated, and queried using familiar methods such as `fit`, `predict`, `predict_proba`, `score`, and `get_params`.

The internal organization of FastSymbolicGP was designed to separate the user-facing estimator interface from the evolutionary search, symbolic-program representation, numerical execution, and reporting components. This separation allows regression, binary classification, and multiclass classification to share a common evolutionary foundation while retaining the task-specific procedures required for output scaling, probability estimation, threshold selection, and multiclass decision construction. Figure 1 presents the principal architectural layers of the library and summarizes the flow from the supported supervised-learning tasks to the final symbolic model outputs.

As illustrated in Figure 1, the library is organized into five interconnected layers. The estimator layer provides the public `FastSymbolicRegressor` and `FastSymbolicClassifier` interfaces and is responsible for input validation, data partitioning, model fitting, prediction, probability estimation, and scoring. The evolutionary layer manages the genetic programming process, including population initialization and maintenance, fitness evaluation, parent selection, crossover, mutation, elitism, duplicate elimination, hall-of-fame preservation, and early stopping. The program layer represents candidate solutions as symbolic expression trees and manages primitives, constants, tree constraints, directed-acyclic-graph representations, and complexity-control mechanisms. The execution layer converts the symbolic structures into efficient numerical representations and evaluates them using vectorized postfix or directed-acyclic-graph execution, protected mathematical operators, optional caching, and NumPy-based array processing. Finally, the reporting and export layer provides the selected expressions, complexity statistics, evolutionary histories, Pareto-optimal candidates, runtime information, and deployment-oriented model outputs. Together, these layers provide a unified estimator workflow while preserving explicit access to the symbolic structure and evolutionary history of the fitted models.

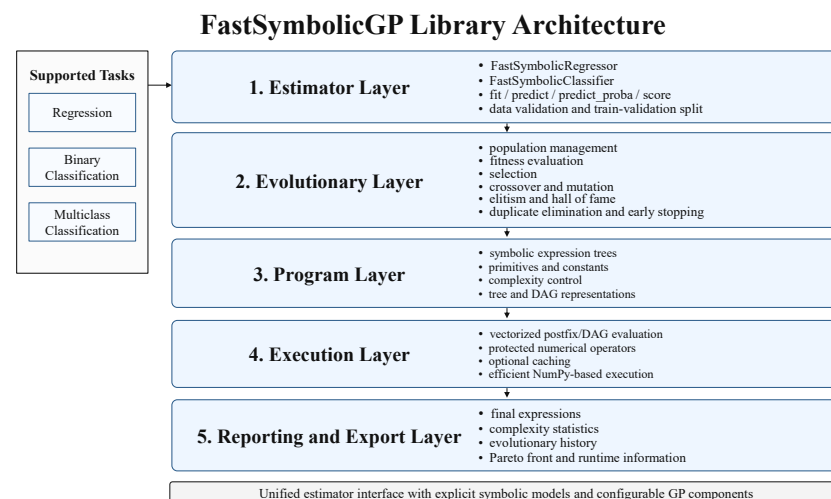


Figure 1. Layered architecture of the FastSymbolicGP library. The supported regression, binary classification, and multiclass classification tasks are exposed through a unified estimator interface, while the underlying evolutionary, symbolic-program, execution, and reporting layers provide the principal model-search and analysis functionality.

2.1.1. Design Objectives and Software Architecture

The library was designed around the following objectives:

1. **Unified supervised-learning interface.** Regression, binary classification, and multiclass classification use the same general estimator workflow, although their loss functions and output transformations differ.
2. **Explicit symbolic outputs.** The fitted model remains available as a mathematical expression rather than only as an opaque prediction object.
3. **Validation-aware evolution.** Candidate programs can be evaluated using both training and validation data, reducing the probability that the final expression is selected solely because it overfits the evolutionary training subset.
4. **Controlled expression complexity.** Tree size, tree depth, directed-acyclic-graph complexity, and algebraically simplified complexity are available for bloat control and model reporting.
5. **Efficient numerical execution.** Symbolic trees can be compiled into postfix or directed-acyclic-graph representations and evaluated over complete NumPy arrays rather than through sample-by-sample recursive Python execution.
6. **Reproducibility and inspectability.** Random seeds, evolutionary histories, Pareto fronts, expressions, feature-use statistics, cache statistics, stopping conditions, and run-time information can be retained and exported.
7. **Extensibility.** Primitive functions, fitness metrics, population models, selection modes, mutation probabilities, complexity objectives, and execution backends are configurable without changing the public estimator interface.

The software architecture separates the public estimators from the evolutionary and execution components. The estimator layer validates the data, determines the supervised-learning task, creates the training and validation partitions, and transforms symbolic outputs into predictions. The evolutionary layer manages populations, selection, variation, elitism, duplicate elimination, early stopping, and the hall of fame. The program layer implements symbolic trees and genetic operators. The execution layer provides vectorized postfix and directed-acyclic-graph evaluation, protected mathematical functions, and optional caching. Finally, the reporting layer exposes expressions, complexity measures, evolutionary histories, Pareto-optimal candidates, and deployment-oriented exports.

2.1.2. Estimator Interface and Supported Tasks

FastSymbolicGP provides separate estimator classes for regression and classification. Both estimators inherit the same evolutionary engine and therefore share parameters controlling population size, number of generations, initialization depth, operator probabilities, selection pressure, parsimony, early stopping, expression limits, execution backend, random state, and resource budgets. A typical workflow is shown in Listing 1.

When the input is a pandas data frame, the original feature names are retained and can be used in the printed expression. Otherwise, features are denoted by indices such as $X_0, X_1, \dots, X_{d-1}$. The primary task-specific behavior is summarized in Table 2. In this table, MSE denotes mean squared error, RMSE denotes root mean squared error, MAE denotes mean absolute error, MCC denotes the Matthews correlation coefficient, and ROC-AUC denotes the area under the receiver operating characteristic curve.

Listing 1. Basic estimator-style use of FastSymbolicGP.

```

1      from fastsymbolicgp import FastSymbolicClassifier
2
3      model = FastSymbolicClassifier(
4          population_size=64,
5          generations=120,
6          optimization="nsga2",
7          random_state=42
8      )
9
10     model.fit(X_train, y_train)
11
12     y_pred = model.predict(X_test)
13     y_proba = model.predict_proba(X_test)
14
15     print(model.get_expression())
16     print(model.get_expression_stats())

```

Table 2. Task-specific design of the FastSymbolicGP estimators.

Task	Symbolic Output	Output Transformation	Supported Fitness Measures
Regression	Continuous score $f(\mathbf{x})$	Linear scaling $af(\mathbf{x}) + b$	MSE, RMSE, MAE, and R^2 -based loss
Binary classification	Continuous decision score $f(\mathbf{x})$	Logistic mapping, probability calibration, and decision threshold	Log loss, accuracy, balanced accuracy, F1, MCC, ROC-AUC, and average precision
Multiclass classification	One or more symbolic class scores	One-vs-rest normalization or a shared symbolic backbone with a softmax head	Class-specific binary losses followed by multiclass probability evaluation

2.1.3. Symbolic Program Representation

Each candidate model in FastSymbolicGP is represented as a rooted symbolic-expression tree. Internal nodes correspond to primitive functions or operators, whereas terminal nodes correspond to input variables or ephemeral numerical constants. This representation is important because the same structure is used both for mathematical interpretation and for genetic programming operations such as crossover, mutation, simplification, and complexity evaluation.

For example, the symbolic program

$$g(\mathbf{x}) = (x_{17} + x_2)x_2 \quad (3)$$

is internally serialized as `mul(add(X17, X2), X2)`.

The corresponding tree representation is shown in Figure 2. The root node is the multiplication operator, its left child is the addition operator, and its right child is the terminal variable X_2 . The addition node then connects to the terminal variables X_{17} and X_2 . Thus, the expression contains five tree nodes and three tree levels.

As illustrated in Figure 2, the serialized representation maps directly to the conventional mathematical expression in Equation (3). This form is used for storage, evaluation, and program manipulation during evolution. For classification tasks, the raw symbolic output $g(\mathbf{x})$ is subsequently transformed into a probability and class label, as described in Section Binary Classification Fitness.

Each symbolic program stores its tree structure, training fitness, validation fitness, selection fitness, output-scaling coefficients, and metadata generated during evolution. The following structural properties are computed directly from the program:

- Total number of tree nodes;
- Maximum tree depth;
- Number of nodes after algebraic simplification;
- Depth after algebraic simplification;
- Number of unique nodes in the compiled directed acyclic graph (DAG);
- Number of repeated subexpressions;
- Directed-acyclic-graph compression ratio;
- Frequency of each input feature;
- Frequency of each primitive function.

Two programs may have different tree sizes but similar effective computational cost. For example, a tree may contain the same subexpression multiple times. The directed-acyclic-graph representation stores structurally identical subexpressions only once and therefore provides an additional complexity measure beyond the conventional node count. FastSymbolicGP can therefore use either tree size or directed-acyclic-graph size as the complexity objective.

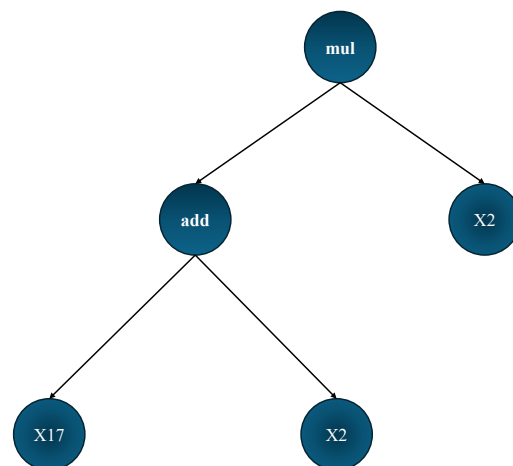


Figure 2. Tree representation of the symbolic program $g(\mathbf{x}) = (x_{17} + x_2)x_2$, internally serialized as `mul(add(x17, x2), x2)`.

2.1.4. Primitive Functions and Protected Numerical Operations

The primitive set determines the mathematical language available to the evolutionary search. The default set contains addition, subtraction, multiplication, and protected division. Additional primitives can be enabled when required by the application. The complete implemented primitive family is summarized in Table 3.

Randomly generated expressions frequently contain numerically invalid combinations. Without protection, division by zero, logarithms of negative values, square roots of negative values, and exponential overflow can propagate NaN or infinite values through the population. FastSymbolicGP therefore converts undefined or unbounded results into finite numerical values. Intermediate values are clipped to a finite interval, and invalid results are replaced by deterministic fallback values. These protections do not make every expression scientifically meaningful, but they prevent numerical exceptions from terminating the evolutionary process.

Table 3. Symbolic primitives available in FastSymbolicGP.

Category	Primitive	Numerical Definition or Protection
Binary arithmetic	add, sub, mul	Standard element-wise addition, subtraction, and multiplication
Protected division	div	Returns a/b when $ b > 10^{-12}$; otherwise returns a finite fallback value
Binary comparison	max, min	Element-wise maximum or minimum
Trigonometric	sin, cos	Element-wise sine and cosine
Nonlinear bounded	tanh	Element-wise hyperbolic tangent
Unary arithmetic	abs, neg, inv	Absolute value, negation, and protected reciprocal
Protected logarithm	log	$\log(a + 10^{-12})$
Protected square root	sqrt	$\sqrt{ a }$
Protected exponential	exp	$\exp(a)$ after clipping a to the interval $[-30, 30]$
Missing-value handling	is_missing, coalesce	Detects missing values or replaces a missing first argument with the second argument

The choice of primitive set should reflect the problem being studied. A large function set increases representational capacity but also enlarges the search space and can make the discovered equations more difficult to interpret. For many initial experiments, the arithmetic set $+, -, \times, \div$ provides an appropriate balance between expressiveness, computational cost, and interpretability. Trigonometric, logarithmic, exponential, and missing-value primitives should be enabled only when they are meaningful for the application.

2.1.5. Population Initialization

The initial population is generated using a ramped combination of full and grow tree construction. Alternating individuals are generated using the two initialization modes.

In full initialization, function nodes are created until the selected target depth is reached, after which terminal nodes are inserted. In grow initialization, terminals can be introduced before the maximum target depth, producing trees with more heterogeneous shapes. For grow initialization, a terminal is selected with probability 0.35 whenever the current depth is above the required minimum depth and below the maximum depth.

For every new tree, a target depth is sampled from the configured initialization interval. The standard interval is two to six levels, subject to the global maximum-depth and maximum-node limits. A terminal node becomes a numerical constant with the configured constant probability and otherwise becomes a randomly selected input feature. Constants are sampled uniformly from the specified constant interval.

This mixed initialization procedure reduces the probability that the initial population consists of trees with nearly identical shapes. Structural duplicates are rejected during initialization when duplicate elimination is active. The population generator also checks the maximum depth and maximum node count before accepting a program.

2.1.6. Task-Specific Fitness Evaluation

Let $f_j(\mathbf{x})$ denote the raw numerical output of candidate program j . The program structure is shared across tasks, but its output scaling and fitness calculation depend on whether the problem is regression or classification.

Regression Fitness

For regression, FastSymbolicGP can fit a linear scale and intercept to the raw symbolic output:

$$\hat{y}_i = a_j f_j(\mathbf{x}_i) + b_j. \quad (4)$$

The coefficients a_j and b_j are obtained analytically from the covariance between the symbolic output and the target. This scaling allows evolution to focus primarily on the nonlinear expression structure while the final output magnitude and offset are adjusted efficiently.

The candidate can be evaluated using mean squared error, root mean squared error, mean absolute error, or an R^2 -based loss. For example, the root mean squared error is

$$L_{\text{RMSE}}(f_j) = \sqrt{\frac{1}{n} \sum_{i=1}^n [y_i - (a_j f_j(\mathbf{x}_i) + b_j)]^2}. \quad (5)$$

Sample weights can be supplied when observations have unequal importance.

Binary Classification Fitness

For binary classification, the symbolic expression produces a continuous decision score. A logistic scale and intercept transform this score into a probability:

$$p_j(\mathbf{x}) = \sigma(a_j f_j(\mathbf{x}) + b_j) = \frac{1}{1 + \exp[-(a_j f_j(\mathbf{x}) + b_j)]}. \quad (6)$$

The coefficients a_j and b_j are estimated for each candidate using a regularized logistic scaling procedure. This is important because the same useful symbolic ordering may appear at different scales or with an inverted sign. Logistic scaling prevents the evolutionary search from unnecessarily spending generations reproducing a simple output rescaling that can be estimated directly.

The default classification objective is logarithmic loss:

$$L_{\log} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]. \quad (7)$$

Other available objectives include accuracy, balanced accuracy, F1-score, Matthews correlation coefficient, ROC-AUC, and average precision. Metrics that are conventionally maximized are internally converted into minimization objectives.

Training and Validation Fitness

When a validation fraction is specified, the available fitting data are separated into an evolutionary training subset and an internal validation subset. Classification splits are stratified so that both subsets preserve the class distribution. Each candidate stores both training and validation fitness.

The selection base can be defined as

$$L_{\text{base}}(f) = \begin{cases} L_{\text{train}}(f), & \text{training selection,} \\ L_{\text{val}}(f), & \text{validation selection,} \\ L_{\text{val}}(f) + \gamma |L_{\text{val}}(f) - L_{\text{train}}(f)|, & \text{combined selection.} \end{cases} \quad (8)$$

where γ controls the penalty assigned to disagreement between training and validation performance. The combined form discourages expressions that obtain a strong training result but deteriorate substantially on validation observations.

The internal validation subset is used for evolutionary model selection and optional probability calibration. It must remain separate from the external test data used for final performance reporting.

2.1.7. Handling Threshold Selection, Class Imbalance, and Multiclass Decisions

Symbolic classification introduces additional decisions that are absent from ordinary symbolic regression. FastSymbolicGP explicitly addresses the principal practical issues of class imbalance, probability scaling, threshold selection, and multiclass output construction.

Class Imbalance

Binary class weights can be omitted, supplied manually, or calculated using the balanced weighting rule. For K classes, the balanced weight assigned to class c is

$$w_c = \frac{n}{Kn_c}, \quad (9)$$

where n is the total number of observations and n_c is the number of observations belonging to class c . These weights are incorporated into logistic scaling and fitness evaluation. Therefore, observations from a minority class contribute more strongly to the evolutionary objective when balanced weighting is enabled.

Class weighting does not itself guarantee optimal minority-class performance. It provides a controlled mechanism for changing the relative cost of errors and should be combined with suitable evaluation measures such as balanced accuracy, macro-F1, Matthews correlation coefficient, ROC-AUC, or average precision.

Probability Calibration

The logistic transformation of a symbolic score provides a probability-like output, but it does not guarantee that the probabilities are well calibrated. The library can therefore apply an additional calibration stage using an internal validation subset.

Available calibration mappings include no calibration, Platt scaling, isotonic calibration, and beta calibration. In automatic mode, candidate calibrators are compared using stratified cross-validated logarithmic loss. A more complex calibrator is retained only when it provides a sufficient improvement over the identity mapping. The calibration report records the requested method, selected method, number of folds, and the validation statistics of the candidate mappings.

Decision-Threshold Selection

The final binary prediction is

$$\hat{y}_i = \begin{cases} 1, & p_i \geq \tau, \\ 0, & p_i < \tau, \end{cases} \quad (10)$$

where τ is the classification threshold. A fixed threshold of 0.5 is available, but this value is not necessarily optimal for imbalanced data or asymmetric classification objectives.

FastSymbolicGP can determine τ from validation probabilities by sorting the unique predicted probabilities and evaluating candidate thresholds between adjacent values. The selected threshold maximizes a specified validation measure, including Matthews correlation coefficient, balanced accuracy, F1-score, or accuracy. When two thresholds provide the same objective value, the value closer to 0.5 is preferred. Threshold optimization is therefore separated from test-set evaluation and does not use the final test labels.

The threshold can also be refitted on a separate calibration dataset using the `refit_threshold` method. This is useful when a deployment environment has a class distribution or error-cost structure that differs from the original training data.

Multiclass Classification

FastSymbolicGP implements two multiclass strategies.

The first strategy is one-vs-rest classification. For K classes, the library evolves K binary symbolic classifiers. Classifier k is trained to distinguish class k from all remaining classes. Let $p_k(\mathbf{x})$ denote its positive-class probability. The multiclass probabilities are normalized as

$$P(y = k \mid \mathbf{x}) \frac{p_k(\mathbf{x})}{\sum_{\ell=1}^K p_{\ell}(\mathbf{x})}. \quad (11)$$

The predicted class is the class with the highest normalized probability. This strategy produces one inspectable symbolic equation per class and is therefore straightforward to analyze.

The second strategy constructs a shared symbolic feature backbone. Candidate symbolic programs are collected from the class-specific hall-of-fame and Pareto populations. Constant-output candidates are removed, and highly correlated candidates are filtered using a configurable correlation limit. The selected symbolic components form

$$\mathbf{z}(\mathbf{x})[f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})]^{\top}. \quad (12)$$

After standardization, a multinomial softmax head maps the symbolic feature vector to class probabilities:

$$P(y = k \mid \mathbf{x}) \frac{\exp(\boldsymbol{\beta}k^{\top} \mathbf{z}(\mathbf{x}) + b_k)}{\sum_{\ell=1}^K \exp(\boldsymbol{\beta}\ell^{\top} \mathbf{z}(\mathbf{x}) + b_{\ell})}. \quad (13)$$

This strategy shares symbolic components among classes and can reduce redundant class-specific expressions. Its output remains inspectable because both the symbolic backbone and the linear softmax coefficients are stored. However, it should be described as a hybrid symbolic–linear multiclass model rather than as a single symbolic equation.

Optional temperature scaling can be applied to multiclass probabilities. The identity mapping is retained unless temperature scaling provides a meaningful improvement in reference logarithmic loss.

2.1.8. Parent Selection and Population Ranking

FastSymbolicGP supports scalar validation-aware ranking and multiobjective Pareto ranking.

Under scalar optimization, the selection value of candidate f is

$$J(f)L_{\text{base}}(f) + \lambda C(f), \quad (14)$$

where L_{base} is the selected training, validation, or combined loss; $C(f)$ is the tree or directed-acyclic-graph complexity; and λ is the current parsimony coefficient. Candidates are sorted first by $J(f)$, then by complexity, and finally by depth. The secondary complexity and depth comparisons provide deterministic preferences when predictive fitness values are equal.

Under multiobjective Pareto optimization, candidates are evaluated using three objectives:

$$(L_{\text{base}}(f), C(f), D(f)), \quad (15)$$

where $D(f)$ is tree depth. A candidate dominates another candidate when it is no worse in all three objectives and strictly better in at least one. The implementation assigns nondominated ranks and crowding distances. Candidates with lower Pareto rank are

preferred, while larger crowding distance is preferred within the same rank to maintain diversity along the loss–complexity frontier.

Parents are selected through tournament selection. For each parent, a configurable number of population indices is sampled with replacement. Under scalar optimization, the candidate Under Pareto or Non-dominated Sorting Genetic Algorithm II (NSGA-II) optimization with the smallest selection value, complexity, and depth wins the tournament. Under NSGA-II optimization, candidates are compared using Pareto rank, crowding distance, selection loss, and complexity. Tournament size therefore controls selection pressure: larger tournaments more strongly favor highly ranked candidates, while smaller tournaments preserve more stochastic diversity.

2.1.9. Crossover and Mutation Operators

Four genetic operators are implemented: subtree crossover, subtree mutation, hoist mutation, and point mutation. Their probabilities are normalized so that one operator is selected for each new offspring.

Subtree Crossover

Subtree crossover selects two parents by tournament selection. A node position is sampled uniformly from the first parent, and a donor subtree is sampled uniformly from the second parent. The selected subtree in the first parent is replaced by a clone of the donor subtree:

$$f_{\text{child}} f_{\text{parent1}} [s_{\text{parent1}} \leftarrow s_{\text{parent2}}]. \quad (16)$$

Because every node is eligible for selection, crossover can replace a small terminal, an intermediate expression, or the complete root expression.

Subtree Mutation

Subtree mutation selects one node position from the parent and replaces it with a newly generated grow tree. The replacement tree is generated from the current primitive set, feature set, constant interval, constant probability, and mutation-depth interval. This operation can introduce both new mathematical structure and new terminal combinations.

Hoist Mutation

Hoist mutation first selects a subtree from the parent. It then selects one node from within that subtree and replaces the original subtree with the selected descendant. Hoist mutation generally reduces or restructures expression size and is particularly useful for counteracting genetic programming bloat.

Point Mutation

Point mutation selects one node and modifies it without replacing the complete surrounding subtree:

- A feature terminal is replaced by another randomly selected feature;
- A numerical constant is perturbed using Gaussian noise and clipped to the permitted constant interval;
- A function node is replaced by another primitive with the same arity.

Restricting function replacement to primitives with equal arity preserves the validity of the existing child structure.

The default operator distribution is

$$p_{\text{cross}} = 0.70, \quad p_{\text{subtree}} = 0.15, \quad p_{\text{hoist}} = 0.05, \quad p_{\text{point}} = 0.10. \quad (17)$$

The relatively high crossover probability emphasizes recombination of useful subexpressions. Subtree mutation provides larger structural innovations, point mutation provides local modifications, and hoist mutation supplies an explicit size-reduction mechanism. These probabilities are configurable and may optionally be adapted using the observed operators associated with successful population members.

After variation, every offspring is checked against the maximum-depth and maximum-node limits. An oversized offspring can be rejected and replaced by its parent. Alternatively, simplification can be attempted before rejection. These checks ensure that genetic operations cannot produce unbounded expression growth.

2.1.10. Elitism, Duplicate Elimination, and Diversity

A configurable number of top-ranked candidates is copied unchanged into the next generation. This elitism mechanism prevents the best currently known expressions from being lost through stochastic variation.

The remaining population positions are filled through tournament selection and genetic variation. Two complementary duplicate-detection mechanisms are available.

Structural duplicate elimination compares the serialized program structure. Expressions with identical nodes, constants, and feature indices have the same structural representation and can be rejected.

Semantic duplicate elimination evaluates candidates on a fixed subset of training observations, rounds their outputs, and hashes the resulting prediction vector. Expressions with different trees but effectively identical sampled behavior can therefore be recognized as duplicates. This is important because algebraically redundant expressions can otherwise occupy a large proportion of the population.

The algorithm allows a limited number of duplicate-rejection attempts. After this limit is reached, an offspring can be accepted to guarantee that population construction terminates. Population diversity is recorded in the evolutionary history as the fraction of structurally distinct programs.

For larger searches, the library also supports an island population model. The complete population is divided into semi-independent islands, which may use different search profiles. At configured intervals, selected individuals migrate between islands using ring, random, best-to-worst, or diversity-guided migration. Island evolution can preserve alternative search trajectories and reduce premature convergence, particularly when islands emphasize different objectives such as accuracy, compactness, diversity, or robustness.

2.1.11. Bloat Control and Complexity-Aware Evolution

Uncontrolled growth of program trees is a well-known difficulty in genetic programming [18,25,26]. FastSymbolicGP combines several complementary bloat-control mechanisms:

- Hard maximum-depth and maximum-node constraints;
- Hoist mutation;
- Scalar parsimony penalties;
- Adaptive parsimony;
- NSGA-II loss–complexity–depth optimization;
- Final selection of the smallest model within a fitness tolerance;
- Optional Tarpeian penalization of randomly selected above-median-size programs;
- Algebraic simplification;
- Rejection of structural and semantic duplicates.

Under adaptive parsimony, the algorithm monitors the mean population size. When the mean size exceeds a user-defined target, the parsimony coefficient is increased in proportion to the excess and to the typical magnitude of the population loss. When the population

returns below the target, the coefficient gradually decreases. This avoids requiring the same fixed parsimony coefficient for datasets whose loss scales differ substantially.

The Tarpeian option probabilistically assigns poor fitness to some programs larger than the median population size before performing their complete evaluation. This can reduce unnecessary computation, although it is disabled by default because it introduces an additional stochastic approximation.

Algebraic simplification is applied safely using deterministic rewrite rules and constant folding. Examples include

$$x + 0 \rightarrow x, \quad x - 0 \rightarrow x, \quad x - x \rightarrow 0, \quad (18)$$

$$x \times 1 \rightarrow x, \quad x \times 0 \rightarrow 0, \quad \frac{x}{1} \rightarrow x. \quad (19)$$

Repeated negation and repeated absolute-value operations are also simplified. The raw selected expression and simplified expression remain separately accessible so that the effect of simplification can be examined.

2.1.12. Hall of Fame, Pareto Pool, and Final Model Selection

The algorithm does not select the final output solely from the last generation. A hall of fame retains the best structurally unique candidates observed during the entire run. When the same structure occurs more than once, the copy with the better selection fitness is retained.

From the hall of fame, the library constructs a Pareto pool containing candidates that are not dominated simultaneously in loss, complexity, and depth. This pool is important because symbolic modeling rarely has a single universally best equation. A slightly less accurate but substantially smaller expression may be preferable for scientific interpretation, deployment, or manual verification [17,19].

Four final-selection modes are available:

1. `raw`: selects the candidate with the best training fitness;
2. `validation`: selects the candidate with the best validation fitness;
3. `pareto`: selects from the nondominated population;
4. `smallest_within_tolerance`: identifies all candidates whose selection loss is within a specified tolerance of the best loss and then selects the candidate with the smallest complexity.

The last mode is the default because small numerical differences in validation loss often do not justify a much larger equation. Let $J_{\min}$ be the best selection value and let ϵ be the configured tolerance. The eligible set is

$$\mathcal{E} = \{f \mid J(f) \leq J_{\min} + \epsilon\}. \quad (20)$$

The final program is then

$$f^* \arg \min_{f \in \mathcal{E}} (C(f), J(f), D(f)). \quad (21)$$

Users can also select a different Pareto candidate after training by imposing constraints on maximum nodes, maximum depth, or maximum acceptable loss. This is particularly useful when scientific considerations require a simpler equation than the automatically selected model.

2.1.13. Expression Pool and Interpretability Outputs

The library exposes substantially more information than a single predictive score. The principal expression-related outputs include:

- The selected expression;
- The unsimplified selected expression;
- The simplified expression;
- The complete hall of fame;
- The nondominated Pareto pool;
- Symbolic ensemble members and their weights;
- One expression per class for one-vs-rest multiclass models;
- Symbolic backbone equations and softmax coefficients for shared multiclass models;
- Node count, depth, simplified node count, and simplified depth;
- Tree and directed-acyclic-graph operation counts;
- Repeated-subexpression count and compression ratio;
- Feature occurrence frequencies;
- Primitive-function occurrence frequencies.

The expression statistics can be retrieved programmatically as shown in Listing 2.

Listing 2. Extraction of the selected expression and complexity measures.

```

1  expression = model.get_expression()
2  raw_expression = model.get_raw_expression()
3  simplified_expression = model.get_simplified_expression()
4  statistics = model.get_expression_stats()
5
6  print(expression)
7  print(statistics["nodes"])
8  print(statistics["depth"])
9  print(statistics["dag_nodes"])
```

The Pareto pool can be used to report an equation set rather than only one selected equation. For each candidate, the loss, number of tree nodes, number of directed-acyclic-graph nodes, depth, and serialized expression can be saved. This allows direct comparison with equation pools returned by other symbolic regression tools.

Cross-run interpretability can also be evaluated. Feature stability reports the fraction of independently fitted models in which a feature appears. Subexpression stability reports how frequently the same symbolic subexpression occurs across repeated runs. These measures distinguish a single compact equation from a relationship that is repeatedly rediscovered under different random seeds.

For classification, optional local feature-occlusion explanations quantify the change in predicted probability when one feature is replaced by a baseline value. A greedy counterfactual search is also available for exploratory use. These local procedures supplement, but do not replace, interpretation of the global symbolic equation.

2.1.14. Symbolic Ensembles

Binary classification can use either the single selected expression or a symbolic probability ensemble. Ensemble candidates are drawn from the hall of fame and ordered using loss, directed-acyclic-graph complexity, and depth. Candidates with nearly constant outputs are removed. Highly correlated probability outputs can also be excluded to preserve functional diversity.

For an ensemble containing M expressions, the positive-class probability is

$$p_{\text{ens}}(\mathbf{x}) = \sum_{m=1}^M \omega_m p_m(\mathbf{x}), \quad (22)$$

subject to

$$\omega_m \geq 0, \quad \sum_{m=1}^M \omega_m = 1. \quad (23)$$

Weights may be uniform, derived from validation loss, or optimized under the non-negative simplex constraint. Every ensemble member, weight, expression size, depth, and equation is retained in the model report.

An ensemble can improve predictive robustness, but it is less compact than a single equation. Therefore, the ensemble is an optional prediction mode and is reported separately from the primary selected expression.

2.1.15. Efficient Expression Evaluation

Fitness evaluation is normally the dominant computational cost of genetic programming because every candidate must be evaluated over many observations and generations. FastSymbolicGP uses several mechanisms to reduce this cost.

First, expression evaluation is vectorized. A primitive receives complete NumPy arrays and produces an output for all observations at once. This avoids a Python loop over individual samples.

Second, each tree can be compiled into postfix instructions. Feature terminals, constants, and primitive functions are represented by numerical operation codes. The postfix representation can be executed by a vectorized NumPy interpreter or by an optional compiled Numba backend.

Third, a symbolic tree can be compiled into a directed acyclic graph. Structurally identical subtrees are represented once, reducing the effective operation count. If the expression

$$f(\mathbf{x}) = h(\mathbf{x}) + h(\mathbf{x}) \quad (24)$$

contains two identical copies of h , the directed-acyclic-graph evaluator calculates h once and reuses the result.

Fourth, a bounded least-recently-used subtree cache can retain the numerical outputs of previously evaluated subexpressions. Cache entries are associated with both the subexpression structure and a fingerprint of the evaluated data matrix, preventing accidental reuse for a different sample matrix. The cache has a hard memory limit and evicts old entries when the limit is reached.

Fifth, population evaluation can be distributed across worker threads. Thread limits are controlled to avoid nested parallelism in underlying numerical libraries. In island mode, islands can also be evaluated in parallel.

Sixth, the `max_samples` parameter can evaluate each generation on a random subset of the evolutionary training data. This is especially useful for large datasets. The validation objective can still be calculated on the fixed validation subset, allowing computationally cheaper training evaluations while preserving a common comparison set.

Finally, time, evaluation-count, and memory budgets can be imposed. The stopping reason records whether evolution ended because of the maximum number of generations, early stopping, time budget, or evaluation budget.

2.1.16. Missing Values and Robustness Objectives

Missing inputs can be rejected, replaced by a constant, replaced by training-set medians, or handled through native symbolic operators. In native mode, missing feature terminals are preserved long enough for the `is_missing` and `coalesce` primitives to inspect them. Function outputs are subsequently converted to finite values.

Optional robustness training evaluates expressions on perturbed versions of the training data. Perturbations can include Gaussian feature noise, feature dropout, or their combination. When enabled, the ordinary loss and perturbed-data loss are combined using a configurable robustness weight. This option is intended for deployment-hardening experiments and should be reported explicitly whenever it is used because it changes the fitness objective and increases the number of expression evaluations.

2.1.17. Hyperparameter Rationale

genetic programming parameters do not have universally optimal values. Their role is to define a search budget and the balance between exploration, exploitation, runtime, and interpretability. The principal defaults and their rationale are summarized in Table 4. These values are library defaults rather than claims of globally optimal settings. Experimental studies should report every overridden value and should compare several search budgets when computationally feasible.

Table 4. Principal evolutionary defaults and their design rationale.

Parameter	Default	Rationale
Population size	500	Provides broad structural diversity for a general-purpose run. Smaller values reduce runtime but increase the risk of premature convergence.
Generations	50	Limits the default computational budget while allowing repeated recombination and mutation. Early stopping can terminate the run sooner.
Tournament size	20	Provides moderate-to-strong selection pressure in the general default configuration. It should be reduced with very small populations.
Initialization depth	2–6	Generates both relatively compact and moderately nonlinear initial structures.
Maximum depth	12	Allows nested nonlinear relationships while preventing uncontrolled tree growth.
Maximum nodes	255	Provides a hard upper bound on memory use, evaluation cost, and expression size.
Crossover probability	0.70	Makes recombination of existing useful subexpressions the primary search operator.
Subtree mutation probability	0.15	Introduces new branches and larger structural changes.
Hoist mutation probability	0.05	Provides an explicit contraction operator for bloat control.
Point mutation probability	0.10	Provides local changes to variables, constants, and same-arity functions.
Validation fraction	0.20	Reserves enough observations for validation-aware selection while retaining most observations for evolutionary fitting.
Elites	2	Preserves the strongest candidates without allowing elites to dominate a large proportion of the next generation.
Patience	20 generations	Stops evolution after sustained stagnation while tolerating temporary periods without improvement.
Selection tolerance	0.02	Allows a smaller expression to be preferred when its selection loss is very close to the best observed loss.
Parsimony target	40 nodes	Defines the reference population size above which adaptive complexity pressure begins to increase.

The main computational budget is approximately proportional to

$$B \propto P, G, N_{\text{eval}}, \quad (25)$$

where P is population size, G is the number of generations, and N_{eval} is the number of observations used for each fitness evaluation. Expression complexity, primitive cost, multiclass strategy, validation evaluation, calibration, robust perturbations, and ensemble construction add further computational cost.

For this reason, a larger population is not automatically superior to more generations. A larger population explores more structures per generation, whereas more generations

provide a longer sequence of selection and refinement. Search profiles should therefore be treated as controlled computational budgets.

The library includes convenience presets that change several parameters together, as summarized in Table 5.

Table 5. Convenience search presets included in FastSymbolicGP. These presets are distinct from the experimental benchmark profiles defined in Section 2.2.

Preset	Population	Generations	Max. Depth	Max. Nodes	Purpose
Fast	32	60	10	127	Short exploratory runs and pipeline verification
Balanced	64	120	14	255	General compromise between search quality and runtime
Accurate	96	200	16	383	Larger search budget for predictive performance
Interpretable	64	140	9	95	Stronger preference for compact single expressions
Large dataset	48	100	10	127	Reduced per-generation data fraction and bounded complexity
Mobile	40	100	8	63	Compact expressions and restricted primitives for deployment

The automatic preset complements the predefined configurations reported in Table 5 by using dataset shape as a coarse resource heuristic. It does not replace dataset-specific validation or a sensitivity analysis. For publication experiments, the selected population size, generation count, function set, depth limits, parsimony configuration, random seeds, and stopping criteria should be reported explicitly.

2.1.18. Dimensional Analysis and Physical Consistency

The current library does not automatically attach physical units to variables or enforce dimensional homogeneity during crossover, mutation, or final selection. Consequently, if dimensional input variables are supplied directly, the unconstrained search can construct expressions that add incompatible physical quantities or apply logarithmic and trigonometric functions to dimensional arguments.

For example, an unconstrained program could construct

$$x_{\text{length}} + x_{\text{time}}, \quad (26)$$

which is numerically executable but physically meaningless. Similarly,

$$\log(x_{\text{pressure}}) \quad (27)$$

is not dimensionally well defined unless the pressure is first divided by a reference pressure.

For scientific applications, one of the following procedures should therefore be applied:

1. Transform all inputs into dimensionless quantities before fitting;
2. Normalize dimensional variables by physically meaningful reference values;
3. Construct unit-consistent derived features before symbolic search;
4. Restrict the primitive set so that invalid transformations cannot occur;
5. Manually inspect the units of every Pareto-selected expression;
6. Reject equations that violate conservation laws, boundary conditions, monotonicity, symmetry, or other known physical constraints.

Physics-aware symbolic regression can incorporate unit constraints directly into the search [15,16]. Such automatic dimensional typing is not presently part of FastSymbolicGP and should be regarded as an important direction for future development rather than an existing capability.

2.1.19. Physical Interpretation and Scientific Model Selection

The automatic final-selection procedure is based on empirical loss, validation behavior, expression complexity, and depth. It does not independently establish that the selected equation represents a valid physical law. Statistical agreement with observed data is necessary but not sufficient for physical justification.

FastSymbolicGP therefore separates computational selection from domain validation. The library returns the selected model and a pool of alternative Pareto-optimal expressions. A domain expert can then compare these candidates according to additional criteria, including:

- Dimensional homogeneity;
- Signs and magnitudes of coefficients;
- Limiting behavior;
- Known boundary conditions;
- Monotonicity;
- Symmetry and invariance;
- Conservation principles;
- Consistency with established theoretical models;
- Numerical stability outside the training range;
- Reproducibility across random seeds or experimental datasets.

A scientifically preferred equation may therefore differ from the expression having the numerically smallest validation error. The `select_from_pareto` functionality allows the user to impose maximum size, maximum depth, or maximum loss constraints and then select a suitable candidate. However, the physical argument for choosing one equation over another must be supplied by the researcher and supported by domain-specific analysis.

2.1.20. Reproducibility, Checkpointing, and Reporting

Every stochastic operation is controlled through a NumPy random generator initialized from the estimator's `random_state`. This includes tree generation, tournament sampling, genetic operators, semantic-sample selection, data subsampling, validation splitting, and island initialization.

During fitting, the evolutionary history records, for each generation:

- Best training, validation, and selection fitness;
- Mean and median population fitness;
- Node count, directed-acyclic-graph node count, and depth of the best candidate;
- Mean population size;
- Structural diversity;
- Pareto-pool size;
- Cache hits and cache hit rate;
- Number of training samples used;
- Current population size;
- Total evaluation count;
- Current parsimony coefficient;
- Effective operator probabilities;
- Generation time and cumulative elapsed time;
- Stagnation counter;

- Execution backend and population model.

Models can be saved using joblib serialization. Evolutionary histories can be exported as comma-separated values (CSV) files, while complete reports can be exported as JavaScript Object Notation (JSON) files. Reports include parameters, stopping reason, expression, complexity statistics, feature use, function use, cache statistics, Pareto candidates, environment information, and per-generation history.

Checkpoint files can retain the population, hall of fame, random-generator state, parsimony coefficient, adaptive operator probabilities, primitive weights, evaluation count, and generation number. Interrupted searches can therefore be resumed. A fitted model can also be branched into an independent warm-start experiment with modified parsimony, depth, generation, or other settings.

2.1.21. Deployment and Model Portability

Because the final model is an explicit expression, it can be evaluated independently of the evolutionary process. The library supports model serialization and source-code export. Binary classifiers and regressors can be exported to portable implementations in Python, C, C++, Java, Kotlin, and JavaScript. Python export also supports the implemented multiclass structures.

Inference profiling reports batch latency, latency per sample, peak working memory, tree-operation count, directed-acyclic-graph operation count, compression ratio, and execution engine. These measures complement training-time results because two equations with similar predictive performance can have substantially different deployment costs.

2.1.22. Summary of the Evolutionary Procedure

The complete fitting procedure can be summarized as follows:

1. Validate the input matrix, target vector, task-specific parameters, primitive set, and resource limits.
2. Apply optional preprocessing for missing values and create the evolutionary training and internal validation subsets.
3. Generate a structurally diverse initial population using full and grow initialization.
4. Evaluate every candidate expression on the training data and, when enabled, on the validation data.
5. Fit task-specific output scaling: linear scaling for regression or logistic scaling for binary classification.
6. Calculate training fitness, validation fitness, complexity, and the final selection objective.
7. Rank candidates using scalar validation-aware ranking or NSGA-II nondominated sorting and crowding distance.
8. Update the hall of fame and Pareto pool.
9. Copy elite programs into the next generation.
10. Generate the remaining offspring through tournament selection, subtree crossover, subtree mutation, hoist mutation, or point mutation.
11. Reject or simplify expressions that violate the maximum-depth or maximum-node constraints.
12. Remove structural and semantic duplicates where possible.
13. Update adaptive parsimony, operator probabilities, primitive weights, island migrations, or population size when the corresponding options are enabled.
14. Stop when the maximum number of generations, stagnation patience, time budget, or evaluation budget is reached.

15. Select the final model from the complete hall of fame, preferably using validation performance and expression complexity jointly.
16. Simplify the selected expression and fit its final regression scaling or classification probability mapping.
17. For classification, perform optional probability calibration, decision-threshold optimization, ensemble construction, or multiclass probability construction.
18. Store the selected equation, alternative Pareto equations, complexity indicators, prediction configuration, evolutionary history, and reproducibility metadata.

This design makes the distinction between symbolic search, numerical output scaling, classification decision construction, and scientific interpretation explicit. FastSymbolicGP searches for compact symbolic structures and provides tools for validation-aware statistical selection. Dimensional validity, physical plausibility, and preference among scientifically competing equations remain domain-dependent decisions that must be justified through the experimental analysis presented with each application.

2.2. Experimental Methodology

This subsection describes the methodology used to evaluate FastSymbolicGP as a symbolic machine learning library for regression, binary classification, multiclass classification, physical law recovery, and dynamical system identification. The revised benchmark was designed not only to measure predictive performance and computational time, but also to examine the properties that motivate the use of symbolic models in scientific applications: explicit mathematical output, expression complexity, equation diversity, dimensional consistency, and the trade-off between predictive accuracy and interpretability.

The experimental investigation addresses the following questions:

1. Can FastSymbolicGP obtain competitive predictive performance on real scientific datasets from different application domains?
2. How does its computational cost compare with established symbolic regression and genetic programming implementations?
3. Does the library produce compact and inspectable symbolic expressions rather than only predictive metrics?
4. How do threshold selection, class weighting, and multiclass decision strategies affect symbolic classification?
5. Can the generated expressions recover known physical relationships and governing equations?
6. How sensitive is the method to population size and number of generations?
7. How do individual evolutionary mechanisms affect predictive performance, runtime, and expression complexity?
8. How does the implementation scale with increasing sample size and input dimensionality?

The complete benchmark contains real scientific regression experiments, known physical law recovery problems, binary and multiclass classification experiments, governing-equation discovery, parameter sensitivity analysis, component ablation, scalability analysis, equation-pool extraction, interpretability analysis, and paired statistical testing. All experiment definitions, random seeds, model settings, evaluation metrics, and output-generation procedures were specified programmatically before the final result analysis.

The experimental procedure was organized as a sequential and reproducible workflow covering dataset preparation, model configuration, symbolic model training, predictive evaluation, equation extraction, and final result analysis. An overview of the complete benchmark pipeline is presented in Figure 3.

FastSymbolicGP Benchmark Workflow

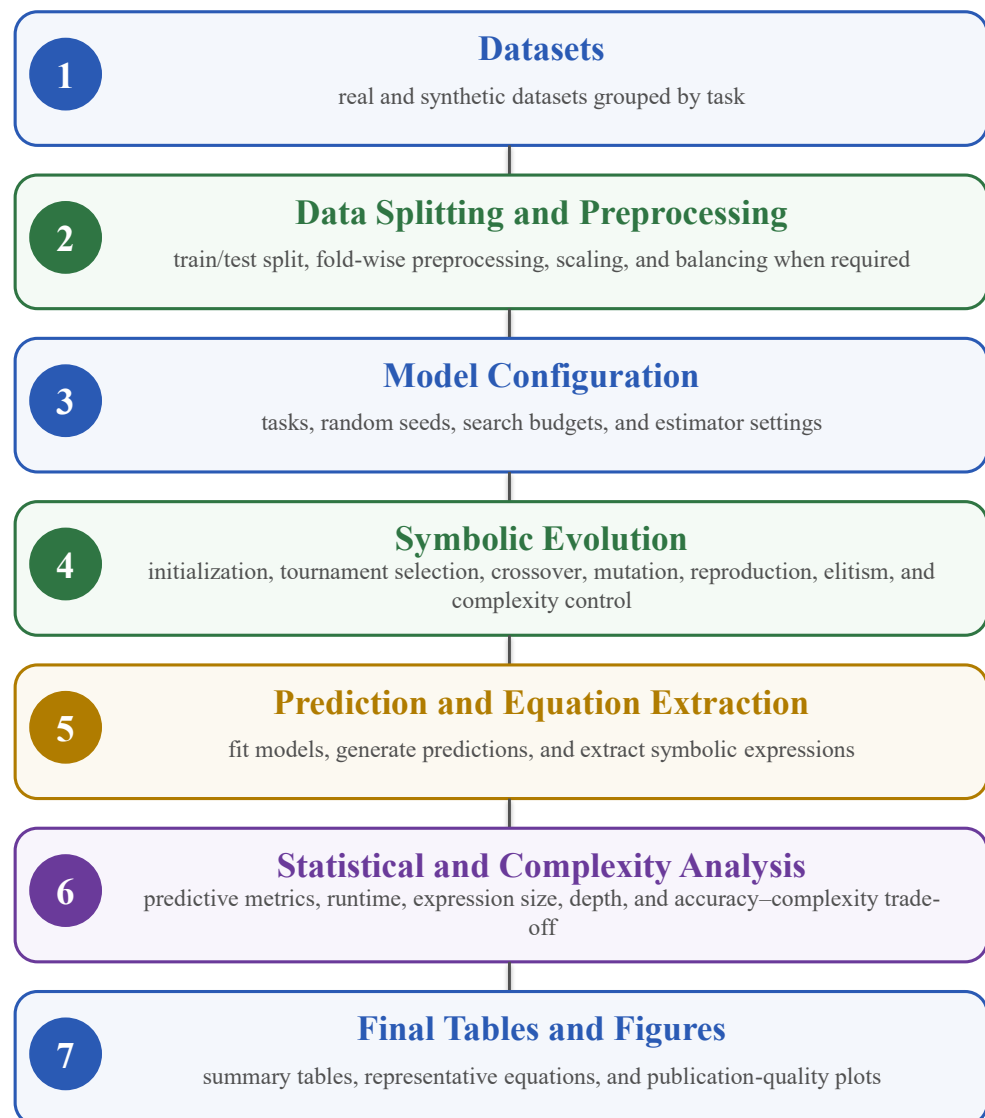


Figure 3. Experimental workflow used for evaluating FastSymbolicGP. The workflow consists of seven successive stages: dataset selection, data splitting and preprocessing, task-specific model configuration, symbolic evolution, prediction and equation extraction, statistical and complexity analysis, and generation of the final tables and figures.

As illustrated in Figure 3, the evaluation begins by organizing the real-world and synthetic datasets according to the regression, binary classification, and multiclass classification tasks. Each dataset is subsequently divided into training and test subsets, while preprocessing, feature scaling, and class-balancing procedures are applied within the appropriate training partitions to avoid information leakage. The experimental configuration then defines the task type, random seed, search budget, and estimator-specific settings used in each run.

The configured models proceed through the symbolic-evolution stage, which includes population initialization, tournament selection, crossover, mutation, reproduction, elitism, and complexity control. After evolution, the fitted models are used to generate predictions and extract the corresponding symbolic expressions. Predictive performance, computational runtime, expression size, expression depth, and the resulting accuracy–complexity trade-off are then analyzed. Finally, the collected results are aggregated into summary

tables, representative-equation comparisons, and publication-quality figures, as indicated in the final stage of Figure 3.

2.2.1. Symbolic Representation and Primitive Set

A candidate FastSymbolicGP model is represented as a symbolic expression tree. Internal nodes contain mathematical functions, whereas terminal nodes contain input variables or ephemeral numerical constants. For the principal experiments, the primitive set was

$$\mathcal{F} = +, -, \times, \div, \sqrt{\cdot}, \log(\cdot), |\cdot|, \sin(\cdot), \cos(\cdot). \quad (28)$$

The corresponding implementation names were `add`, `sub`, `mul`, `div`, `sqrt`, `log`, `abs`, `sin`, and `cos`. Protected implementations supplied by the library were used for potentially unsafe operations such as division, logarithm, and square root so that candidate expressions could be evaluated without interrupting the evolutionary process because of invalid floating-point values.

Initial expressions were generated with depths between two and five. Numerical constants were sampled from the interval $[-2, 2]$, and a constant-generation probability of 0.15 was used. The maximum expression depth in the principal paper profile was six, while the maximum number of tree nodes was 63. Candidate offspring exceeding these structural restrictions were rejected. These restrictions were introduced to reduce uncontrolled expression growth and to ensure that the resulting equations remained suitable for direct inspection.

2.2.2. Fitness Evaluation

For regression, candidate expressions were evaluated using root mean squared error:

$$\text{RMSE}(f) = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - f(\mathbf{x}_i))^2}. \quad (29)$$

After symbolic evolution, FastSymbolicGP may estimate an affine output transformation. Therefore, the complete regression model is reported as

$$\hat{y}(\mathbf{x}) = af(\mathbf{x}) + b, \quad (30)$$

where $f(\mathbf{x})$ is the evolved symbolic expression, and a and b are the fitted output scale and intercept. Both the raw expression f and the complete prediction equation were stored.

For binary classification, the evolved expression produces a real-valued symbolic score. The complete probabilistic model is

$$s(\mathbf{x}) = af(\mathbf{x}) + b, \quad (31)$$

$$p(y = 1 | \mathbf{x}) = \sigma(s(\mathbf{x})) = \frac{1}{1 + \exp[-s(\mathbf{x})]}. \quad (32)$$

Candidate classifiers were evaluated using binary logarithmic loss:

$$L_{\log} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]. \quad (33)$$

The classifier used the library's elite-logistic fitness-scaling mechanism to estimate the affine transformation applied before the sigmoid link. Additional probability calibration was disabled in the benchmark. Consequently, the relationship between the evolved ex-

pression, sigmoid transformation, and decision threshold remained explicit in the reported classification equation.

2.2.3. Training, Validation, and Selection Loss

FastSymbolicGP supports training-based, validation-based, and combined model-selection criteria. The corresponding base loss can be written as

$$L_{\text{base}}(f) = \begin{cases} L_{\text{train}}(f), & \text{training selection,} \\ L_{\text{val}}(f), & \text{validation selection,} \\ L_{\text{val}}(f) + \gamma |L_{\text{val}}(f) - L_{\text{train}}(f)|, & \text{combined selection,} \end{cases} \quad (34)$$

where γ controls the penalty assigned to disagreement between training and validation performance. The benchmark configuration used $\gamma = 0.25$ when the combined criterion was applicable.

Validation-based selection was used in the principal regression, physical law, sensitivity, scalability, and multiclass experiments. FastSymbolicGP internally reserved 20% of the supplied development partition for validation. Binary threshold experiments were treated differently: internal validation was disabled, the symbolic structure was selected using the training data, and a separate external validation partition was reserved exclusively for threshold optimization. This separation prevented test-set information from influencing either symbolic evolution or threshold selection.

An adaptive parsimony mechanism was used to discourage uncontrolled expression growth. The selection criterion can be expressed generally as

$$L_{\text{selection}}(f) = L_{\text{base}}(f) + P_g(f), \quad (35)$$

where $P_g(f)$ is the generation-dependent complexity penalty. The adaptive controller used a target expression size of 40 nodes and a growth factor of 1.05. The purpose of this mechanism was not to force all expressions to contain fewer than 40 nodes, but to increase selection pressure against unnecessary growth when the population tended to exceed the target complexity.

The evolutionary search used scalar optimization. At the end of a run, the selected output was the smallest model whose selection loss remained within a tolerance of 2% of the best observed selection loss. Let

$$L_{\min} = \min_{f \in \mathcal{H}} L_{\text{selection}}(f), \quad (36)$$

where $\mathcal{H}$ denotes the stored candidate set. The eligible model set was

$$\mathcal{H}_{0.02} = \{f \in \mathcal{H} \mid L_{\text{selection}}(f) \leq (1 + 0.02)L_{\min}\}. \quad (37)$$

The final expression was selected as the smallest member of $\mathcal{H}_{0.02}$. This criterion directly incorporates interpretability into model selection by preferring a simpler equation when its predictive loss is practically indistinguishable from that of the best candidate.

2.2.4. Tournament Selection and Genetic Operators

Parents were chosen using tournament selection. For each parent selection, k candidate individuals were sampled from the current population and compared according to their selection loss. The candidate with the lowest loss was selected as the parent. A tournament size of $k = 10$ was used in the principal benchmark profile. Tournament

selection was chosen because it provides controllable selection pressure without requiring fitness-proportionate probabilities, which can become unstable when fitness values are closely grouped or have different numerical scales.

Four variation operators were used. Their probabilities summed to one in the principal configuration.

Subtree Crossover

Subtree crossover was applied with probability 0.70. Two parents were selected independently by tournament selection. A crossover point was selected in each parent, defining one subtree in each expression. The selected subtree from the first parent was replaced by the selected subtree from the second parent to form an offspring. The offspring was accepted only when its depth and number of nodes satisfied the configured structural limits. Otherwise, the oversized offspring was rejected.

Subtree crossover exchanges complete functional components between expressions and therefore provides the principal mechanism for combining useful symbolic building blocks discovered in different individuals.

Subtree Mutation

Subtree mutation was applied with probability 0.15. A mutation point was randomly selected in the parent expression, and the corresponding subtree was removed. It was replaced with a newly generated random subtree constructed from the same primitive and terminal sets. The newly generated subtree had a depth between one and four.

This operator can introduce mathematical structures that are absent from the selected parent and therefore provides broader structural exploration than point mutation.

Hoist Mutation

Hoist mutation was applied with probability 0.05. First, an outer subtree was selected from the parent expression. A second, inner subtree was then selected from within the outer subtree. The inner subtree was promoted, or hoisted, to replace the complete outer subtree.

Because the replacement is always contained within the removed structure, hoist mutation commonly reduces expression size or depth. It therefore serves both as a variation mechanism and as an additional form of bloat control.

Point Mutation

Point mutation was applied with probability 0.10. A node was selected from the parent expression. When the selected node was a function, it was replaced by another function with compatible arity. When the selected node was a terminal, it was replaced by another input variable or numerical constant. Point mutation preserves the surrounding tree topology and therefore performs localized refinement of an existing symbolic structure.

Elitism and Duplicate Control

Two elite individuals were transferred directly to the next generation. A hall of fame containing up to 15 candidates was maintained to preserve strong expressions discovered during different generations.

Exact structural duplicate elimination was enabled. Semantic duplicate elimination was disabled in the reported reference configuration. Therefore, the benchmark removed syntactically duplicated expressions but did not reject two structurally different expressions solely because they produced similar numerical outputs.

Operator-probability adaptation and function-set adaptation were disabled so that the same evolutionary probabilities and primitive set were used throughout every principal run.

2.2.5. Termination Conditions

A maximum of 20 generations was specified for the principal paper profile. Early stopping was also enabled. Evolution terminated when the monitored selection criterion failed to improve by at least 10^{-4} for six consecutive generations. The actual number of completed generations and the stopping reason were recorded for every FastSymbolicGP run.

The nominal population–generation product of the principal FastSymbolicGP configuration was

$$250 \times 20 = 5000 \quad (38)$$

candidate positions before accounting for elitism, duplicate rejection, and early termination. This value is reported only as a description of the FastSymbolicGP search budget and is not interpreted as an exact count of unique candidate evaluations.

2.2.6. Numba-Only Expression Evaluation

Every FastSymbolicGP experiment was forced to use the postfix Numba evaluation engine. The relevant settings were

- `evaluation_backend = "numba",`
- `dag_execution = False,`
- `evaluation_cache = False,`
- `subtree_cache = False,`
- `complexity_measure = "tree",`
- `n_jobs = 1,` and
- `thread_limit = 1.`

Before timed experiments, a small postfix expression was evaluated once to trigger Numba compilation. This untimed warm-up prevented just-in-time compilation overhead from being included in the first reported training time.

After every fitted FastSymbolicGP model, the benchmark verified that the fitted backend was numba and that the execution engine was postfix_numba. For multiclass models, every underlying symbolic estimator was checked separately. An experiment was terminated if a model used the NumPy evaluator, DAG execution, or a cached subtree evaluator.

Evaluation caches and DAG execution were intentionally disabled because the objective of this benchmark was to evaluate one clearly specified execution pathway rather than a mixture of automatically selected backends. DAG node counts were still recorded as descriptive structural statistics, but DAG execution was not used during timing.

2.2.7. Principal Experimental Configuration

The principal configuration used for the manuscript experiments is summarized in Table 6.

Table 6. Principal FastSymbolicGP configuration used in the paper experiments.

Parameter	Value
Random seeds	0, 1, 2, 3, and 4
Population size	250
Maximum generations	20
Tournament size	10
Initial expression depth	2–5
Maximum expression depth	6
Maximum tree nodes	63
Mutation subtree depth	1–4
Constant interval	[−2, 2]

Table 6. *Cont.*

Parameter	Value
Constant probability	0.15
Crossover probability	0.70
Subtree-mutation probability	0.15
Hoist-mutation probability	0.05
Point-mutation probability	0.10
Elites	2
Hall-of-fame size	15
Parsimony mechanism	Adaptive
Parsimony target	40 tree nodes
Parsimony growth factor	1.05
Validation fraction	0.20 of supplied development data
Principal selection metric	Validation loss
Final selection	Smallest model within 2% of best loss
Early-stopping patience	6 generations
Minimum improvement	10^{-4}
Optimization mode	Scalar
Exact duplicate elimination	Enabled
Semantic duplicate elimination	Disabled
Automatic simplification	Disabled
Evaluation backend	Postfix Numba
DAG execution and evaluation caches	Disabled
Execution threads	One

The population size and number of generations were not presented as universally optimal values. They define a moderate computational profile selected for the principal comparison. Their adequacy was evaluated separately through a sensitivity experiment spanning substantially smaller and larger search budgets.

2.2.8. Benchmark Datasets and Scientific Domains

The benchmark was designed to cover several scientific and machine learning domains rather than relying only on one or two small illustrative datasets. The principal datasets and controlled problems are summarized in Table 7.

Table 7. Datasets and controlled problems included in the revised benchmark.

Dataset or System	Task	Scientific Field or Purpose
Concrete Compressive Strength	Regression	Civil engineering
Energy Efficiency—Heating Load	Regression	Building physics
Airfoil Self-Noise	Regression	Aeroacoustics
Parkinson’s Total UPDRS	Regression	Biomedicine
Breast Cancer Wisconsin	Binary classification	Biomedicine
Spambase	Binary classification	Cybersecurity and spam detection
Controlled imbalanced datasets	Binary classification	Class-imbalance analysis
Iris	Multiclass classification	Biology
Wine	Multiclass classification	Analytical chemistry
Digits	Multiclass classification	Pattern recognition
Momentum	Physical law recovery	Classical mechanics
Kinetic energy	Physical law recovery	Classical mechanics
Density	Physical law recovery	Mechanics and material properties
Mechanical power	Physical law recovery	Mechanical engineering
Pendulum period	Physical law recovery	Oscillatory mechanics
Lorenz system	Governing-equation discovery	Nonlinear dynamics
Lotka–Volterra system	Governing-equation discovery	Mathematical biology
Friedman 1	Controlled regression	Parameter sensitivity and ablation
Controlled nonlinear regression	Regression	Scalability analysis

The four real scientific regression datasets were obtained from the University of California, Irvine (UCI) Machine Learning Repository. Breast Cancer Wisconsin, Iris, Wine, and Digits were loaded through scikit-learn [5]. Spambase was obtained from the UCI repository. The controlled classification, physical law, sensitivity, ablation, and scalability data were generated directly by the benchmark scripts.

2.2.9. Data Preparation and Experimental Partitioning

All UCI input columns were converted to numerical values. Rows containing missing or non-convertible values were removed. Dataset feature names were sanitized only to make them valid symbolic variable identifiers; a mapping to the original feature names was stored with the results. Subject-identification columns were removed from the Parkinson's dataset because they identify individuals rather than representing general predictive measurements.

For real regression, physical law recovery, sensitivity, ablation, and scalability experiments, the data were first divided into an 80% development partition and a 20% independent test partition. The FastSymbolicGP regressor subsequently reserved 20% of its supplied development data for internal validation. Therefore, its effective partition corresponded approximately to 64% evolutionary training data, 16% internal validation data, and 20% independent testing data. gplearn and PySR received the same outer 80% development partition, although those libraries used their own native training procedures.

Regression inputs were not standardized. This decision preserved the original feature values and made the resulting equations easier to interpret in relation to the source variables. The only exception was the explicitly defined nondimensional physical law representation.

For classification, an initial stratified 80:20 development–test split was followed by a second stratified split of the development data, producing 60% training data, 20% validation data, and 20% test data. A StandardScaler was fitted exclusively on the 60% training partition. The fitted transformation was then applied to the validation and test partitions. Consequently, neither validation nor test information was used to estimate preprocessing parameters.

In the binary threshold experiments, models were fitted on the 60% training partition, thresholds were selected using the 20% validation partition, and final metrics were calculated once on the untouched 20% test partition.

For the principal multiclass experiments, the scaled training and validation partitions were recombined into an 80% development set. FastSymbolicGP applied its configured internal validation procedure to this development set, whereas the gplearn one-versus-rest baseline used the supplied development data according to its native implementation.

All stochastic experiments used the same five random seeds, 0–4, wherever the compared method exposed an applicable random-state parameter.

2.2.10. Compared Symbolic Methods

FastSymbolicGP was compared with three established symbolic modeling approaches:

1. **gplearn**, representing a conventional Python genetic programming implementation [8];
2. **PySR**, representing a modern high-performance symbolic regression framework [9]; and
3. **PySINDy**, representing sparse identification of nonlinear dynamical systems [11].

The methods were used only in tasks for which their design was methodologically appropriate. PySR was used for scientific regression, physical law recovery, scalability, and scalar governing-equation discovery. It was not forced into the principal classification comparison because it is primarily a symbolic regression system rather than a native prob-

abilistic classifier. PySINDy was used only for dynamical system identification. gplearn was used for regression and binary classification, while multiclass classification was implemented by wrapping its binary classifier in a one-versus-rest strategy.

The task-specific comparisons are summarized in Table 8.

Table 8. Use of the compared symbolic methods across benchmark components.

Experiment	FastSymbolicGP	gplearn	PySR	PySINDy
Scientific regression	Yes	Yes	Yes	No
Physical law recovery	Yes	Yes	Yes	No
Binary classification	Yes	Yes	No	No
Multiclass classification	Yes	Yes, OVR	No	No
Governing-equation discovery	Yes	Yes	Yes	Yes
Scalability	Yes	Yes	Yes	No

Search mechanisms differ substantially among FastSymbolicGP, gplearn, PySR, and PySINDy. Therefore, population and iteration settings were reported transparently, but the experiments were not described as having identical candidate-evaluation budgets.

gplearn Configuration

For the principal comparison, gplearn used a population of 250, 20 generations, and a tournament size of 10. The same primitive set, initial depth range, constant range, and genetic operator probabilities were used wherever supported. Its parsimony coefficient was 0.001.

For regression, gplearn evaluated each candidate on a random 90% subsample of the supplied development data through `max_samples=0.9`. For classification, all supplied training samples were used. gplearn was executed with one processing job.

PySR Configuration

PySR used 20 symbolic regression iterations, two populations, and 50 individuals per population. The maximum permitted expression size was 60. Its binary operators were addition, subtraction, multiplication, and division, while its unary operators were square root, logarithm, absolute value, sine, and cosine. Model selection was set to `best`. Deterministic serial execution and the corresponding FastSymbolicGP random seed were used.

These settings constitute a moderate PySR profile rather than an assertion of exact computational equivalence with the 250-by-20 genetic programming runs.

PySINDy Configuration

PySINDy used a polynomial feature library of degree two and sequentially thresholded least squares. The sparsity threshold was 0.05, and the regularization parameter was 10^{-5} . PySINDy identified all state derivative equations jointly, whereas the symbolic regression methods identified each derivative equation separately. The distinction was retained in the reported results.

2.2.11. Real Scientific Regression Experiments

The real scientific regression benchmark included Concrete Compressive Strength, Energy Efficiency Heating Load, Airfoil Self-Noise, and Parkinson's Total UPDRS. These datasets were selected to provide examples from civil engineering, building physics, aeroacoustics, and biomedicine.

FastSymbolicGP, gplearn, and PySR were trained using the same outer development and test partitions for every seed. For each fitted model, the benchmark recorded

- R^2 ,
- RMSE,
- mean absolute error,
- fitting time,
- prediction time,
- process-memory change,
- peak Python memory,
- the raw symbolic expression,
- the complete scaled expression, and
- expression-complexity indicators.

The purpose of this experiment was to assess whether the methods could produce useful explicit equations from real measurements rather than only from idealized analytical functions.

2.2.12. Known Physical Law Recovery

Five analytical physical relationships were used to test whether the symbolic tools could recover equations with recognizable scientific structure:

$$p = mv, \quad E_k = \frac{1}{2}mv^2, \quad \rho = \frac{m}{V}, \quad P = Fv, \quad T = 2\pi\sqrt{\frac{L}{g}}. \quad (39)$$

For each law and seed, 2000 samples were generated from predefined physically plausible ranges. The clean target y_{clean} was calculated from the reference equation. Gaussian noise was added to the development target according to

$$y_{\text{noisy}} = y_{\text{clean}} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.01, \sigma_y), \quad (40)$$

where σ_y is the standard deviation of the clean target. Training was therefore performed using targets containing 1% relative noise, while final predictions were evaluated against clean test targets. This design examined whether the methods could recover an underlying relationship rather than reproduce the injected noise.

Each physical problem was evaluated using two input representations:

1. the original dimensional variables; and
2. a nondimensional representation.

For nondimensionalization, each input feature was divided by its median absolute magnitude:

$$\tilde{x}_j = \frac{x_j}{\text{median}(|x_j|)}, \quad (41)$$

and the target was divided by its median absolute magnitude:

$$\tilde{y} = \frac{y}{\text{median}(|y|)}. \quad (42)$$

Zero-valued scaling factors were replaced by one to avoid division by zero.

2.2.13. Dimensional-Consistency Analysis

FastSymbolicGP does not impose physical units during evolution. Variables with incompatible units can therefore be combined unless the user constructs a dimensionally constrained primitive set or nondimensionalizes the data. This limitation was addressed explicitly rather than assuming that a low prediction error implies physical validity.

A strict dimensional-consistency check was applied to equations recovered from the known physical law problems. Dimension vectors were represented using the base dimensions mass, length, and time:

$$\mathbf{d} = (d_M, d_L, d_T). \quad (43)$$

The following rules were used:

- numerical constants were assumed to be dimensionless;
- addition and subtraction were valid only when their operands had identical dimension vectors;
- multiplication added dimension exponents;
- division subtracted dimension exponents;
- a numerical power multiplied the dimension vector by its exponent;
- logarithmic, exponential, trigonometric, and hyperbolic functions required dimensionless arguments;
- absolute value preserved the dimension of its argument; and
- the inferred output dimension had to match the known target dimension.

An expression was marked dimensionally valid only when all internal operations and the final output satisfied these rules. For the nondimensional experiments, all input and target dimension vectors were set to zero.

This check was deliberately performed after model discovery. Therefore, the current method should not be described as unit-aware or physics-constrained. Predictive performance, resemblance to the reference law, and dimensional validity were reported as separate properties. An equation was not considered physically meaningful solely because it achieved a low prediction error.

2.2.14. Binary Classification, Threshold Selection, and Class Imbalance

Binary classification was evaluated using Breast Cancer Wisconsin, Spambase, and controlled synthetic datasets. The synthetic datasets contained 3000 observations, 20 features, 10 informative features, five redundant features, a class-separation value of 1.2, and a label-noise probability of 0.02. Minority-class fractions of 0.50, 0.20, 0.10, and 0.05 were examined.

Three FastSymbolicGP decision configurations were evaluated:

1. unweighted learning with a fixed threshold of 0.5;
2. unweighted learning with a validation-optimized MCC threshold; and
3. balanced class weighting with a validation-optimized balanced-accuracy threshold.

gplearn was evaluated with a fixed threshold of 0.5 and with a validation-optimized MCC threshold.

For a selected threshold τ , the binary decision rule was

$$\hat{y} = \begin{cases} 1, & p(y = 1 | \mathbf{x}) \geq \tau, \\ 0, & p(y = 1 | \mathbf{x}) < \tau. \end{cases} \quad (44)$$

Thresholds were optimized exclusively on the validation partition. The candidate set consisted of 0.5 and probability quantiles evaluated at 199 equally spaced levels between 0.01 and 0.99:

$$\mathcal{T} = \{0.5\} \cup \{Q_q(\mathbf{p}_{\text{val}}) \mid q \in \text{linspace}(0.01, 0.99, 199)\}. \quad (45)$$

For MCC optimization,

$$\tau_{\text{MCC}} = \arg \max_{\tau \in \mathcal{T}} \text{MCC}(\mathbf{y}_{\text{val}}, \mathbb{I}[\mathbf{p}_{\text{val}} \geq \tau]). \quad (46)$$

For balanced-accuracy optimization,

$$\tau_{\text{BACC}} = \arg \max_{\tau \in \mathcal{T}} \text{BACC}(\mathbf{y}_{\text{val}}, \mathbb{I}[\mathbf{p}_{\text{val}} \geq \tau]). \quad (47)$$

The selected threshold was then frozen and applied once to the independent test probabilities. This design directly addresses the practical classification problem that a threshold of 0.5 may be inappropriate when classes are imbalanced or when probability distributions are asymmetric.

The balanced FastSymbolicGP configuration applied inverse-frequency class weighting during fitness evaluation. Its purpose was to prevent the majority class from dominating logarithmic loss as the minority fraction decreased. The imbalance experiment separately measured the effects of class weighting and decision-threshold selection.

2.2.15. Multiclass Symbolic Classification

Multiclass classification was evaluated on Iris, Wine, and the ten-class Digits dataset. FastSymbolicGP was evaluated using one-versus-rest and shared-softmax strategies. gplearn was evaluated through a one-versus-rest wrapper.

One-Versus-Rest Strategy

For K classes, the one-versus-rest strategy evolved one binary symbolic classifier per class. For class k , the target was transformed into

$$y_i^{(k)} = \begin{cases} 1, & y_i = k, \\ 0, & y_i \neq k. \end{cases} \quad (48)$$

Each classifier produced a class-specific probability

$$p_k(\mathbf{x}) = \sigma(a_k f_k(\mathbf{x}) + b_k). \quad (49)$$

The final prediction was obtained as

$$\hat{y} = \arg \max_{k \in 1, \dots, K} p_k(\mathbf{x}). \quad (50)$$

The complete multiclass output therefore consisted of K explicit symbolic probability equations rather than one opaque multiclass model.

Shared-Sigmoid Strategy

The shared-sigmoid strategy evolved a collection of symbolic components shared across classes. The number of components was

$$m = \min(12, p), \quad (51)$$

where p is the number of input features. The shared components were combined into class-specific scores, which were transformed using the softmax function:

$$p_k(\mathbf{x}) = \frac{\exp[z_k(\mathbf{x})]}{\sum_{j=1}^K \exp[z_j(\mathbf{x})]}. \quad (52)$$

The predicted class was the class with maximum softmax probability. This strategy was included to determine whether reusing a common symbolic representation could reduce the redundancy associated with evolving a completely independent equation for every class.

Balanced class weighting was enabled for both FastSymbolicGP multiclass strategies. Additional multiclass probability calibration was disabled so that the reported equations corresponded directly to the symbolic models used for prediction.

2.2.16. Dynamical System Identification

The governing-equation benchmark used the Lorenz and Lotka–Volterra systems.

The Lorenz system was defined as

$$\dot{x} = 10(y - x), \quad \dot{y} = x(28 - z) - y, \quad \dot{z} = xy - \frac{8}{3}z. \quad (53)$$

The Lotka–Volterra system was defined as

$$\dot{x} = 1.5x - xy, \quad \dot{y} = 0.75xy - y. \quad (54)$$

Trajectories were generated using `solve_ivp` with relative and absolute tolerances of 10^{-10} . Each system contained 3000 equally spaced points over the interval $[0, 20]$. Exact derivatives were evaluated from the known right-hand side rather than estimated through finite differences.

The first 75% of each trajectory was used for training and the remaining 25% for testing. The chronological split prevented future trajectory points from being included in the training partition.

FastSymbolicGP, gplearn, and PySR independently modeled each derivative as a scalar symbolic regression problem:

$$\dot{x}_j = f_j(\mathbf{x}). \quad (55)$$

PySINDy identified all state equations jointly. For each symbolic regression method, the benchmark stored the discovered derivative equation, the known reference equation, test-set regression metrics, fitting time, and expression complexity.

2.2.17. Parameter-Sensitivity Analysis

The principal population and generation settings were contextualized using a full factorial sensitivity experiment. Population sizes were

$$P \in 100, 250, 500, 1000, \quad (56)$$

and generation limits were

$$G \in 5, 10, 20, 50. \quad (57)$$

Every population–generation combination was evaluated with five random seeds on three representative tasks:

- Friedman 1 regression with 2000 samples, 10 features, and Gaussian noise;
- Breast Cancer Wisconsin binary classification; and
- Iris shared-softmax multiclass classification.

For the sensitivity experiments, the maximum depth was eight and the maximum number of nodes was 127. Tournament size was calculated as

$$k = \min\left(20, \max\left(3, \left\lfloor \frac{P}{10} \right\rfloor\right)\right). \quad (58)$$

The sensitivity analysis measured predictive performance, fitting time, expression complexity, actual generations, and evaluation count. It was used to determine whether increasing the evolutionary budget produced consistent improvements and to demonstrate that the principal $P = 250$, $G = 20$ profile represented a computational compromise rather than an arbitrarily asserted optimum.

2.2.18. Ablation Analysis

Ablation experiments were conducted on the Friedman 1 regression problem using the principal paper profile. The following configurations were compared:

1. the Numba-only reference configuration;
2. subtree mutation disabled;
3. crossover disabled;
4. adaptive parsimony disabled; and
5. four-island evolution.

In the subtree-mutation ablation, the subtree-mutation probability was set to zero. In the crossover ablation, the crossover probability was set to zero. The remaining probability was handled by the estimator's native reproduction logic.

For the parsimony ablation, the adaptive parsimony mechanism was replaced with no parsimony penalty.

The island configuration divided evolution among four islands. Migration was performed every 10 generations, two individuals were migrated at each event, and migration followed a ring topology. Parallel island execution was disabled so that the experiment remained compatible with the one-thread timing protocol.

The reference configuration already disabled semantic duplicate elimination. Therefore, semantic duplicate elimination was not treated as a valid ablation factor in the methodological interpretation.

2.2.19. Scalability Analysis

Scalability was evaluated using a controlled nonlinear regression problem:

$$y = 2x_1 - 1.5x_2^2 + \sin(x_3) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.1^2). \quad (59)$$

The number of observations was varied over

$$N \in 1000, 5000, 10,000, 25,000, 50,000, \quad (60)$$

while the number of input features was varied over

$$p \in 5, 10, 20, 50. \quad (61)$$

Features not appearing in the generating equation acted as irrelevant variables. This allowed the benchmark to examine both increasing sample size and increasing dimensionality.

The scalability profile used 250 individuals, 10 generations, and three seeds. FastSymbolicGP, gplearn, and PySR were compared using fitting time, prediction time, memory measurements, R^2 , RMSE, MAE, and symbolic complexity.

2.2.20. Equation Extraction and Equation-Pool Analysis

The benchmark stored actual symbolic output for every successful run. For FastSymbolicGP, the following were recorded:

- the raw evolved expression;
- the complete expression including output scale and intercept;

- the classification sigmoid and decision threshold;
- class-specific equations for one-versus-rest models;
- shared symbolic components for shared-softmax models;
- candidate expressions exposed through the fitted Pareto-front record; and
- up to the first ten stored hall-of-fame expressions.

For gplearn, up to ten of the strongest available expressions from the final generation were stored. If the final population was unavailable, its selected program was stored. For one-versus-rest classification, equations were extracted independently for every class.

For PySR, the complete equations table was extracted, including the equation, reported loss, complexity, and score. Equations obtained from scientific regression, physical law recovery, classification, and dynamical system experiments were combined into a unified equation-pool file after duplicate rows were removed.

This procedure makes it possible to compare not only the final metric achieved by each tool but also the range of candidate equations that it discovered.

2.2.21. Interpretability and Complexity Measures

Interpretability was evaluated quantitatively using several complementary indicators:

- expression token length;
- tree-node count;
- expression depth;
- DAG-node count;
- number of distinct variables;
- number of nonlinear operators;
- variable-use frequencies;
- function-use frequencies;
- number of shared subexpressions; and
- DAG compression ratio.

Tree-node count measures the size of the original symbolic tree. DAG-node count measures the size obtained when repeated symbolic subexpressions are represented once. Although DAG execution was disabled, DAG size was retained as a descriptive measure of structural redundancy.

For expressions generated by different symbolic tools, an implementation-independent text complexity score was also calculated. Let N_{tok} be the number of parsed expression tokens, D the estimated parenthesis depth, and N_{nonlin} the number of nonlinear function calls. The score was defined as

$$C_{\text{text}} = N_{\text{tok}} + 2D + N_{\text{nonlin}}. \quad (62)$$

The coefficient of two assigns additional weight to nested expression structure, while the nonlinear-operator term distinguishes expressions containing functions such as logarithm, sine, cosine, or square root from similarly sized linear arithmetic expressions.

The composite score is not presented as a universal measure of human interpretability. It is used as a reproducible benchmark indicator that permits equations produced by different libraries to be compared using the same parsing procedure.

For manuscript presentation, two representative equations were selected for each applicable dataset–method combination:

1. the equation with the strongest predictive metric; and
2. the simplest equation whose predictive metric was within 1% of the best observed value.

For regression, R^2 was used as the predictive criterion. For multiclass classification, macro-F1 was used. For binary classification, balanced accuracy was used. This selection procedure supports direct discussion of why a slightly less accurate equation may be scientifically preferable when it is substantially simpler.

2.2.22. Evaluation Metrics

Regression performance was evaluated using

- coefficient of determination, R^2 ;
- root mean squared error;
- mean absolute error.

Binary classification was evaluated using

- accuracy;
- balanced accuracy;
- F1-score;
- Matthews correlation coefficient; and
- receiver operating characteristic area under the curve.

Multiclass classification was evaluated using

- accuracy;
- balanced accuracy;
- macro-averaged F1-score;
- weighted F1-score; and
- multiclass Matthews correlation coefficient.

In addition to predictive metrics, fitting time, prediction time, resident-memory change, peak Python memory, actual generation count, evaluation count, stopping reason, and expression-complexity measures were stored whenever supported.

All final predictive metrics were calculated on independent test data that were not used for symbolic evolution, preprocessing estimation, threshold optimization, or final expression selection.

2.2.23. Statistical Analysis

Results were paired by dataset, random seed, and any experiment-specific factors such as representation, sample size, feature count, or threshold strategy.

Two-sided paired Wilcoxon signed-rank tests were used because the distributions of stochastic symbolic learning results cannot be assumed to be Gaussian. A comparison was performed only when at least three valid paired observations were available.

For each comparison, the benchmark reported

- the mean result for both methods;
- mean and median paired differences;
- a 95% bootstrap confidence interval for the mean paired difference;
- the Wilcoxon statistic;
- the uncorrected p -value; and
- the rank-biserial effect size.

Bootstrap intervals were calculated using 5000 resamples and a fixed bootstrap seed. To control the family-wise error rate across the collection of comparisons, all p -values were adjusted using the Holm procedure. Statistical significance was evaluated at an adjusted level of

$$\alpha = 0.05. \quad (63)$$

Statistical significance, effect size, predictive magnitude, runtime, and equation complexity were interpreted jointly. A statistically significant result was not automatically interpreted as practically important when its absolute magnitude was negligible.

2.2.24. Generation of Tables and Figures

All benchmark figures were generated directly from the raw comma-separated result files using Python and Matplotlib. Times New Roman typography, black axes, grayscale or black-and-white plotting elements, a nominal figure size of 12 × 8 inches, and a resolution of 300 dpi were used.

The generated figures included scientific regression performance, fitting-time comparisons, predictive-performance versus complexity plots, scalability curves, class-imbalance and threshold comparisons, and parameter-sensitivity heat maps.

No benchmark figure was generated using a generative artificial intelligence image system. The diagrams used to explain the library and benchmark workflow were manually constructed, while all empirical figures were programmatically generated from recorded experimental results.

2.2.25. Hardware and Software Environment

All timed experiments were executed on the same workstation. The hardware environment is summarized in Table 9.

Table 9. Hardware environment used for the revised experiments.

Component	Specification
Operating system	Microsoft Windows 10 Education, 64-bit, version 10.0.19045 (Build 19045)
Processor	AMD PROCESSOR MODEL
Processor architecture	x64-based PC
Physical processors	1
Logical processors	16
Physical memory	32,696 MB (approximately 32 GB)
Graphics processing unit (GPU) acceleration	Not used
FastSymbolicGP execution	One central processing unit (CPU) thread using the postfix Numba backend

The principal fixed software versions are summarized in Table 10.

Table 10. Principal software environment used for the revised benchmark.

Component	Version or Role
Python	3.11.15
FastSymbolicGP	0.7.0
scikit-learn	1.8.0
gplearn	0.4.3
PySR	1.5.10
PySINDy	2.1.0
SymPy	1.14.0
statsmodels	0.14.6
ucimlrepo	0.0.7
Numba	Postfix expression evaluation
NumPy and pandas	Numerical computation and result storage
SciPy	Statistical analysis and trajectory integration
Matplotlib	Programmatic figure generation

The benchmark automatically stored the exact Python, operating-system, processor, package-version, CPU, memory, backend, and execution-engine information with the raw output. This allowed the computational environment to be verified from the result files rather than reconstructed manually.

2.2.26. Reproducibility and Result Storage

Every experiment stored raw results before aggregation. Separate files were produced for scientific regression, physical law recovery, binary classification, multiclass classification, sensitivity analysis, ablation, scalability, and dynamical system identification. Separate equation-pool files were generated for regression, classification, and dynamical systems and subsequently combined.

The analysis procedure generated

- a unified raw-result table;
- aggregated mean, standard deviation, median, and sample-count tables;
- paired statistical-test results;
- representative-equation tables;
- the combined equation pool;
- manuscript-ready L^AT_EX tables;
- black-and-white figures; and
- a machine-readable analysis manifest.

Fixed seeds, deterministic PySR execution, single-thread FastSymbolicGP and gplearn runs, an untimed Numba warm-up, backend assertions, incremental raw-result storage, and automatic environment recording were used to improve reproducibility.

Nevertheless, exact symbolic expressions may remain dependent on software versions, floating-point behavior, and implementation-specific stochastic operations. For this reason, the complete benchmark scripts, raw result files, environment record, equation pools, and fitted-model reports should be archived together with the manuscript or its supplementary material.

3. Results

The complete experimental campaign produced 940 successful benchmark records, with no failed runs in the final result set. The experiments comprised 60 real-world scientific regression runs, 85 dynamical system identification runs, 150 physical law recovery runs, 150 binary classification runs, 45 multiclass classification runs, 180 scalability runs, 240 parameter-sensitivity runs, and 30 ablation runs. In addition, 5619 candidate symbolic models were retained in the equation pools for the subsequent complexity and representative-expression analyses. Unless stated otherwise, the reported values are means and standard deviations over independent random seeds. Paired statistical comparisons were performed on matched dataset–seed observations using two-sided Wilcoxon signed-rank tests. Holm correction was applied within each benchmark family, and rank-biserial correlation was used as the effect-size measure.

3.1. Scientific Regression on Real-World Data

Table 11 summarizes the predictive accuracy, fitting time, and symbolic complexity obtained on four real-world scientific and engineering regression datasets. FastSymbolicGP achieved the highest mean test R^2 on every dataset. Its mean R^2 was 0.6132 on Airfoil Self-Noise, 0.6694 on Concrete Compressive Strength, 0.9189 on Energy-Efficiency Heating Load, and 0.1910 on Parkinson's Total UPDRS. The corresponding PySR values were 0.3609, 0.5249, 0.9016, and 0.1219, whereas gplearn obtained substantially lower results, including a strongly negative mean R^2 on Airfoil Self-Noise.

Table 11. Scientific regression results. Values are mean $\pm$ standard deviation over five seeds. Complexity denotes the composite symbolic-complexity score.

Dataset	Method	Test R^2	Test RMSE	Fit Time (s)	Complexity
Airfoil Self-Noise	FastSymbolicGP	0.6132 ± 0.0557	4.2311	13.6296	61.6
	PySR	0.3609 ± 0.0193	5.4530	4.6045	58.4
	gplearn	-16.7593 ± 8.8246	27.1965	29.3458	195.4
Concrete Compressive Strength	FastSymbolicGP	0.6694 ± 0.0291	9.4934	13.5452	64.6
	PySR	0.5249 ± 0.0371	11.3668	6.6559	37.2
	gplearn	0.3996 ± 0.2156	12.6249	28.3402	101.6
Energy-Efficiency Heating Load	FastSymbolicGP	0.9189 ± 0.0189	2.8450	12.8008	71.2
	PySR	0.9016 ± 0.0323	3.1160	3.1681	74.4
	gplearn	0.7324 ± 0.1145	5.1308	28.4713	152.6
Parkinson's Total UPDRS	FastSymbolicGP	0.1910 ± 0.0657	9.6820	14.6719	65.2
	PySR	0.1219 ± 0.0683	10.0908	16.1412	12.8
	gplearn	0.0119 ± 0.1099	10.6963	28.7801	148.4

Across all 20 matched dataset–seed cases, FastSymbolicGP obtained an overall mean test R^2 of 0.5981, compared with 0.4773 for PySR and -3.9038 for gplearn. The difference between FastSymbolicGP and PySR was statistically significant ($\Delta R^2 = 0.1208$, $p_{\text{Holm}} = 4.20 \times 10^{-4}$, $r_{\text{rb}} = 0.895$). The difference relative to gplearn was also statistically significant ($p_{\text{Holm}} = 2.9 \times 10^{-5}$, $r_{\text{rb}} = 0.981$). The dataset-level variation in predictive performance is illustrated in Figure 4.

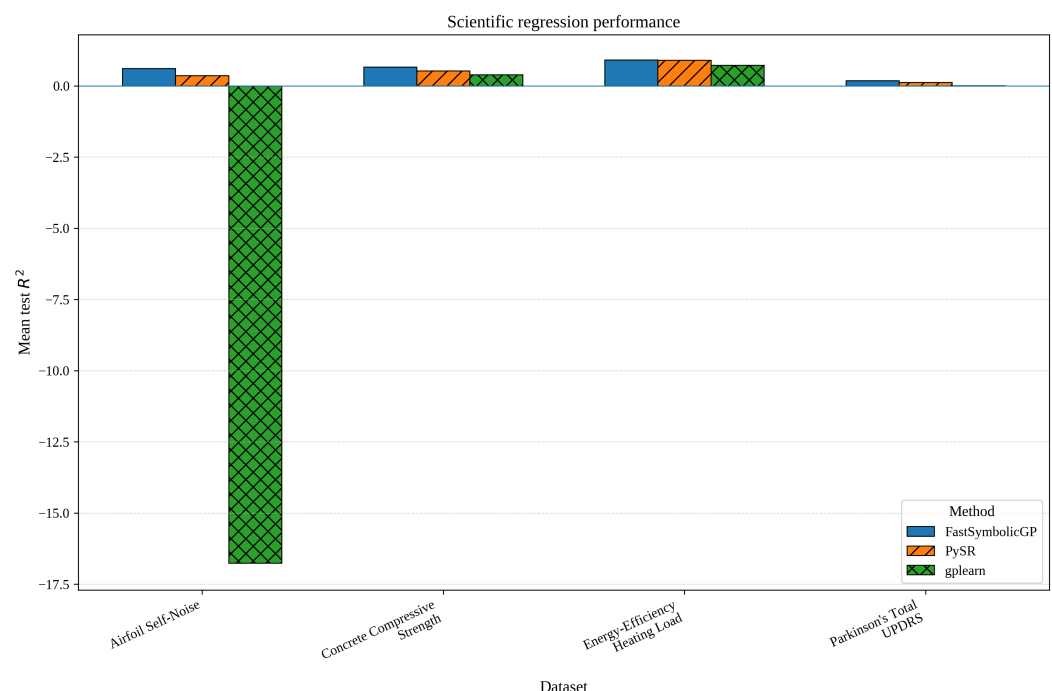


Figure 4. Mean test R^2 on the real-world scientific regression datasets. The strongly negative gplearn result on Airfoil Self-Noise is retained rather than clipped.

As shown in Figure 4, FastSymbolicGP produced the highest mean test R^2 on all four datasets. Its advantage was especially pronounced for Airfoil Self-Noise and Concrete Compressive Strength, whereas the difference between FastSymbolicGP and PySR was smaller on the Energy-Efficiency Heating Load dataset. Nevertheless, FastSymbolicGP remained the best-performing method on that dataset, with a mean test R^2 of 0.9189. The Parkinson's Total UPDRS dataset was the most difficult problem for all three methods,

as indicated by the comparatively low positive R^2 values. The strongly negative gplearn result on Airfoil Self-Noise indicates that its predictions generalized substantially worse than a constant prediction based on the test-set mean. This result was retained in the figure to avoid artificially improving the apparent performance of the method, although it necessarily expands the vertical-axis range and visually compresses the positive results.

Predictive performance must also be considered together with computational cost. The mean fitting times obtained for the four datasets are compared in Figure 5. A logarithmic vertical axis is used because the fitting times differed substantially among the evaluated methods.

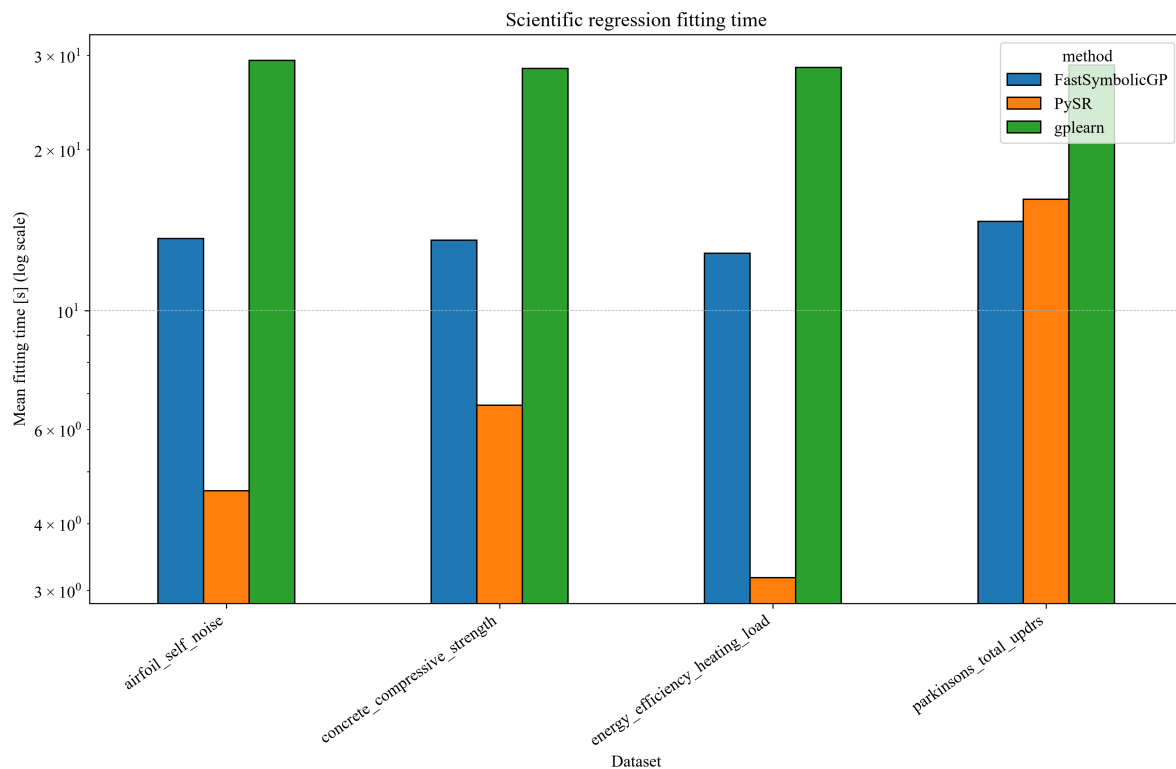


Figure 5. Mean fitting time for the scientific regression experiments. The vertical axis is logarithmic.

Figure 5 shows that PySR was the fastest method on Airfoil Self-Noise, Concrete Compressive Strength, and Energy-Efficiency Heating Load. FastSymbolicGP required an average of 13.66 s per fit across the four datasets, compared with 7.64 s for PySR and 28.73 s for gplearn. FastSymbolicGP was therefore approximately 2.10 times faster than gplearn, while PySR was approximately 1.79 times faster than FastSymbolicGP on average. On Parkinson's Total UPDRS, however, the fitting times of FastSymbolicGP and PySR were more similar, at 14.67 s and 16.14 s, respectively, with FastSymbolicGP being slightly faster.

The mean symbolic-complexity score of FastSymbolicGP was 65.7, which was markedly lower than the gplearn mean of 149.5 but higher than the PySR mean of 45.7. These results indicate that FastSymbolicGP provided the strongest overall predictive performance while maintaining substantially lower runtime and symbolic complexity than gplearn. PySR produced the most compact expressions and was generally the fastest method, but this computational advantage was accompanied by lower predictive accuracy on all four datasets. FastSymbolicGP therefore occupied an intermediate position in the runtime–complexity trade-off while providing the best predictive results.

3.2. Dynamical System Identification

The dynamical system experiments evaluated the recovery of the Lorenz and Lotka–Volterra differential equations. Table 12 reports results for the three general-purpose symbolic regression methods and for PySINDy, which is included as a specialized sparse system-identification baseline. For Lorenz, FastSymbolicGP obtained a mean R^2 of 0.9049, exceeding PySR and gplearn. For Lotka–Volterra, all methods produced high-quality models, with FastSymbolicGP reaching $R^2 = 0.9998 \pm 0.0002$. PySINDy recovered both systems exactly under the predefined sparse candidate library, as expected for this specialized setting.

Table 12. Dynamical system identification results. Complexity is the composite score of the recovered right-hand-side expressions.

System	Method	Test R^2	Fit Time (s)	Complexity
Lorenz	FastSymbolicGP	0.9049 ± 0.2040	9.8674	47.0
	PySR	0.7779 ± 0.3899	6.5063	25.7
	gplearn	0.7850 ± 0.1931	30.1178	180.3
	PySINDy	1.0000 ± 0.0000	0.0076	22.0
Lotka–Volterra	FastSymbolicGP	0.9998 ± 0.0002	11.3677	56.2
	PySR	1.0000 ± 0.0000	4.6727	10.8
	gplearn	0.9950 ± 0.0055	28.1363	55.1
	PySINDy	1.0000 ± 0.0000	0.0050	13.0

When the 25 state-equation and seed pairs available for the general-purpose methods were pooled, FastSymbolicGP achieved a mean R^2 of 0.9428, compared with 0.8690 for gplearn and 0.8667 for PySR. The improvement over gplearn was statistically significant ($p_{\text{Holm}} = 0.0057$, $r_{\text{rb}} = 0.775$), whereas the difference from PySR was not significant after Holm correction ($p_{\text{Holm}} = 0.6299$). FastSymbolicGP was approximately 2.80 times faster than gplearn and reduced the mean complexity score from 130.2 to 50.7. PySR remained faster and produced simpler expressions on average, illustrating that the principal advantage of FastSymbolicGP in this experiment was its combination of accuracy and moderate complexity rather than absolute dominance on every criterion.

The state-wise results show that the Lorenz y equation was the most difficult component: FastSymbolicGP obtained a mean R^2 of 0.7210 for this state, compared with 1.0000 for the Lorenz x equation and 0.9936 for the z equation. Nevertheless, the library exactly recovered the compact Lorenz relation

$$\widehat{\frac{dx}{dt}} = -10(x - y) = 10(y - x), \quad (64)$$

which matches the reference equation. This example demonstrates that the approach can recover scientifically recognizable structures rather than only numerically accurate black-box surrogates.

3.3. Physical Law Recovery and Dimensional Consistency

Table 13 compares direct learning from dimensional variables with learning after nondimensionalization. Predictive accuracy alone was insufficient to guarantee physically meaningful expressions. On the raw dimensional representation, FastSymbolicGP achieved a mean R^2 of approximately 1.0000, but only 24% of the recovered raw expressions passed the strict post hoc dimensional-consistency check. After nondimensionalization, the validity rate increased to 96%, while the mean R^2 remained 1.0000. Nondimensionalization also reduced the mean fitting time from 11.18 to 5.13 s and the mean complexity score from 55.36 to 29.08, corresponding to reductions of approximately 54.1% and 47.5%, respectively.

Table 13. Physical law recovery under dimensional and nondimensional representations. Validity is the percentage of raw recovered expressions passing the strict dimensional-consistency check.

Representation	Method	Test R^2	Fit Time (s)	Complexity	Validity (%)
Dimensional raw	FastSymbolicGP	1.0000 ± 0.0000	11.1750	55.36	24
	PySR	0.9990 ± 0.0021	5.7512	14.64	76
	gplearn	0.9890 ± 0.0208	27.4424	58.08	60
Nondimensional	FastSymbolicGP	1.0000 ± 0.0000	5.1316	29.08	96
	PySR	1.0000 ± 0.0001	4.6328	15.64	100
	gplearn	0.9933 ± 0.0170	27.3084	52.60	88

The representative pendulum model recovered from the dimensional data was

$$\hat{T} = 6.2854\sqrt{\frac{L}{g}} - 0.00156, \quad (65)$$

with test $R^2 = 1.0000$ and complexity score 26. The leading coefficient is close to the theoretical value $2\pi \approx 6.2832$, and the intercept is negligible. Similarly, the recovered density expressions simplified numerically to $\rho \approx m/V$. These cases show that the equation pool contains models that are both accurate and directly comparable with known physical laws. At the same time, the low validity rate on untransformed dimensional inputs demonstrates why explicit unit handling or nondimensionalization is necessary: a model may fit the data almost perfectly while violating dimensional homogeneity.

3.4. Binary Classification, Class Imbalance, and Threshold Selection

Table 14 reports balanced accuracy for the two real-world binary datasets and for four synthetic minority-class fractions. The unweighted FastSymbolicGP classifier was evaluated with the fixed threshold 0.5 and with a threshold selected on the validation subset by maximizing MCC. The balanced variant used class weighting and selected its threshold by validation balanced accuracy. Both fixed and validation-optimized gplearn results are included for comparison.

Table 14. binary classification balanced accuracy. FSGP-U denotes unweighted FastSymbolicGP, FSGP-B denotes class-balanced FastSymbolicGP, MCC denotes validation-based threshold selection maximizing MCC, and BA denotes validation-based threshold selection maximizing balanced accuracy.

Dataset	FSGP-U, 0.5	FSGP-U, MCC	FSGP-B, BA	gplearn, 0.5	gplearn, MCC
Breast Cancer	0.963 ± 0.012	0.951 ± 0.016	0.956 ± 0.016	0.966 ± 0.013	0.960 ± 0.011
Spambase	0.862 ± 0.015	0.878 ± 0.010	0.889 ± 0.018	0.848 ± 0.029	0.862 ± 0.028
Synthetic, 5% minority	0.713 ± 0.033	0.758 ± 0.045	0.827 ± 0.046	0.530 ± 0.038	0.671 ± 0.056
Synthetic, 10% minority	0.833 ± 0.023	0.833 ± 0.015	0.865 ± 0.023	0.593 ± 0.081	0.663 ± 0.102
Synthetic, 20% minority	0.845 ± 0.017	0.859 ± 0.011	0.876 ± 0.015	0.726 ± 0.085	0.778 ± 0.042
Synthetic, 50% minority	0.888 ± 0.009	0.886 ± 0.009	0.878 ± 0.031	0.811 ± 0.042	0.801 ± 0.049

The largest benefit of class balancing occurred under severe imbalance. At a 5% minority fraction, the balanced FastSymbolicGP configuration achieved a mean balanced accuracy of 0.827, compared with 0.713 for the unweighted fixed-threshold configuration and 0.530 for fixed-threshold gplearn. Validation-based MCC threshold selection improved the unweighted FastSymbolicGP result from 0.713 to 0.758 in this most imbalanced setting. At a 50% minority fraction, however, class balancing was unnecessary and the fixed unweighted configuration achieved the best FastSymbolicGP result. This confirms that

class weighting and threshold optimization should be treated as task-dependent controls rather than universally superior defaults.

Across all 30 matched dataset–seed cases, fixed-threshold FastSymbolicGP achieved mean balanced accuracy, F1, and MCC values of 0.8507, 0.8034, and 0.7557, respectively. The corresponding gplearn values were 0.7458, 0.5926, and 0.5452. All three differences remained significant after Holm correction ($p_{\text{Holm}} < 2.4 \times 10^{-5}$), with large rank-biserial effects between 0.837 and 0.880. FastSymbolicGP also reduced the mean fitting time from 27.30 to 18.23 s, giving a speedup of approximately 1.50. With validation-optimized MCC thresholds, FastSymbolicGP increased its mean balanced accuracy to 0.8607 and F1 to 0.8115, while continuing to significantly outperform gplearn.

The balanced FastSymbolicGP configuration increased the overall mean balanced accuracy further to 0.8819. This gain was accompanied by lower overall F1 and MCC values than the unweighted MCC-optimized configuration because the balanced objective deliberately shifted the decision boundary toward minority-class recall. The trade-off was statistically detectable for all three measures ($p_{\text{Holm}} = 0.0037$). Consequently, the appropriate configuration depends on whether the application prioritizes symmetric class-level performance or a single positive-class operating point.

The influence of class imbalance was examined by progressively varying the minority-class fraction while comparing class weighting and threshold-selection strategies. Figure 6 presents the mean balanced accuracy obtained by the evaluated configurations at minority-class fractions of 0.05, 0.10, 0.20, and 0.50.

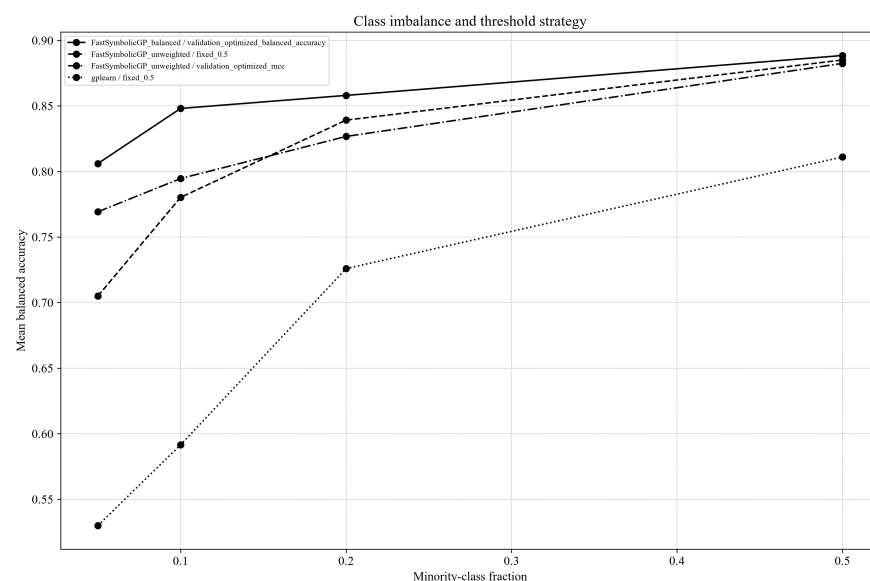


Figure 6. Effect of minority-class fraction, class balancing, and validation-based threshold selection on mean balanced accuracy.

As shown in Figure 6, all configurations improved as the minority-class fraction increased and the classification problem became more balanced. The balanced FastSymbolicGP configuration with a validation-optimized balanced-accuracy threshold produced the highest balanced accuracy across all four class distributions. Its advantage was most pronounced under severe imbalance, reaching approximately 0.81 at a minority-class fraction of 0.05, compared with approximately 0.71 for the unweighted FastSymbolicGP configuration using the fixed threshold of 0.5 and approximately 0.53 for gplearn.

Validation-based MCC threshold optimization substantially improved the unweighted FastSymbolicGP configuration under severe imbalance, increasing balanced accuracy to approximately 0.77 at a minority-class fraction of 0.05. However, as the minority-class

fraction approached 0.50, the differences among the three FastSymbolicGP configurations became small, with all three achieving balanced accuracies of approximately 0.88–0.89. In contrast, gplearn remained below the FastSymbolicGP configurations across the complete range of evaluated class distributions.

When averaged across the evaluated imbalance conditions, the balanced FastSymbolicGP configuration increased the overall mean balanced accuracy to 0.8819. This gain was accompanied by lower overall F1 and MCC values than those of the unweighted MCC-optimized configuration because the balanced objective deliberately shifted the decision boundary toward minority-class recall. The trade-off was statistically detectable for all three measures ($p_{\text{Holm}} = 0.0037$). Consequently, the appropriate configuration depends on whether the application prioritizes symmetric class-level performance or a particular positive-class operating point.

3.5. Multiclass Classification

Table 15 compares the one-versus-rest strategy, the shared symbolic-backbone softmax strategy, and a gplearn one-versus-rest baseline. On Iris and Wine, the shared-softmax strategy achieved the highest mean macro-F1 values, reaching 0.980 and 0.990, respectively. On the more difficult ten-class Digits dataset, one-versus-rest FastSymbolicGP was clearly preferable, obtaining a macro-F1 of 0.890, compared with 0.805 for shared softmax and 0.591 for gplearn.

Table 15. Multiclass classification results. Complexity is computed over the complete multiclass symbolic model.

Dataset	Method	Accuracy	Macro-F1	MCC	Fit (s)	Predict (s)	Complexity
Iris	FastSymbolicGP OVR	0.947 ± 0.061	0.947 ± 0.061	0.921 ± 0.090	48.4	0.0020	185.8
	FastSymbolicGP shared softmax	0.980 ± 0.030	0.980 ± 0.030	0.970 ± 0.045	48.3	0.0010	181.4
	gplearn OVR	0.973 ± 0.043	0.973 ± 0.044	0.961 ± 0.065	86.2	0.0024	390.6
Wine	FastSymbolicGP OVR	0.978 ± 0.023	0.979 ± 0.022	0.967 ± 0.035	42.1	0.0021	159.0
	FastSymbolicGP shared softmax	0.989 ± 0.015	0.990 ± 0.014	0.984 ± 0.022	42.1	0.0011	395.8
	gplearn OVR	0.983 ± 0.015	0.984 ± 0.015	0.976 ± 0.022	83.7	0.0026	554.4
Digits	FastSymbolicGP OVR	0.890 ± 0.012	0.890 ± 0.012	0.878 ± 0.013	188.3	0.0090	702.6
	FastSymbolicGP shared softmax	0.808 ± 0.031	0.805 ± 0.031	0.787 ± 0.035	188.1	0.0017	916.6
	gplearn OVR	0.621 ± 0.084	0.591 ± 0.100	0.584 ± 0.090	278.8	0.0096	790.8

Averaged over all 15 dataset–seed pairs, one-versus-rest FastSymbolicGP achieved macro-F1 = 0.9385 and MCC = 0.9220, shared softmax achieved macro-F1 = 0.9249 and MCC = 0.9136, and gplearn achieved macro-F1 = 0.8492 and MCC = 0.8402. Because the relative advantage varied strongly between the small three-class datasets and Digits, the paired predictive differences between FastSymbolicGP OVR and gplearn were not significant after Holm correction. The computational differences were more consistent: FastSymbolicGP OVR reduced mean fitting time from 149.6 to 92.9 s and mean complexity from 578.6 to 349.1, with both differences remaining significant after correction.

No significant difference in macro-F1 or MCC was found between the two FastSymbolicGP multiclass strategies. Shared softmax reduced mean prediction time from 0.0044 to 0.00125 s, corresponding to a 3.53-fold inference speedup, but increased mean symbolic complexity by approximately 42.6%. The results therefore indicate that shared softmax is attractive for compact low-class-count problems and latency-sensitive inference, whereas one-versus-rest is more robust when the class structure is more heterogeneous or the number of classes is larger.

3.6. Scalability with Sample Size and Dimensionality

The scalability benchmark varied the number of samples from 1000 to 50,000 and the number of input features from 5 to 50. Table 16 aggregates the results over the four feature dimensionalities. FastSymbolicGP fitting time increased from 6.79 s at 1000 samples to 11.82 s at 50,000 samples, a factor of 1.74. Over the same range, PySR increased from 2.51 to 73.24 s, a factor of 29.15, while gplearn increased from 14.03 to 19.28 s.

Table 16. Scalability with sample size, averaged over 5, 10, 20, and 50 input features and three seeds.

Samples	FastSymbolicGP Time	PySR Time	gplearn Time	FastSymbolicGP R^2	PySR R^2	gplearn R^2
1000	6.79	2.51	14.03	0.932	0.914	0.601
5000	7.33	5.53	14.39	0.899	0.859	0.620
10,000	7.49	10.36	14.55	0.907	0.954	0.669
25,000	8.80	31.02	15.76	0.904	0.872	0.698
50,000	11.82	73.24	19.28	0.906	0.907	0.704

At 50,000 samples, FastSymbolicGP was approximately 6.20 times faster than PySR and 1.63 times faster than gplearn, while maintaining mean $R^2 = 0.9063$, essentially equal to PySR's 0.9070 and well above gplearn's 0.7038. The runtime-scaling behavior across all evaluated sample sizes is shown in Figure 7. PySR was fastest at the smallest sample sizes, but its runtime crossed FastSymbolicGP between 5000 and 10,000 samples and then increased steeply. This crossover is important for the intended use case: the Numba-compiled postfix evaluator has a larger fixed setup cost than some alternatives, but its per-sample evaluation cost scales favorably as the dataset grows. The measured Python-level peak memory at 50,000 samples was higher for FastSymbolicGP than for the other methods; therefore, the observed advantage should be interpreted specifically as a runtime-scaling result rather than as simultaneous superiority in every resource measure.

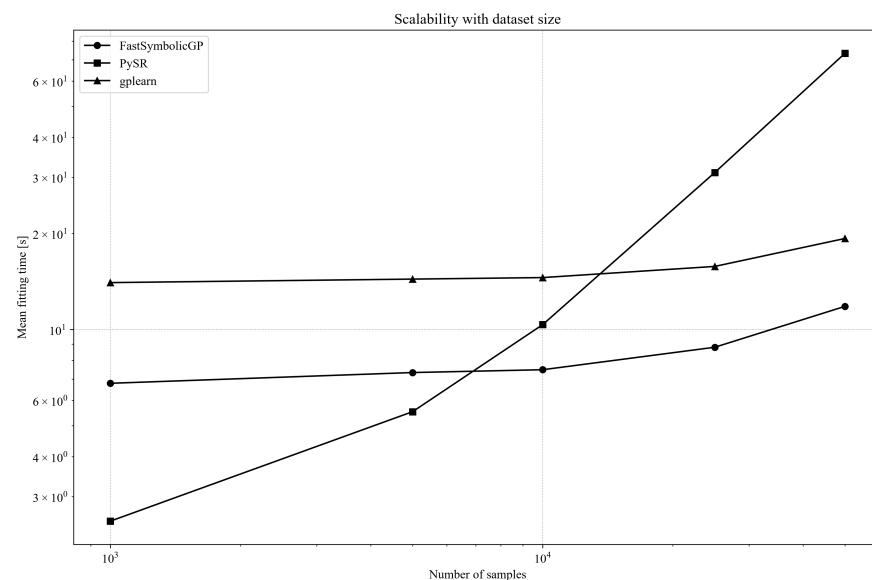


Figure 7. Mean fitting time as the number of samples increases. Values are averaged over four feature dimensionalities.

3.7. Parameter Sensitivity

The sensitivity experiment evaluated population sizes of 100, 250, 500, and 1000 together with 5, 10, 20, and 50 generations. Figure 8 shows the regression results. For Friedman-1 regression, increasing the number of generations produced the clearest

and most consistent improvement. The highest mean R^2 , 0.8502, was obtained with population size 500 and 50 generations, but this configuration required 85.81 s. The common medium-budget setting of 250 individuals and 20 generations obtained $R^2 = 0.8040$ in 14.59 s, making it approximately 5.9 times faster while sacrificing 0.046 in absolute R^2 .

For binary classification, the highest mean balanced accuracy was 0.9619 at population size 100 and 50 generations, whereas the 250-by-20 setting obtained 0.9558. For multiclass classification, the best mean macro-F1 was 0.9866 at 100 individuals and 20 generations; larger populations did not improve this result and often increased complexity and runtime. Consequently, no single population–generation combination dominated all task types. The selected common configuration should therefore be understood as a reproducible middle-budget compromise used to avoid dataset-specific tuning, not as a claim of universal optimality. The sensitivity results also show diminishing returns and, for some classification settings, mild degradation when the population becomes unnecessarily large.

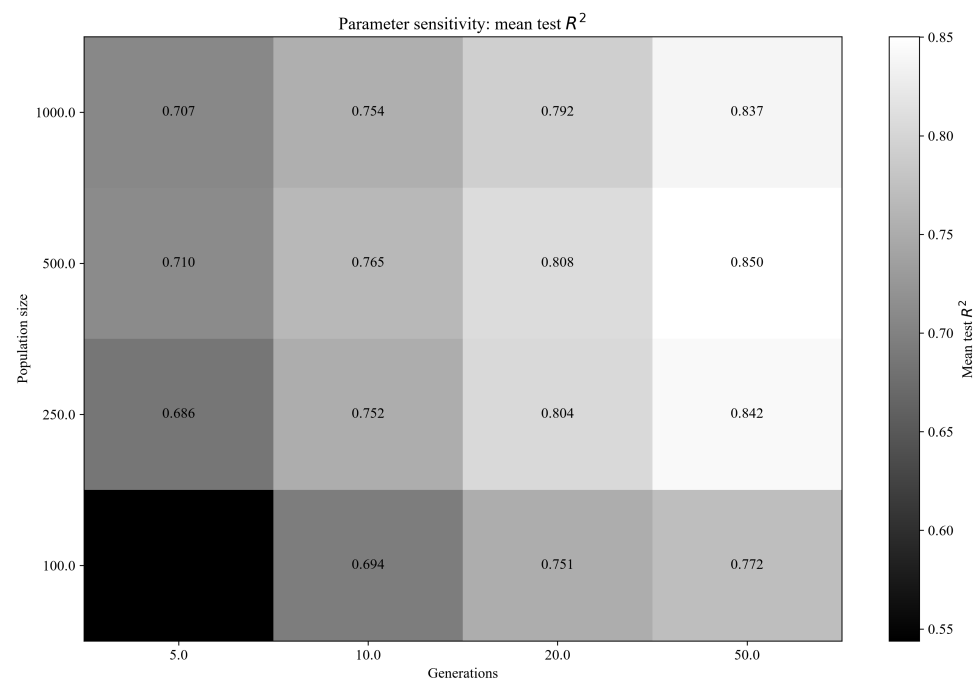


Figure 8. Parameter sensitivity of FastSymbolicGP on Friedman-1 regression. Each cell reports mean test R^2 over five seeds.

3.8. Ablation Study

Table 17 reports the Friedman-1 ablation study. Removing crossover produced the largest deterioration relative to the reference configuration, decreasing mean R^2 from 0.7242 to 0.7018. This reduction was accompanied by shorter expressions and lower runtime, indicating that crossover contributed useful structural exploration rather than only increasing tree size. Disabling subtree mutation increased mean R^2 to 0.7580 in this particular experiment, but also increased the mean complexity score from 56.8 to 66.8. The result indicates a dataset-specific predictive–complexity trade-off rather than evidence that subtree mutation is generally harmful.

Table 17. Ablation results on Friedman-1 over five seeds.

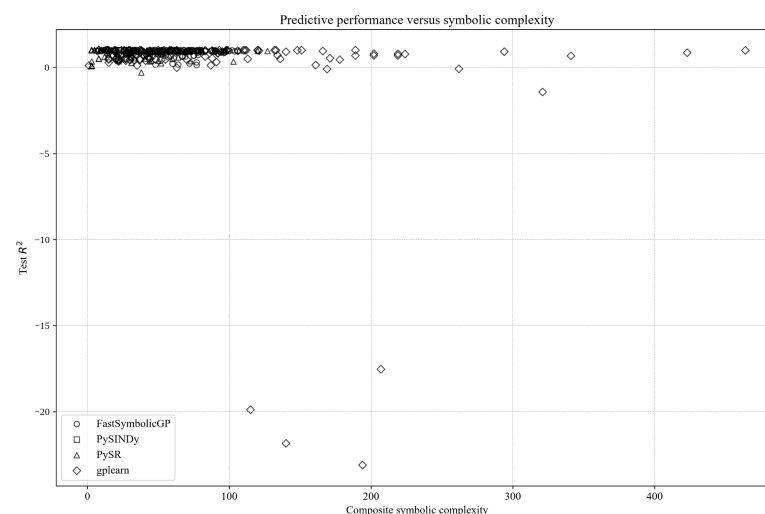
Configuration	Test R^2	RMSE	Fit Time (s)	Complexity
Reference configuration	0.7242 ± 0.0298	2.5854	13.3612	56.8
No subtree mutation	0.7580 ± 0.0312	2.4200	13.3996	66.8
No crossover	0.7018 ± 0.0572	2.6787	11.7864	46.6
No adaptive parsimony	0.7242 ± 0.0298	2.5854	13.4326	56.8
Island evolution	0.7254 ± 0.0337	2.5796	13.4326	59.4

Disabling adaptive parsimony did not change the final predictive metrics in this small five-seed experiment, although the mechanism may still affect population composition and search behavior before final model selection. Island evolution produced performance similar to the reference configuration. Because the ablation was intentionally restricted to a single controlled regression problem, these findings are interpreted as implementation-level mechanism checks rather than universal rankings of genetic operators.

3.9. Interpretability and Performance–Complexity Trade-Off

The interpretability analysis used expression length, estimated depth, number of variables, number of nonlinear operators, tree-node count, DAG-node count, and a composite symbolic-complexity score. Figure 9 plots predictive performance against the composite score for the regression and dynamical system experiments. Most FastSymbolicGP solutions occupy the high-performance, low-to-moderate-complexity region. In particular, the method produced substantially simpler expressions than gplearn in scientific regression, dynamical system identification, and multiclass classification, although PySR was frequently simpler.

The stored equation pools also enabled an explicit post hoc simplicity selection. For each of 93 benchmark–method–configuration groups, the best predictive model was compared with the simplest model, whose validation loss was within 1% of the best value. The simpler model reduced the complexity score in 31 of the 93 groups (33.3%). Among the groups where a reduction was possible, mean complexity decreased by 28.6%, while the mean loss in the reported predictive metric was only 0.00063 and the maximum loss was 0.00832. In the remaining groups, the best predictive model was already the simplest model satisfying the tolerance. These results show that the Pareto archive is not merely diagnostic: it can be used to obtain materially simpler expressions with negligible predictive degradation.

**Figure 9.** Predictive performance versus composite symbolic complexity for the regression and dynamical system experiments.

3.10. Representative Equations and Cross-Method Comparison

The aggregate performance results describe the general predictive and computational behavior of the evaluated methods, but they do not show whether the resulting symbolic models are scientifically meaningful or practically interpretable. Therefore, representative equations were selected from the combined pool of 5619 stored candidate expressions and are compared in Tables 18 and 19.

The best predictive equation was selected for each method unless a lower-complexity equation achieved a test score within 1% of the method-specific best result. In the latter case, the simpler equation was preferred. The reported metric consequently corresponds to the held-out test performance of the exact run that generated the displayed equation rather than to the mean performance across repeated runs. Algebraically redundant expressions were simplified for presentation without changing their predictions over the benchmark domain; however, the reported complexity score corresponds to the original expression produced by the method. The protected square-root and logarithm operators are denoted by $\text{sqrt } p!(\cdot)$ and $\log p!(\cdot)$, respectively.

For concrete compressive strength, the best FastSymbolicGP and gplearn equations obtained nearly identical test performance, with $R^2 = 0.69475$ and $R^2 = 0.69295$, respectively. However, the FastSymbolicGP expression had a complexity score of 55, compared with 219 for gplearn. Thus, the FastSymbolicGP equation provides a substantially better accuracy–complexity trade-off. This result was not limited to a single favorable seed: across the five repeated runs, the mean R^2 values were 0.6694 for FastSymbolicGP, 0.5249 for PySR, and 0.3996 for gplearn.

The difference was more pronounced for the airfoil self-noise dataset. The displayed FastSymbolicGP equation achieved $R^2 = 0.67856$, compared with $R^2 = 0.38941$ for the best PySR equation. None of the five gplearn runs produced a positive test R^2 ; its strongest result was $R^2 = -1.42853$ with a complexity score of 321. The FastSymbolicGP expression is also scientifically informative because it combines the principal aeroacoustic variables, including frequency, chord length, free-stream velocity, angle of attack, and boundary-layer displacement thickness.

For the Parkinson’s dataset, all methods obtained lower R^2 values, reflecting the difficulty of predicting the total UPDRS score from the available voice measurements. Nevertheless, FastSymbolicGP produced the strongest displayed equation, with $R^2 = 0.29563$, compared with 0.24215 for PySR and 0.13684 for gplearn. The gplearn tree is particularly illustrative of why raw tree size and algebraic simplification should both be considered: despite its complexity score of 161, the expression simplifies to a function of age alone.

To complement the aggregate predictive and complexity results, Table 19 presents representative equations obtained for physical law recovery, dynamical system identification, binary classification, and multiclass one-versus-rest classification. The comparison includes equations produced by FastSymbolicGP and the relevant baseline methods, together with their test performance, expression complexity, and the principal reason for selecting each model. For the multiclass datasets, the complete set of class-specific equations is reported because the final prediction depends on the joint one-versus-rest system rather than on an individual symbolic model.

Table 18. Representative equations obtained for real-world scientific regression datasets. The complexity column reports the text-based complexity score of the original generated expression.

Dataset	Scientific Field	Method	Representative Equation	Test Metric	Complexity	Reason for Selection
Concrete compressive strength	Civil engineering	Fast Symbolic GP	$\hat{f}_c = 276.8208 \frac{\sqrt{p!(C \log p!(A)) + \sqrt{p!(S + W)}}}{W} - 32.7410$	$R^2 = 0.69475$	55	Highest test R^2 with a compact four-variable relationship.
Concrete compressive strength	Civil engineering	PySR	$\hat{f}_c = 4.6007 \log p! \left(\left \begin{array}{c} A(C - 15.7204) \\ \times (-C + 0.4153 \sqrt{p!(C(F_a - 1.9266))}) \\ + 8.1949 \\ \times (AS + C - 12.1658) \end{array} \right \right)$ $- 54.6148$ $\hat{f}_c = \sqrt{p!} \left(\frac{C(C + P)}{W} + \frac{C(C/W + S \log p!(A))}{W} + 2PA \right)$ $+ \left \frac{C(PA + P + A)}{W} + (C + P) \right $ $\left \sqrt{p!} \left(\frac{C(PA + P + A)}{W} + (C + P) \cos(F_a) \right) \log p!(A) \right ^{\frac{1}{2}}$	$R^2 = 0.57813$	64	Best PySR model, but less accurate and slightly more complex than Fast Symbolic GP.
Concrete compressive strength	Civil engineering	gplearn	$\hat{f}_c = 132.9319 - 2.8161 \sqrt{p!} \left((\log p!(\delta) - c) f c \delta + \frac{\alpha + \sqrt{p!(U)}}{f \delta} \right)$	$R^2 = 0.69295$	219	Nearly the same predictive score as FastSymbolicGP, but approximately four times more complex.
Airfoil self-noise	Aeroacoustics	Fast Symbolic GP	$\hat{y} = 110.6131$ $+ \left[\frac{U(231.7943 - U)(39.9116\alpha + 136.0050U)}{3.6230U\delta(f - 63.2538) + 214.6883} \right]^{1/4}$	$R^2 = 0.67856$	75	Best predictive result; combines frequency, geometry, flow velocity, angle, and boundary-layer thickness.
Airfoil self-noise	Aeroacoustics	PySR	$\hat{y} = 4.7778[\cos(s + 2a) + \cos(s + a) - \sqrt{p!(s + Q_{11}) + \sqrt{p!(a)}}] - 6.9446$	$R^2 = 0.38941$	50	Simpler than the FastSymbolicGP equation, but with substantially lower predictive performance.
Parkinson's total UPDRS	Biomedicine	Fast Symbolic GP	$\hat{y} = \left 31.7359 \cos(\sin(a + s) \times \sqrt{p!(\cos(\sin(1.2939a + \cos S_h)))) \right) + 3.0694 \right $	$R^2 = 0.29563$	77	Highest test R^2 ; depends only on age, sex, and Shimmer APQ11.
Parkinson's total UPDRS	Biomedicine	PySR	$\hat{y} = 3 \sqrt{p!(a)} + \sqrt{p!(3 \sqrt{p!(a)})} + \cos(a) + \cos(3 \sqrt{p!(a)} + \sqrt{p!(3 \sqrt{p!(a)})})$	$R^2 = 0.24215$	52	Lower complexity, but lower predictive performance and a more strongly nested nonlinear structure.
Parkinson's total UPDRS	Biomedicine	gplearn		$R^2 = 0.13684$	161	The original tree is highly redundant and reduces to an age-only model with the lowest predictive score.

C: cement; S: blast-furnace slag; F_a : fly ash; W: water; P: superplasticizer; A: concrete age; f: frequency; α : angle of attack; c: chord length; U: free-stream velocity; δ : suction-side displacement thickness; a: patient age; s: sex; Q_{11} : Shimmer APQ11; and S_h : Shimmer.

Table 19. Representative physical law, dynamical system, binary classification, and multiclass equations. For multiclass one-versus-rest models, all class-specific equations are reported. The test metric and total complexity refer to the complete one-versus-rest system.

Dataset or System	Scientific Field	Method	Representative Equation	Test Metric	Complexity	Reason for Selection
Pendulum period	Classical mechanics	Fast Symbolic GP	$\hat{T} = 6.2854\sqrt{\frac{L}{g}} - 0.00156$	$R^2 = 1.00000$	26	Simplest equation within 1% of the best result; dimensionally valid and closest to the theoretical law.
Pendulum period	Classical mechanics	PySR	$\hat{T} = \frac{\sqrt{ L(0.5484L^2 + 36.3368g) + g - 1.2096 }}{g}$	$R^2 = 0.99558$	33	Simplest PySR equation within 1% of its best score, but dimensionally invalid.
Pendulum period	Classical mechanics	gplearn	$\hat{T} = 1.4494\sqrt{L} \left[\sqrt{g} + 1.4494 \right. \\ \times \left. \sqrt{\left \sqrt{g} + \sqrt{2.1008L \sin \log g - \sin(\cos g) + \cos g} \right } \right]^{1/4} \\ \times \sqrt{\sin \log g }$	$R^2 = 0.99248$	148	Best gplearn result, but substantially more complex and dimensionally invalid.
Lorenz $\dot{x}$	Nonlinear dynamics	Fast Symbolic GP	$\dot{x} = -10(x - y) + 2.22 \times 10^{-16} \equiv 10(y - x)$	$R^2 = 1.00000$	20	Exact recovery of the reference equation from a three-node raw symbolic tree.
Lorenz $\dot{x}$	Nonlinear dynamics	PySR	$\dot{x} = 10(y - x)$	$R^2 = 1.00000$	10	Exact recovery and the lowest complexity among the general symbolic regression methods.
Lorenz $\dot{x}$	Nonlinear dynamics	gplearn	$\dot{x} = 10(y - x)$ after algebraic simplification	$R^2 = 1.00000$	120	Exact prediction, but the original tree repeats the term $(y - x)$ ten times.
Lorenz system	Nonlinear dynamics	PySINDy	$\begin{aligned} \dot{x} &= -10x + 10y, \\ \dot{y} &= 28x - y - xz, \\ \dot{z} &= xy - \frac{8}{3}z \end{aligned}$	$R^2 = 1.00000$	22	Exact recovery of the complete sparse dynamical system and strongest agreement with theory.
Breast cancer	Biomedical classification	Fast Symbolic GP	$p_1 = \sigma \left[-2.7983(2r_w + t_w + p_w + p_e \right. \\ \left. + a_e + \sin a_e + s_w) + 0.8842 \right],$ $\hat{y} = \begin{cases} 1, & p_1 \geq 0.5, \\ 0, & p_1 < 0.5 \end{cases}$	BAcc = 0.98115	75	Best predictive equation; achieves the same balanced accuracy as gplearn with lower complexity.

Table 19. Cont.

Dataset or System	Scientific Field	Method	Representative Equation	Test Metric	Complexity	Reason for Selection
Breast cancer	Biomedical classification	gplearn	$p_1 = \sigma[a_e \text{sym}_w - 2t_w - 6p_w - 2.951A_w - 2s_w - 2c_m],$ $\hat{y} = \begin{cases} 1, & p_1 \geq 0.5, \\ 0, & p_1 < 0.5 \end{cases}$ $p_0 = \sigma[-37.2456(SL + \cos(PL + PW)) - 30.3015],$	B _{Acc} = 0.98115	121	Equal predictive performance, but 61% higher complexity than FastSymbolicGP.
Iris OVR system	Biological classification	Fast Symbolic GP OVR	$p_1 = \sigma[12.6125(\text{sqrt } p!(\cos(PL + PW)) + \cos(SW PW) + \cos(PW^3)) - 30.7452],$ $p_2 = \sigma[1.3682(PL + \text{sqrt } p!(\text{sqrt } p!(SL)) + PW) \times (2PL + 3PW - SW) - 8.4845],$ $\hat{y} = \arg \max_{k \in \{0,1,2\}} p_k.$ $s_0 = 57.5938SW - 406.7157PL + 3.8220 \log p!(PL) - 177.8630,$	F1 _{macro} = 1.000	175 total; 44 total nodes; 6, 18, 20	Perfect classification using all three class-specific models. The complete OVR system is compact and contains only 44 raw tree nodes.
Iris OVR system	Biological classification	gplearn OVR	$s_1 = \log p! \left(\frac{\left \frac{\log p!(PW)/(PL PW)}{-1.446 PL(SW - PW)} \right }{PL PW} \log p!(PL) \right),$ $\ell_1 = \log p!(PL(PW - SW)), \quad \ell_2 = \log p!(PL PW),$ $\ell_3 = \log p!(PW), \quad \ell_4 = \log p!(\sin(PW)),$ $s_2 = 4PL + 3PW + 2\ell_1 - \ell_1 - \ell_2 + \ell_3 - \ell_3 + \ell_4,$ $\hat{y} = \arg \max_{k \in \{0,1,2\}} s_k.$	F1 _{macro} = 1.000	325 total; 113 syntax nodes; 37, 24, 52	Perfect classification, but the complete OVR system is substantially more complex than the corresponding FastSymbolicGP system.

Table 19. Cont.

Dataset or System	Scientific Field	Method	Representative Equation	Test Metric	Complexity	Reason for Selection
Wine OVR system	Chemical classification	Fast Symbolic GP OVR	$p_0 = \sigma \Big[24.7297 \big(\sin(A_{\text{alk}}) + P + D + A - A_{\text{alk}} \big) - 13.7867 \Big],$	$F1_{\text{macro}} = 1.000$	138 total; 32 total nodes; 10, 13, 9	Perfect classification with three compact class-specific equations. This is the smallest selected multiclass OVR system among the reported examples.
			$p_1 = \sigma \Big[-3.9630 \big(\cos(D)(FP + P) - \cos(D) + CI \big) - 3.3721 \Big],$			
			$p_2 = \sigma \Big[-7.5211 \big(D + F - A_{\text{sh}} - 2CI \big) - 9.8076 \Big],$			
			$\hat{y} = \arg \max_{k \in \{0,1,2\}} p_k.$			

Notes: $\sigma(u) = 1/(1 + \exp(-u))$. The operators $\sqrt{p!}(\cdot)$ and $\log p!(\cdot)$ denote the protected square-root and protected logarithm operators, respectively. For breast-cancer classification, r_w , t_w , p_w , A_w , s_w , sym_w , and c_m denote worst radius, worst texture, worst perimeter, worst area, worst smoothness, worst symmetry, and mean concavity, respectively; p_e and a_e denote perimeter error and area error. For Iris, SL , SW , PL , and PW denote sepal length, sepal width, petal length, and petal width, respectively. Classes 0, 1, and 2 correspond to Iris setosa, Iris versicolor, and Iris virginica. For the Wine dataset, A denotes alcohol, A_{sh} denotes ash, A_{alk} denotes alkalinity of ash, F denotes flavanoids, CI denotes color intensity, D denotes the OD280/OD315 ratio of diluted wines, and P denotes proline. For every OVR system, the final prediction is obtained by selecting the class with the largest class-specific probability or decision score. The class-specific node counts are reported in class order.

Table 19 shows that FastSymbolicGP generally achieves a favorable balance between predictive performance, symbolic compactness, and scientific interpretability. For the pendulum-period problem, FastSymbolicGP recovers an equation that closely matches the theoretical relationship ($T = 2\pi\sqrt{L/g}$), achieves an essentially perfect test (R^2), and remains dimensionally valid. In contrast, the PySR and gplearn expressions are less accurate, more complicated, and dimensionally inconsistent. This result demonstrates that predictive accuracy alone is insufficient when symbolic models are intended to represent physical relationships.

For the Lorenz system, all evaluated methods recover the ($\dot{x} = 10(y - x)$) relationship after simplification. PySR produces the lowest-complexity representation of this individual equation, whereas FastSymbolicGP achieves exact prediction with only a negligible numerical residual. PySINDy provides the strongest overall scientific result because it exactly identifies the complete sparse system, including the equations for ($\dot{x}$), ($\dot{y}$), and ($\dot{z}$). The gplearn model also predicts the derivative exactly, but its original symbolic tree is substantially larger because the same component is repeatedly represented.

The breast-cancer example demonstrates that FastSymbolicGP can produce a compact nonlinear decision function without sacrificing classification performance. As reported in Table 19, FastSymbolicGP and gplearn obtain the same balanced accuracy of (0.98115). However, the FastSymbolicGP expression has a complexity of 75, compared with 121 for gplearn, representing an approximately 38% reduction relative to the gplearn model. The selected FastSymbolicGP equation therefore provides a better accuracy–complexity trade-off while retaining an explicit probability-based decision rule.

The multiclass results further illustrate the compactness of the FastSymbolicGP one-versus-rest formulation. On the Iris dataset, both FastSymbolicGP and gplearn achieve a macro-averaged F1-score of (1.000). Nevertheless, the FastSymbolicGP system contains only 44 raw tree nodes and has a total complexity of 175, whereas the gplearn system contains 113 syntax nodes and has a total complexity of 325. Thus, the FastSymbolicGP representation is considerably smaller while preserving perfect classification performance. Similarly, the Wine dataset is classified perfectly using three compact class-specific equations with only 32 total raw tree nodes. Among the reported multiclass examples, this is the smallest selected one-versus-rest system.

Overall, the equations in Table 19 confirm that FastSymbolicGP is capable of producing models that are not only accurate but also comparatively compact and interpretable. Although specialized methods such as PySINDy remain particularly effective for sparse dynamical system discovery, FastSymbolicGP provides a unified framework that performs well across physical law regression, nonlinear dynamics, binary classification, and multiclass classification.

3.11. Summary of Statistical Comparisons

Table 20 collects the principal paired comparisons. The strongest and most consistent advantages of FastSymbolicGP were observed against gplearn in binary classification, scientific regression, dynamical system identification, and runtime. Comparisons against PySR were more nuanced: FastSymbolicGP achieved higher scientific regression accuracy, whereas PySR was generally faster and simpler on the smaller scientific and dynamical system problems. In multiclass classification, heterogeneity across Iris, Wine, and Digits prevented the average predictive advantage from reaching significance, although the runtime and complexity reductions were consistent.

Table 20. Selected paired Wilcoxon comparisons. Δ is method A minus method B, p_H is the Holm-adjusted p -value within the corresponding benchmark family, and r_{rb} is the rank-biserial effect size.

Domain	Method A	Method B	Metric	Δ	p_H	r_{rb}
Scientific regression	FastSymbolicGP	PySR	R^2	0.1208	4.20×10^{-4}	0.895
Scientific regression	FastSymbolicGP	gplearn	R^2	4.5019	2.90×10^{-5}	0.981
Dynamical systems	FastSymbolicGP	gplearn	R^2	0.0738	0.0057	0.775
Dynamical systems	FastSymbolicGP	PySR	R^2	0.0761	0.6299	0.193
Binary, fixed threshold	FastSymbolicGP	gplearn	Balanced accuracy	0.1050	1.66×10^{-5}	0.880
Binary, optimized threshold	FastSymbolicGP	gplearn	Balanced accuracy	0.0718	2.39×10^{-5}	0.858
Multiclass	FastSymbolicGP OVR	gplearn OVR	Macro-F1	0.0893	0.6586	0.457
Multiclass	FastSymbolicGP OVR	gplearn OVR	Fit time (s)	−56.65	6.10×10^{-4}	−1.000
Multiclass	FastSymbolicGP OVR	Shared softmax	Complexity	−148.8	0.0226	−0.850

Overall, the results support three conclusions. First, FastSymbolicGP consistently improves on the conventional gplearn baseline in predictive performance or computational efficiency across the principal regression and classification experiments. Second, comparison with PySR reveals a genuine trade-off rather than uniform dominance: FastSymbolicGP is particularly competitive in predictive accuracy and large-sample runtime scaling, whereas PySR is often faster and simpler on small problems. Third, the dimensional-analysis, threshold-selection, Pareto-selection, sensitivity, and ablation experiments demonstrate that reliable symbolic modeling depends not only on the evolutionary search itself but also on representation, operating-point selection, and explicit control of model complexity.

4. Discussion

The experimental results demonstrate that FastSymbolicGP provides a favorable balance among predictive performance, computational efficiency, task coverage, and symbolic model complexity. The library was not uniformly superior to every competing method under every evaluation criterion. Instead, its principal strength was the ability to obtain competitive or improved predictive performance while retaining a lightweight Python implementation, explicit symbolic outputs, native classification support, and favorable runtime scaling. This positioning is important because symbolic learning systems differ substantially in their intended use, internal execution engines, model representations, and optimization strategies. Consequently, the present results should be interpreted as evidence that FastSymbolicGP is a practical general-purpose symbolic learning framework rather than as evidence that one implementation universally dominates all alternatives.

4.1. Predictive Performance and Comparison with Existing Tools

The real-world scientific regression results provide strong support for the predictive capability of FastSymbolicGP. The library achieved the highest mean test R^2 on all four investigated datasets, covering civil engineering, building physics, aeroacoustics, and biomedicine. Its overall mean R^2 was 0.5981, compared with 0.4773 for PySR. The paired difference was statistically significant and was associated with a large rank-biserial effect size. The comparison with gplearn was even more favorable, although the very poor gplearn result on Airfoil Self-Noise increased the magnitude of the overall difference.

The strong result on Energy-Efficiency Heating Load indicates that the method can identify compact nonlinear relationships when the target is governed by comparatively structured physical and geometrical variables. By contrast, the mean R^2 of 0.1910 on Parkinson's Total UPDRS was substantially lower than the results obtained on the engineering datasets. This result should not be interpreted as a failure of the symbolic approach alone. Biomedical outcomes such as UPDRS scores may depend on subject-specific, temporal, clinical, and latent factors that are not fully represented by the available predictors. The result

therefore illustrates an important limitation of explicit low-complexity symbolic models: when the measured variables contain only a modest proportion of the information required to explain the target, a compact analytical equation cannot be expected to produce high predictive accuracy.

The scientific regression comparison with PySR was more nuanced than the comparison with gplearn. PySR frequently produced shorter equations and required less time on the smaller regression datasets, whereas FastSymbolicGP obtained higher average predictive performance. This trade-off is consistent with the different design goals of the two systems. PySR uses a highly optimized Julia backend, multi-population search, numerical constant optimization, and specialized symbolic regression mechanisms [9]. FastSymbolicGP instead emphasizes a lightweight Python installation, a unified estimator interface, classification support, and configurable genetic programming execution. The results therefore do not suggest that PySR is generally inferior. Rather, they indicate that FastSymbolicGP is competitive with a modern high-performance symbolic regression system while also addressing a broader set of supervised-learning tasks.

The comparison with gplearn was more consistent across the investigated domains. FastSymbolicGP generally achieved higher predictive performance, shorter fitting times, and lower symbolic complexity. Both libraries expose estimator-style Python interfaces and are based on tree-oriented genetic programming, making this comparison particularly relevant. The observed differences suggest that implementation choices such as postfix Numba evaluation, validation-aware selection, bounded expression growth, adaptive parsimony, and final complexity-aware model selection can materially improve practical GP performance.

Nevertheless, the comparison should not be interpreted as a definitive ranking of the libraries under every possible configuration. The experiments used a common moderate search profile rather than an extensive independent hyperparameter optimization for each framework. Different configurations of gplearn or PySR, including alternative hyperparameters, primitive sets, numerical constant optimization settings, stopping criteria, and computational budgets, could produce different comparative outcomes. The present comparison therefore evaluates how the methods behave under transparent and broadly matched experimental settings rather than determining the maximum achievable performance of each software package.

4.2. Scientific Equation Discovery

The dynamical system experiments show that FastSymbolicGP can recover meaningful governing relationships in addition to fitting static datasets. The library obtained a pooled mean R^2 of 0.9428 across the evaluated state equations and significantly outperformed gplearn. The comparison with PySR was not statistically significant after Holm correction, indicating that the two general-purpose symbolic regression systems achieved broadly comparable predictive performance on these systems.

An especially important qualitative result was the exact recovery of the Lorenz relation

$$\widehat{\frac{dx}{dt}} = -10(x - y) = 10(y - x). \quad (66)$$

This expression is algebraically equivalent to the known governing equation and is directly interpretable. Such a result is more informative than a high R^2 alone because it demonstrates that the algorithm can recover a scientifically recognizable structural relationship.

The Lorenz y equation was considerably more difficult than the x and z equations. Its structure contains several interacting terms,

$$\frac{dy}{dt} = x(28 - z) - y, \quad (67)$$

including a multiplicative interaction and a relatively large numerical coefficient. The lower accuracy obtained for this state suggests that the moderate evolutionary budget was not always sufficient to recover all interacting terms reliably. Improved constant optimization, larger search budgets, typed primitives, or a domain-informed function set may improve recovery of such equations.

PySINDy exactly recovered both investigated systems and required substantially less computational time. This result is expected because PySINDy was supplied with a polynomial library containing the true governing terms and solves a sparse coefficient-selection problem rather than a general free-form symbolic search [11]. The comparison therefore illustrates a distinction between specialized and general-purpose equation-discovery methods. When the appropriate candidate library is known in advance, sparse system identification can be more efficient and reliable. Genetic programming becomes particularly relevant when the required operators, interactions, or equation structures are not known sufficiently well to define a compact predetermined library.

The dynamical system benchmark also used exact derivatives evaluated from the known right-hand sides. Real scientific data would normally involve measurement noise, incomplete state observations, irregular sampling, and numerical derivative estimation. Consequently, the reported results represent controlled governing-equation recovery rather than a complete simulation of experimental system identification. Future evaluation should include noisy trajectories, finite-difference or smoothed derivative estimates, partially observed systems, and out-of-trajectory validation.

4.3. Physical Law Recovery and the Importance of Dimensional Consistency

The physical law experiments produced one of the most important findings of the study. FastSymbolicGP obtained nearly perfect predictive accuracy on the controlled physical relationships, but only 24% of the expressions recovered from raw dimensional inputs passed the strict dimensional-consistency check. Thus, predictive accuracy alone did not guarantee physical validity.

This result directly supports the methodological distinction between symbolic interpretability and scientific correctness. A symbolic expression is transparent because its structure can be inspected, but transparency does not imply that the structure respects physical units, conservation laws, boundary conditions, or other domain constraints. An equation may fit a limited numerical range extremely well while combining incompatible quantities or applying nonlinear functions to dimensional arguments.

Nondimensionalization substantially improved the results. The dimensional-validity rate increased from 24% to 96%, while predictive accuracy remained effectively unchanged. Nondimensionalization also reduced mean fitting time by approximately 54.1% and reduced mean expression complexity by approximately 47.5%. These improvements indicate that nondimensionalization is beneficial for more than post hoc physical interpretation. It can also simplify the numerical search space by reducing scale disparities and allowing the evolutionary process to discover more compact relationships.

The recovered pendulum expression,

$$\hat{T} = 6.2854\sqrt{\frac{L}{g}} - 0.00156, \quad (68)$$

provides a representative example. The leading coefficient is very close to the theoretical value 2π , while the intercept is negligible. This equation is predictive, dimensionally meaningful, structurally compact, and physically recognizable. It therefore demonstrates the type of output that motivates symbolic regression in scientific applications.

However, the current library does not enforce dimensional constraints during initialization, crossover, mutation, or final selection. The dimensional analysis performed in this study was post hoc. FastSymbolicGP should therefore not be described as a physics-constrained or unit-aware symbolic regression system. In applications involving dimensional measurements, users should nondimensionalize the variables, construct unit-consistent derived features, restrict the primitive set, or manually verify the recovered equations. Automatic dimensional typing represents an important direction for future development and could substantially improve both search efficiency and scientific reliability [15,16].

4.4. Binary Classification and Operating-Point Selection

The binary classification experiments demonstrate that an evolved symbolic score is only one component of a complete classifier. Class weighting, probability mapping, and threshold selection materially affected the final operating characteristics, particularly under severe class imbalance.

With a fixed threshold of 0.5, FastSymbolicGP achieved a mean balanced accuracy of 0.8507 across all binary benchmark cases, compared with 0.7458 for gplearn. Validation-based MCC threshold optimization increased the FastSymbolicGP mean balanced accuracy to 0.8607. The improvement was most visible when the minority class represented only 5% of the samples, where threshold optimization increased balanced accuracy from 0.713 to 0.758.

The balanced FastSymbolicGP configuration increased the overall mean balanced accuracy to 0.8819. At the 5% minority fraction, it achieved a mean balanced accuracy of 0.827, compared with 0.713 for the unweighted fixed-threshold configuration and 0.530 for fixed-threshold gplearn. This result confirms that class weighting and validation-based threshold selection can substantially improve minority-class detection.

The appropriate configuration nevertheless depended on the target objective. Class balancing increased balanced accuracy but reduced overall F1 and MCC relative to the unweighted MCC-optimized configuration. At a balanced 50:50 class distribution, class weighting provided no advantage. These findings show that threshold optimization and class weighting should not be activated mechanically. The selected method should reflect the relative costs of false positives and false negatives, the expected deployment prevalence, and the metric that is most relevant to the application.

A further implication is that the symbolic equation alone does not fully define the deployed classifier. A complete report should include the evolved expression, affine logistic scaling, probability transformation, class weighting, and final threshold. FastSymbolicGP retains these elements explicitly, which improves reproducibility and allows the operating point to be changed without evolving a new symbolic structure.

The benchmark did not apply an additional probability-calibration stage. The sigmoid transformation provides probability-like outputs, but those probabilities are not necessarily perfectly calibrated. Future experiments should evaluate Platt scaling, isotonic regression, and beta calibration using calibration-specific metrics such as the Brier score, expected calibration error, and calibration curves.

4.5. Multiclass Symbolic Classification

The multiclass experiments confirm that native multiclass symbolic learning is feasible within the proposed framework, but they also demonstrate that no single output strategy was optimal for every dataset.

The shared-softmax strategy achieved the highest mean macro-F1 on Iris and Wine, reaching 0.980 and 0.990, respectively. These datasets contain only three classes and relatively low-dimensional feature spaces. In such settings, a common symbolic backbone can identify shared discriminative relationships and reuse them across class scores.

On the ten-class Digits dataset, one-versus-rest FastSymbolicGP performed substantially better, obtaining macro-F1 = 0.890, compared with 0.805 for shared softmax. This difference suggests that a limited shared symbolic representation may become restrictive when classes have heterogeneous decision boundaries. Independent one-versus-rest classifiers allow each class to evolve its own expression and may therefore better represent complex multiclass structure.

Shared softmax provided a 3.53-fold reduction in prediction time, but its mean symbolic complexity was approximately 42.6% greater than that of the one-versus-rest approach. This result initially appears counterintuitive because shared representations are often expected to reduce redundancy. However, the symbolic backbone and the complete set of softmax coefficients jointly define the multiclass model. A larger common component set may reduce execution overhead while still increasing the reported structural complexity.

The results therefore support a task-dependent recommendation. Shared softmax is attractive for problems with relatively few classes, common discriminative structure, or strict inference-latency requirements. One-versus-rest is preferable when classes require distinct symbolic boundaries, when class-specific equations are scientifically useful, or when the number of classes is larger.

The comparison with gplearn showed substantial runtime and complexity advantages for FastSymbolicGP, but the average predictive difference was not statistically significant after Holm correction. This was caused by strong dataset heterogeneity: gplearn was competitive on Iris and Wine but performed poorly on Digits. Broader evaluation on multiclass benchmark collections is required before making general claims about average predictive superiority.

4.6. Computational Efficiency and Scalability

The scalability analysis provides direct evidence that the Numba-based postfix evaluator is advantageous when the number of observations becomes large. FastSymbolicGP fitting time increased from 6.79 s at 1000 samples to 11.82 s at 50,000 samples, corresponding to a factor of 1.74. PySR increased from 2.51 to 73.24 s over the same range, while gplearn increased from 14.03 to 19.28 s.

PySR was faster on the smallest datasets, indicating that FastSymbolicGP has a non-negligible fixed compilation and evolutionary setup cost. However, FastSymbolicGP became faster between 5000 and 10,000 samples. At 50,000 samples, it was approximately 6.20 times faster than PySR and 1.63 times faster than gplearn, while its mean R^2 was almost identical to that of PySR.

This crossover illustrates the intended role of the Numba execution engine. Compiling symbolic evaluation instructions introduces an initial cost, but repeated vectorized evaluation over larger arrays amortizes this overhead. The approach is therefore particularly suitable for repeated population evaluation on medium-sized and large datasets.

The scalability result must be interpreted within the experimental configuration. Only the postfix Numba evaluator was enabled. DAG execution, subtree caching, evaluation caching, and parallel execution were deliberately disabled to isolate one clearly defined

backend. The results therefore characterize the specified single-thread Numba pathway rather than the maximum performance of the complete library.

The measured Python-level peak memory was also higher for FastSymbolicGP at the largest sample sizes. Consequently, the implementation should not be described as simultaneously optimal in runtime and memory consumption. Future work should analyze resident memory, native allocations, cache behavior, expression-batch size, and the trade-off between temporary vectorized arrays and execution speed.

4.7. Hyperparameter Sensitivity and Evolutionary Search Budget

The sensitivity analysis confirms that genetic programming parameters define a computational trade-off rather than a universally optimal configuration. For Friedman-1 regression, increasing the generation count produced the most consistent improvement. The best mean R^2 of 0.8502 was obtained using 500 individuals and 50 generations, but required 85.81 s. The 250-by-20 configuration achieved $R^2 = 0.8040$ in 14.59 s. The larger configuration was therefore almost six times slower for an absolute R^2 improvement of approximately 0.046.

For binary and multiclass classification, increasing the population size did not consistently improve performance. The best binary balanced accuracy occurred with 100 individuals and 50 generations, while the best multiclass macro-F1 occurred with 100 individuals and 20 generations. Larger populations sometimes increased complexity and runtime without improving the test result.

These findings justify the use of the common 250-by-20 profile as a moderate and reproducible benchmark configuration. It was not the optimal setting for every dataset, but it avoided selecting a separate favorable configuration after observing each test result. This is important for credible benchmarking. Dataset-specific tuning could improve predictive performance, but it would also substantially increase computational cost and complicate comparisons between methods.

For practical use, the sensitivity results suggest that generation count should often be increased before population size, particularly when the initial population already provides sufficient structural diversity. However, this recommendation remains problem-dependent. High-dimensional, discontinuous, or highly multimodal problems may benefit from larger populations, islands, or adaptive population sizing.

4.8. Interpretability and Complexity-Aware Selection

FastSymbolicGP was designed to treat expression complexity as a primary model property rather than as a secondary reporting statistic. The final selection procedure preferred the smallest model whose loss remained within 2% of the best stored loss. The equation-pool analysis provided additional evidence that this approach can produce practically useful simplification.

Across 93 benchmark–method–configuration groups, a simpler model within 1% of the best predictive result was available in 31 groups. In these cases, the mean complexity reduction was 28.6%, while the mean decrease in the reported predictive metric was only 0.00063. The maximum observed decrease was 0.00832. Thus, meaningful reductions in expression size were frequently possible without materially changing predictive performance.

This result supports the use of Pareto archives and tolerance-based final selection. Selecting the numerically best candidate without considering complexity may retain unnecessary operators, constants, and nested subexpressions. In scientific and engineering applications, a slightly less accurate but substantially simpler equation may be easier to validate, communicate, implement, and compare with existing theory.

At the same time, node count, depth, and the composite complexity score are only proxies for interpretability. Two expressions with the same number of nodes may differ greatly in cognitive difficulty. An expression containing several nested logarithmic and trigonometric functions may be harder to understand than a larger polynomial. The number of variables, repeated subexpressions, nonlinear operators, numerical constants, and domain familiarity all influence human interpretation [17,19].

The composite score used in the benchmark was intentionally defined as an implementation-independent descriptive indicator. It should not be presented as a validated universal measure of human comprehensibility. Future work should include human-subject evaluations, task-specific interpretability criteria, minimum-description-length measures, symbolic equivalence reduction, and domain-expert assessment.

4.9. Interpretation of the Ablation Results

The ablation analysis indicates that crossover contributed positively to the Friedman-1 search. Removing crossover reduced the mean R^2 from 0.7242 to 0.7018. The resulting expressions were smaller and the runs were faster, but the predictive deterioration suggests that crossover successfully recombined useful symbolic components.

Disabling subtree mutation increased the mean R^2 to 0.7580, but also increased complexity. This result should not be interpreted as evidence that subtree mutation is unnecessary. Mutation supports exploration and can prevent the population from becoming restricted to combinations of existing material. On this specific dataset and under the selected budget, crossover and point mutation may have provided sufficient diversity, while subtree mutation occasionally disrupted useful structures. Additional ablation experiments across several datasets and larger numbers of seeds are required before drawing general conclusions.

Disabling adaptive parsimony did not change the final metrics in the reported experiment. One explanation is that the maximum depth, maximum node count, hoist mutation, and tolerance-based final selection already provided strong complexity control. The adaptive penalty may therefore have affected intermediate population composition without changing the final selected model.

Island evolution performed similarly to the reference configuration. The selected run used only four islands, infrequent migration, and no parallel execution. Island models may provide greater advantages for larger populations, longer searches, heterogeneous island profiles, or strongly multimodal problems.

The ablation study should therefore be interpreted as a verification that the implemented mechanisms behave plausibly, not as a definitive ranking of evolutionary operators. Its main limitation is that it used one synthetic regression problem and five seeds.

4.10. Practical Positioning of FastSymbolicGP

The results suggest that FastSymbolicGP is particularly suitable for users who require several of the following properties simultaneously:

- Explicit symbolic regression and classification models;
- A compact Python and scikit-learn-compatible interface;
- Native binary and multiclass classification;
- Moderate or large sample sizes;
- Repeated experiments across seeds and parameter configurations;
- Direct extraction of candidate-equation pools;
- Validation-aware and complexity-aware model selection; and
- Source-code export and low-dependency deployment.

The library is not necessarily the preferred choice in every symbolic learning scenario. PySINDy is more appropriate when a reliable sparse candidate library can be specified for a dynamical system. PySR or Operon may be preferable when maximum symbolic regression performance, extensive parallelism, advanced constant optimization, or highly optimized compiled backends are the primary requirements [9,36]. AI Feynman is particularly relevant when the problem satisfies its assumptions regarding physical decomposition and separability [14].

FastSymbolicGP should therefore be positioned as a lightweight and practical complementary tool rather than as a replacement for every symbolic regression system. Its distinctive contribution lies in combining efficient tree-based genetic programming, regression, binary classification, multiclass classification, estimator compatibility, reproducible equation extraction, and complexity-aware output selection within one Python library.

4.11. Limitations

Several limitations should be considered when interpreting the results.

First, the benchmark covered a diverse but finite collection of datasets. Four real-world regression datasets, two real binary datasets, three multiclass datasets, five physical laws, and two dynamical systems are insufficient to establish universal performance. Larger evaluations based on SRBench, PMLB, OpenML, or domain-specific scientific collections would provide stronger evidence of generalization [30,38].

Second, the compared methods did not use identical candidate-evaluation budgets. FastSymbolicGP, gplearn, PySR, and PySINDy employ different population models, constant-optimization procedures, simplification mechanisms, and stopping behavior. Their population and iteration values cannot be converted into a perfectly equivalent number of evaluated symbolic structures. The comparisons therefore reflect practical configurations rather than mathematically identical search budgets.

Third, the experiments were performed on one Windows workstation using single-thread execution. The relative performance could differ on Linux, on processors with different vectorization capabilities, or under parallel execution. PySR and Operon are designed to exploit compiled and parallel execution, while the present study intentionally restricted execution to improve timing reproducibility.

Fourth, only the postfix Numba execution pathway was evaluated for FastSymbolicGP. DAG execution, expression caching, subtree caching, semantic duplicate elimination, adaptive operator probabilities, and parallel island execution were disabled or not fully evaluated. The benchmark therefore does not characterize the maximum performance of every library component.

Fifth, the physical law and dynamical system experiments were controlled. The physical law datasets were generated directly from known analytical equations with a small amount of noise, while exact derivatives were used for the dynamical systems. Real experimental data would introduce measurement error, missing variables, imperfect sensors, nonstationarity, and numerical differentiation error.

Sixth, dimensional consistency was evaluated after symbolic evolution rather than enforced during search. The library can consequently produce numerically accurate but physically invalid equations unless the user performs nondimensionalization or domain-specific validation.

Seventh, the interpretability analysis relied primarily on structural proxies. The composite complexity score has not been validated against human judgments and does not capture every aspect of semantic simplicity, familiarity, or scientific usefulness.

Eighth, the binary experiments evaluated discrimination and threshold-dependent metrics but did not perform a complete probability-calibration analysis. Similarly,

the classification datasets used a fixed development–validation–test procedure rather than nested cross-validation.

Finally, the library was developed and evaluated by its author. Independent reproduction, external users, additional software testing, and community benchmarking are required to establish robustness beyond the presented experiments.

4.12. Future Development

Several extensions follow directly from the identified limitations. Automatic dimensional typing could prevent invalid operations during tree generation and variation. Physics-aware constraints could incorporate monotonicity, symmetry, conservation relationships, and known boundary conditions into fitness evaluation or program construction.

Improved numerical-constant optimization could help recover equations such as the more difficult Lorenz components and reduce the need for the genetic search to approximate coefficients structurally. Possible approaches include local nonlinear optimization, linear scaling of multiple symbolic components, gradient-based constant refinement, and hybrid evolutionary–deterministic search.

The execution engine should be evaluated with DAG compilation, bounded subtree caching, semantic duplicate elimination, vectorized batch scheduling, multithreading, multiprocessing, and island parallelism. A more complete profiling study should jointly examine fitting time, prediction latency, native memory consumption, cache efficiency, and energy usage.

Classification development should include explicit calibration benchmarks, cost-sensitive threshold optimization, multilabel classification, ordinal classification, class-specific reject options, and distribution-shift-aware threshold refitting. The multiclass shared-backbone strategy could also be extended with automatic component-number selection and sparsity penalties on the softmax head.

From an interpretability perspective, future releases should provide stronger algebraic canonicalization, symbolic equivalence detection, unit-aware simplification, minimum-description-length criteria, stability analysis across bootstrap samples, and tools for domain experts to rank Pareto-optimal equations.

Finally, the benchmark should be expanded to include more symbolic regression libraries, conventional black-box models, larger multiclass datasets, noisy dynamical systems, real experimental physical data, and repeated nested validation. Such evaluation would clarify when the computational and interpretability advantages of FastSymbolicGP translate into practical advantages over established symbolic and nonsymbolic learning methods.

5. Conclusions

This paper presented FastSymbolicGP, a lightweight Python library for symbolic regression, binary classification, and multiclass classification based on tree-structured genetic programming. The library combines a scikit-learn-compatible estimator interface with tournament selection, subtree crossover, subtree mutation, hoist mutation, point mutation, elitism, protected numerical operations, validation-aware evolution, adaptive parsimony, explicit complexity measurement, candidate-equation pools, and Numba-compiled post-fix evaluation.

The experimental evaluation considered predictive accuracy, computational efficiency, symbolic complexity, physical law recovery, dimensional consistency, class imbalance, threshold selection, multiclass decision strategies, dynamical system identification, scalability, parameter sensitivity, and evolutionary ablation. Under the evaluated configurations, FastSymbolicGP achieved the highest mean R^2 on all four investigated real-world scientific regression datasets and obtained significantly better pooled predictive results than gplearn

and PySR. On the dynamical system benchmark, it significantly outperformed gplearn and obtained performance comparable to PySR, while recovering scientifically recognizable expressions such as the exact Lorenz x equation.

The classification experiments showed that FastSymbolicGP can support both conventional and imbalance-aware symbolic decision models. Validation-based threshold selection improved performance under severe imbalance, while balanced class weighting increased mean balanced accuracy when minority classes were rare. In multiclass classification, the one-versus-rest and shared-softmax strategies exhibited complementary strengths. Shared softmax performed strongly on the smaller three-class problems and reduced prediction latency, whereas one-versus-rest was more accurate on the ten-class Digits dataset.

The scalability analysis showed that the Numba postfix evaluator becomes increasingly advantageous as the number of samples grows. At 50,000 samples, FastSymbolicGP was approximately 6.20 times faster than PySR and 1.63 times faster than gplearn while obtaining predictive performance nearly identical to PySR and substantially higher than gplearn. These results support the use of FastSymbolicGP for repeated symbolic learning experiments on medium-sized and large datasets.

The physical law experiments also established an important methodological limitation. Nearly perfect predictive performance did not guarantee physical validity when dimensional variables were supplied directly. Nondimensionalization increased the dimensional-validity rate of FastSymbolicGP expressions from 24% to 96%, while simultaneously reducing runtime and symbolic complexity. Consequently, symbolic transparency should not be confused with automatic scientific correctness. Recovered equations must still be evaluated using units, physical constraints, limiting behavior, and domain knowledge.

The equation-pool analysis confirmed the practical value of complexity-aware model selection. In benchmark groups where a simpler near-optimal model was available, expression complexity decreased by an average of 28.6%, while the average predictive reduction was only 0.00063. This finding supports the use of Pareto archives and tolerance-based model selection when the objective is not only accurate prediction but also compact and communicable mathematical structure.

FastSymbolicGP is therefore not presented as a universally superior symbolic learning algorithm. The comparative results are specific to the evaluated datasets, configurations, primitive sets, stopping criteria, and computational budgets, and alternative tuning of the compared libraries may produce different outcomes. Specialized systems can be preferable when a known sparse candidate library, extensive compiled parallelism, advanced numerical optimization, or physics-specific decomposition is available. The contribution of FastSymbolicGP lies instead in providing a practical balance of predictive quality, computational efficiency, task coverage, estimator compatibility, and model interpretability within a lightweight Python framework.

Future work will focus on unit-aware genetic programming, physics-constrained evolution, improved numerical-constant optimization, probability calibration, semantic duplicate elimination, DAG and cache acceleration, parallel island execution, stronger algebraic simplification, and broader independent benchmarking. Subject to these extensions and external validation, FastSymbolicGP provides a promising foundation for reproducible symbolic machine learning in scientific, engineering, biomedical, and classification-oriented applications.

Funding: This research was supported by the University of Rijeka scientific grant GENE-CLASS, uniri-iz-25-220, funded by the European Union–NextGenerationEU.

Data Availability Statement: The FastSymbolicGP library presented in this study is publicly available through the Python Package Index (PyPI) at <https://pypi.org/project/fastsymbolicgp/> accessed on 25 July 2026, and through the GitHub repository at <https://github.com/nandelic2022/FastSymbolicGP> accessed on 25 July 2026. The benchmark datasets used in this study are publicly available standard datasets distributed through the Python scientific machine learning ecosystem, including datasets available in scikit-learn and synthetically generated datasets created using reproducible Python scripts. The generated benchmark results, tables, and figures are available from the author upon reasonable request.

Acknowledgments: The author acknowledges the University of Rijeka, Faculty of Engineering for institutional and scientific support. This research was supported by the University of Rijeka scientific grant GENE-CLASS, uniri-iz-25-220, funded by the European Union—NextGenerationEU.

Conflicts of Interest: The author declares no conflicts of interest.

References

- Schmidt, M.; Lipson, H. Distilling Free-Form Natural Laws from Experimental Data. *Science* **2009**, *324*, 81–85. <https://doi.org/10.1126/science.1165893>.
- Makke, N.; Chawla, S. Interpretable Scientific Discovery with Symbolic Regression: A Review. *Artif. Intell. Rev.* **2024**, *57*, 2. <https://doi.org/10.1007/s10462-023-10622-0>.
- Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*; MIT Press: Cambridge, MA, USA, 1992.
- Poli, R.; Langdon, W.B.; McPhee, N.F. *A Field Guide to Genetic Programming*; Lulu Enterprises: London, UK, 2008.
- Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
- Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API Design for Machine Learning Software: Experiences from the Scikit-learn Project. In Proceedings of the ECML PKDD Workshop: Languages for Data Mining and Machine Learning, Prague, Czech Republic, 23–27 September 2013; pp. 108–122.
- Fortin, F.A.; De Rainville, F.M.; Gardner, M.A.; Parizeau, M.; Gagné, C. DEAP: Evolutionary Algorithms Made Easy. *J. Mach. Learn. Res.* **2012**, *13*, 2171–2175.
- Stephens, T. gplearn: Genetic Programming in Python. 2016. Available online: <https://github.com/trevorstephens/gplearn> (accessed on 16 June 2026).
- Cranmer, M. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. *arXiv* **2023**, arXiv:2305.01582. <https://doi.org/10.48550/arXiv.2305.01582>.
- Angelis, D.; Sofos, F.; Karakasidis, T.E. Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives. *Arch. Comput. Methods Eng.* **2023**, *30*, 3845–3865. <https://doi.org/10.1007/s11831-023-09922-z>.
- Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 3932–3937. <https://doi.org/10.1073/pnas.1517384113>.
- Rudy, S.H.; Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Data-Driven Discovery of Partial Differential Equations. *Sci. Adv.* **2017**, *3*, e1602614. <https://doi.org/10.1126/sciadv.1602614>.
- Cranmer, M.; Sanchez-Gonzalez, A.; Battaglia, P.; Xu, R.; Cranmer, K.; Spergel, D.; Ho, S. Discovering Symbolic Models from Deep Learning with Inductive Biases. In Proceedings of the Advances in Neural Information Processing Systems, Online, 6–12 December 2020; Volume 33, pp. 17429–17442.
- Udrescu, S.M.; Tegmark, M. AI Feynman: A Physics-Inspired Method for Symbolic Regression. *Sci. Adv.* **2020**, *6*, eaay2631. <https://doi.org/10.1126/sciadv.aay2631>.
- Haider, C.; de França, F.O.; Burlacu, B.; Kronberger, G. Shape-Constrained Multi-Objective Genetic Programming for Symbolic Regression. *Appl. Soft Comput.* **2023**, *132*, 109855. <https://doi.org/10.1016/j.asoc.2022.109855>.
- Kubalík, J.; Derner, E.; Babuška, R. Multi-Objective Symbolic Regression for Physics-Aware Dynamic Modeling. *Expert Syst. Appl.* **2021**, *182*, 115210. <https://doi.org/10.1016/j.eswa.2021.115210>.
- Vladislavleva, E.J.; Smits, G.F.; den Hertog, D. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Trans. Evol. Comput.* **2009**, *13*, 333–349. <https://doi.org/10.1109/TEVC.2008.926486>.
- Vanneschi, L.; Castelli, M.; Silva, S. Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, 7–11 July 2010; pp. 877–884. <https://doi.org/10.1145/1830483.1830643>.

19. Aldeia, G.S.I.; de França, F.O. Interpretability in Symbolic Regression: A Benchmark of Explanatory Methods Using the Feynman Data Set. *Genet. Program. Evolvable Mach.* **2022**, *23*, 309–349. <https://doi.org/10.1007/s10710-022-09435-x>.
20. Virgolin, M.; Pissis, S.P. Symbolic Regression is NP-Hard. *Trans. Mach. Learn. Res.* **2022**.
21. McConaghy, T. FFX: Fast, Scalable, Deterministic Symbolic Regression Technology. In *Genetic Programming Theory and Practice IX*; Riolo, R., Vladislavleva, E., Moore, J.H., Eds.; Springer: New York, NY, USA, 2011; pp. 235–260. https://doi.org/10.1007/978-1-4614-1770-5_13.
22. Petersen, B.K.; Landajuela, M.; Mundhenk, T.N.; Santiago, C.P.; Kim, S.K.; Kim, J.T. Deep Symbolic Regression: Recovering Mathematical Expressions from Data via Risk-Seeking Policy Gradients. In Proceedings of the International Conference on Learning Representations, Online, 3–7 May 2021.
23. Kamienny, P.A.; d’Ascoli, S.; Lample, G.; Charton, F. End-to-End Symbolic Regression with Transformers. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022; Volume 35, pp. 10269–10281.
24. Kommenda, M.; Burlacu, B.; Kronberger, G.; Affenzeller, M. Parameter Identification for Symbolic Regression Using Nonlinear Least Squares. *Genet. Program. Evolvable Mach.* **2020**, *21*, 471–501. <https://doi.org/10.1007/s10710-019-09371-3>.
25. Luke, S.; Panait, L. Fighting Bloat with Nonparametric Parsimony Pressure. In Proceedings of the Parallel Problem Solving from Nature—PPSN VII, Granada, Spain, 7–11 September 2002; Volume 2439, pp. 411–421. https://doi.org/10.1007/3-540-45712-7_40.
26. Poli, R.; McPhee, N.F. Parsimony Pressure Made Easy. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, 12–16 July 2008; pp. 1267–1274. <https://doi.org/10.1145/1389095.1389340>.
27. Bomarito, G.F.; Leser, P.E.; Strauss, N.C.M.; Garbrecht, K.M.; Hochhalter, J.D. Bayesian Model Selection for Reducing Bloat and Overfitting in Genetic Programming for Symbolic Regression. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, New York, NY, USA, 9–13 July 2022; pp. 526–529. <https://doi.org/10.1145/3520304.3528899>.
28. Kronberger, G.; Olivetti de França, F.; Desmond, H.; Bartlett, D.J.; Kammerer, L. The Inefficiency of Genetic Programming for Symbolic Regression. In *Proceedings of the Parallel Problem Solving from Nature—PPSN XVIII*; Affenzeller, M., Winkler, S.M., Kononova, A.V., Bäck, T., Trautmann, H., Tušar, T., Machado, P., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2024; Volume 15148, pp. 273–289. https://doi.org/10.1007/978-3-031-70055-2_17.
29. Orzechowski, P.; La Cava, W.; Moore, J.H. Where Are We Now? A Large Benchmark Study of Recent Symbolic Regression Methods. In Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, 15–19 July 2018; pp. 1183–1190. <https://doi.org/10.1145/3205455.3205539>.
30. La Cava, W.; Orzechowski, P.; Burlacu, B.; de França, F.O.; Virgolin, M.; Jin, Y.; Kommenda, M.; Moore, J.H. Contemporary Symbolic Regression Methods and Their Relative Performance. In Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks, Online, 6–14 December 2021; Volume 1.
31. Korn, M.F. Large-Scale, Time-Constrained Symbolic Regression. In *Genetic Programming Theory and Practice V*; Riolo, R.L., Soule, T., Worzel, B., Eds.; Springer: Boston, MA, USA, 2008; pp. 53–68. https://doi.org/10.1007/978-0-387-76308-8_4.
32. Espejo, P.G.; Ventura, S.; Herrera, F. A Survey on the Application of Genetic Programming to Classification. *IEEE Trans. Syst. Man, Cybern. Part C Appl. Rev.* **2010**, *40*, 121–144. <https://doi.org/10.1109/TSMCC.2009.2033566>.
33. Oechsle, O.; Clark, A.F. Feature Extraction and Classification by Genetic Programming. In *Computer Vision Systems*; Gasteratos, A., Vincze, M., Tsotsos, J.K., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5008, pp. 131–140. https://doi.org/10.1007/978-3-540-79547-6_13.
34. La Cava, W.; Moore, J.H. Learning Feature Spaces for Regression with Genetic Programming. *Genet. Program. Evolvable Mach.* **2020**, *21*, 433–467. <https://doi.org/10.1007/s10710-020-09383-4>.
35. La Cava, W.; Silva, S.; Danai, K.; Spector, L.; Vanneschi, L.; Moore, J.H. Multidimensional Genetic Programming for Multiclass Classification. *Swarm Evol. Comput.* **2019**, *44*, 260–272. <https://doi.org/10.1016/j.swevo.2018.03.015>.
36. Burlacu, B.; Kronberger, G.; Kommenda, M. Operon C++: An Efficient Genetic Programming Framework for Symbolic Regression. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, Cancún, Mexico, 8–12 July 2020; pp. 1562–1570. <https://doi.org/10.1145/3377929.3398099>.
37. de França, F.O. Transformation-Interaction-Rational Representation for Symbolic Regression. In Proceedings of the Genetic and Evolutionary Computation Conference, Boston, MA, USA, 9–13 July 2022; pp. 920–928. <https://doi.org/10.1145/3512290.3528695>.
38. Olson, R.S.; La Cava, W.; Orzechowski, P.; Urbanowicz, R.J.; Moore, J.H. PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *Biodata Min.* **2017**, *10*, 36. <https://doi.org/10.1186/s13040-017-0154-4>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.