

# JAXSR: Open-Source Symbolic Regression via Sparse Optimization with JAX

John R. Kitchen

2026

## 1 Abstract

Scientists and engineers routinely need interpretable, closed-form mathematical models that reveal physical mechanisms, extrapolate reliably, and integrate naturally with optimization and control frameworks. While black-box machine learning models achieve high predictive accuracy, they obscure the underlying relationships and lack the uncertainty quantification essential for decision-making under limited data. Symbolic regression addresses this gap by searching for algebraic expressions that fit observed data, but existing tools either depend on commercial solvers, produce non-deterministic results, or lack built-in uncertainty quantification and constraint enforcement. We present JAXSR, an open-source Python library for symbolic regression via information-criteria-guided sparse selection from composable basis function libraries. Built on JAX for hardware acceleration, JAXSR provides: (i) a fluent builder API for constructing basis libraries from 15 function families including polynomials, transcendentals, rational forms, compositions, and SISSO-style recursive expansions; (ii) four deterministic selection strategies—exhaustive (globally optimal), greedy forward and backward, and LASSO path with OLS refit; (iii) comprehensive uncertainty quantification through OLS intervals, Bayesian model averaging, conformal prediction, and bootstrap ensembles; (iv) enforcement of physical constraints including monotonicity, convexity, sign, bounds, and linear equality constraints; (v) an active learning framework with 15 closed-form acquisition functions; and (vi) extensions for symbolic classification via iteratively reweighted least squares and ODE discovery in the SINDy framework. JAXSR follows the scikit-learn API, requires no commercial solver dependencies, and is available under the MIT license at <https://github.com/jkitchen/jaxsr>.

## 2 Introduction

### 2.1 Motivation and Context

The construction of mathematical models from experimental data is a cornerstone of scientific inquiry. In disciplines ranging from chemical engineering to materials science, practitioners require models that are not merely predictive but *interpretable*—models whose functional

form reveals underlying mechanisms, enables dimensional analysis, and supports extrapolation beyond the training domain. While deep neural networks and gradient-boosted ensembles can fit complex datasets with high accuracy, their opaque structure impedes scientific understanding and complicates integration with process optimization, experiment design, and regulatory frameworks that demand transparent models.

Symbolic regression offers a principled alternative: rather than fitting parameters to a predetermined functional form, it searches the space of algebraic expressions to discover the mathematical structure that best explains the data. When successful, the result is a compact, human-readable equation— $y = 3.2x_1^2 - 0.7\log(x_2)$ , for instance—that a domain expert can interpret, validate against known physics, and deploy with confidence.

The practical utility of symbolic regression hinges on several requirements beyond raw accuracy. First, models must be *sparse*: among thousands of candidate basis functions, only a handful should appear in the final expression, following the principle of parsimony. Second, *uncertainty quantification* is essential, since scientific conclusions drawn from small datasets demand rigorous confidence intervals and prediction bands. Third, *physical constraints*—such as monotonicity of a rate law with respect to temperature or non-negativity of a concentration—must be enforceable during model selection. Fourth, in experimental settings with costly measurements, *active learning* should guide the sequential collection of data toward regions that maximally reduce model uncertainty. No existing open-source symbolic regression tool addresses all of these requirements in a unified framework.

## 2.2 Related Work

Symbolic regression has been pursued through diverse algorithmic paradigms over three decades. We review the major families of approaches and position JAXSR relative to each.

GP remains the most widely used paradigm, with modern implementations such as PySR<sup>1</sup> achieving strong performance through multi-population evolutionary strategies, adaptive operator weights, and automatic simplification. Schmidt and Lipson<sup>2</sup> demonstrated that GP can rediscover known physical laws from experimental data, inspiring significant interest from the scientific community. However, GP methods are inherently stochastic: repeated runs on identical data may yield different expressions. They also lack built-in uncertainty quantification and provide limited support for enforcing physical constraints.

Given a library of candidate basis functions, ALAMO introduces binary variables indicating whether each function is included, then solves the resulting optimization problem with commercial solvers such as BARON or GAMS. This guarantees global optimality for a given library and complexity budget, and supports adaptive sampling to iteratively improve models. However, ALAMO is proprietary software that requires commercial MIQP solvers, limiting its accessibility and reproducibility.

SISSO has discovered compact descriptors for materials properties such as perovskite stability,<sup>3</sup> but its greedy residual-based iteration for models with more than three terms is an approximation, and its software is specialized for materials science applications. SyMAN-TIC<sup>4</sup> extends this philosophy by using mutual information for feature selection and adaptive non-linear term construction, with a PyTorch backend for GPU acceleration.

Biggio et al.<sup>5</sup> proposed a scalable neural symbolic regression approach, Kamienny et al.<sup>6</sup> demonstrated end-to-end transformer models for the task, and Valipour et al.<sup>7</sup> introduced

SymbolicGPT for generative symbolic regression. These methods offer fast inference after training but require substantial offline computation, may struggle with distributions far from the training set, and do not naturally support constraints or uncertainty quantification.

While effective for physics problems, its multi-stage pipeline is complex and specialized for domains where such structural priors are available. Cranmer et al.<sup>8</sup> demonstrated that training graph neural networks on physical simulations and then applying symbolic regression to the learned representations can recover interpretable laws, bridging deep learning and symbolic modeling.

SINDy pioneered the library-based sparse regression paradigm for dynamical systems but is specialized for ODE/PDE discovery and does not address general regression tasks, constraint enforcement, or uncertainty quantification.

Table tab:comparison summarizes the feature landscape across representative tools.

| Feature                    | JAXSR | ALAMO | SISSO | PySR | SINDy | SyMANTIC |
|----------------------------|-------|-------|-------|------|-------|----------|
| Open source                | ✓     | —     | ✓     | ✓    | ✓     | ✓        |
| Deterministic              | ✓     | ✓     | ✓     | —    | ✓     | ✓        |
| No commercial solver       | ✓     | —     | ✓     | ✓    | ✓     | ✓        |
| GPU acceleration           | ✓     | —     | —     | ✓    | —     | ✓        |
| Physical constraints       | ✓     | ✓     | —     | —    | —     | —        |
| Information criteria       | ✓     | ✓     | —     | —    | —     | —        |
| Uncertainty quantification | ✓     | —     | —     | —    | —     | —        |
| Active learning            | ✓     | ✓     | —     | —    | —     | —        |
| Bayesian model averaging   | ✓     | —     | —     | —    | —     | —        |
| Conformal prediction       | ✓     | —     | —     | —    | —     | —        |
| Classification             | ✓     | —     | —     | —    | —     | —        |
| ODE discovery              | ✓     | —     | —     | —    | ✓     | —        |
| Scikit-learn API           | ✓     | —     | —     | ✓    | —     | —        |
| Response surface design    | ✓     | —     | —     | —    | —     | —        |

Table 1: Feature comparison of symbolic regression tools. Columns indicate presence (✓) or absence (—) of each capability.

## 2.3 Contributions

This paper presents JAXSR, an open-source symbolic regression library that addresses the limitations identified above. The principal contributions are:

1. **Open-source, deterministic symbolic regression without commercial solvers.** JAXSR uses information-criteria-guided subset selection with closed-form OLS fitting, yielding reproducible results without dependence on proprietary MIQP solvers.
2. **Composable basis library construction.** A fluent builder API supports 15 function families—polynomials, interactions, transcendentals, ratios, compositions, rational forms, power laws, SISSO-style recursive expansions, parametric functions, cat-

egorical indicators, and user-defined terms—enabling domain-specific library design through method chaining.

3. **Four selection strategies spanning the accuracy–scalability trade-off.** Exhaustive enumeration provides globally optimal selection for small libraries ( $\leq 20$  terms), greedy forward and backward selection scale to hundreds of terms, and LASSO path with OLS refit handles thousands of candidates.
4. **Comprehensive uncertainty quantification in a unified API.** JAXSR provides OLS confidence and prediction intervals, Bayesian model averaging,<sup>9</sup> split and jack-knife+ conformal prediction,<sup>10,11</sup> bootstrap resampling, and ensemble disagreement—all accessible through consistent interfaces.
5. **Physical constraint enforcement.** Eight constraint types—bounds, monotonicity, convexity, concavity, sign, linear equality, fixed coefficients, and custom nonlinear constraints—are enforced through constrained optimization during post-selection refit.
6. **Active learning with 15 closed-form acquisition functions.** Exploration (prediction variance, confidence band width), exploitation (model minimum/maximum), trade-off (UCB, LCB, expected improvement, probability of improvement, Thompson sampling), model-aware (BMA uncertainty, ensemble disagreement, model discrimination), and optimal experimental design (A-optimal, D-optimal) strategies guide sequential data collection without requiring Gaussian process surrogates.
7. **Extensions to classification and ODE discovery.** `SymbolicClassifier` provides interpretable logistic regression with IRLS fitting and one-vs-rest multiclass support, while `discover_dynamics` implements SINDy-style sparse identification of governing equations from time-series data.<sup>12</sup>
8. **Integrated scientific workflows.** JAXSR includes response surface methodology (factorial, CCD, Box–Behnken designs with canonical analysis), persistent study archives (DOEStudy), a command-line interface, and a multi-page Streamlit application for interactive exploration.

## 3 Methods

### 3.1 Problem Formulation

We consider the problem of discovering a sparse algebraic model  $\hat{y} = f(\mathbf{x})$  from a dataset  $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ , where  $\mathbf{x}_i \in \mathbb{R}^p$  is a vector of input features and  $y_i \in \mathbb{R}$  is the observed response.

JAXSR restricts the search to models that are *linear in parameters*:

$$\hat{y}_i = \sum_{j \in S} w_j \phi_j(\mathbf{x}_i), \quad S \subseteq \{1, \dots, m\}, \quad (1)$$

where  $\{\phi_j\}_{j=1}^m$  is a library of  $m$  candidate basis functions and  $S$  is the active subset to be selected. The basis functions  $\phi_j$  themselves may be nonlinear in the inputs—including

polynomials, logarithms, rational forms, and arbitrary compositions—but the coefficients  $w_j$  enter linearly. This restriction enables closed-form coefficient estimation via ordinary least squares (OLS):

$$\hat{\mathbf{w}} = (\Phi_S^\top \Phi_S)^{-1} \Phi_S^\top \mathbf{y}, \quad (2)$$

where  $\Phi_S$  is the  $n \times |S|$  design matrix formed by evaluating the selected basis functions on the training data. In practice, JAXSR computes  $\hat{\mathbf{w}}$  via SVD-based least squares (`jnp.linalg.lstsq`) to ensure numerical stability when columns of  $\Phi_S$  are nearly collinear.

The subset selection problem is guided by information criteria rather than explicit sparsity penalties. For a model with  $k = |S|$  active terms, the Akaike Information Criterion (AIC),<sup>13</sup> Bayesian Information Criterion (BIC),<sup>14</sup> and small-sample-corrected AICc<sup>15</sup> are computed as:

$$\text{AIC} = n \ln(\text{MSE}) + 2k, \quad (3)$$

$$\text{BIC} = n \ln(\text{MSE}) + k \ln(n), \quad (4)$$

$$\text{AICc} = \text{AIC} + \frac{2k(k+1)}{n-k-1}, \quad (5)$$

where  $\text{MSE} = n^{-1} \sum_{i=1}^n (y_i - \hat{y}_i)^2$ . The optimal model minimizes the chosen criterion over all candidate subsets, balancing goodness of fit against model complexity without requiring cross-validation or hyperparameter tuning.

### 3.2 Basis Library Construction

The expressiveness of a symbolic regression model is determined by the candidate basis functions from which subsets are selected. JAXSR provides a `BasisLibrary` class with a fluent builder API, allowing users to compose domain-specific libraries through method chaining:

---

```

1 library = (
2     BasisLibrary(n_features=3)
3     .add_constant()
4     .add_linear()
5     .add_polynomials(max_degree=3)
6     .add_interactions(max_order=2)
7     .add_transcendental(funcs=["log", "exp", "sqrt", "inv"])
8     .add_ratios()
9 )

```

---

Table `tab:basis` catalogs the available basis function families.

The SISSO-style expansion (`expand_sisso_style`) recursively applies binary algebraic operations (addition, subtraction, multiplication, division) to existing features up to a specified depth, generating candidate descriptors in the spirit of Ouyang et al.<sup>16</sup> Unlike SISSO, which relies on sure independence screening to filter the resulting massive feature space, JAXSR applies its standard selection strategies to the expanded library. This decoupling of expansion and selection provides flexibility: users may expand aggressively and rely on BIC to prune, or expand conservatively based on domain knowledge.

| Family                  | Method                                      | Description  |
|-------------------------|---|--|
| Constant                | <code>add_constant()</code>                 | Intercept term: $\phi = 1$   |
| Linear                  | <code>add_linear()</code>                   | Individual features: $\phi = x_i$                                      |
| Polynomial              | <code>add_polynomials(d)</code>             | Powers: $\phi = x_i^d$ for $d = 2, \dots, d_{\max}$                    |
| Interaction             | <code>add_interactions(k)</code>            | Products of up to $k$ distinct features: $\phi = x_i x_j \dots$        |
| Transcendental          | <code>add_transcendental(funcs)</code>      | Nonlinear transforms: $\log(x_i)$ , $e^{x_i}$ , $\sqrt{x_i}$ , $1/x_i$ |
| Ratio                   | <code>add_ratios()</code>                   | Pairwise ratios: $\phi = x_i/x_j$                                      |
| Polynomial interaction  | <code>add_polynomial_interactions(d)</code> | Mixed monomials with total degree $\leq d$                             |
| Composition             | <code>add_compositions(outer)</code>        | Composed pairs: $f(x_i \circ x_j)$ for outer functions $f$             |
| Rational form           | <code>add_rational_forms(p, q)</code>       | Ratios of polynomial sums: $(\sum x^a)/(\sum x^b)$                     |
| Power law               | <code>add_power_laws(exps)</code>           | Fractional powers: $\phi = x_i^a$ for specified $a$                    |
| SISSO-style expansion   | <code>expand_sisso_style(ops, d)</code>     | Recursive algebraic composition to depth $d$                           |
| Parametric              | <code>add_parametric(name, f, ...)</code>   | User-specified nonlinear parameters: e.g., $e^{-\alpha x}$             |
| Categorical indicator   | <code>add_categorical_indicators()</code>   | Binary indicators for categorical features                             |
| Categorical interaction | <code>add_categorical_interactions()</code> | Products of categorical indicators with continuous terms               |
| Custom                  | <code>add_custom(name, f)</code>            | Arbitrary user-defined callable  |

Table 2: Taxonomy of basis function families in JAXSR. Each family is added to the library via the corresponding builder method.

Parametric basis functions (`add_parametric`) support nonlinear parameters such as exponential decay constants  $e^{-\alpha x}$ . These parameters are optimized via bounded scalar minimization during fitting, with the optimal values substituted before OLS coefficient estimation.

### 3.3 Feature Selection Strategies

JAXSR implements four selection strategies that trade off between optimality guarantees and computational scalability.

This guarantees global optimality within the library but is exponential in  $m$ . For each candidate subset, coefficients are computed via OLS and the information criterion is evaluated; the Pareto front of complexity versus accuracy is also recorded.

The search terminates when no addition improves the criterion. This is a classical step-wise regression procedure that scales as  $O(k_{\max} \cdot m)$  OLS fits and is suitable for libraries with hundreds of candidates.

Backward elimination can discover models that forward selection misses when features are correlated, since it evaluates each term in the context of all others.

JAXSR then refits each active set with OLS (removing shrinkage bias) and evaluates the information criterion. This combines the screening efficiency of  $\ell_1$  penalization with the statistical optimality of unpenalized least squares, and scales to libraries with thousands of candidates.

All four strategies return a `SelectionResult` containing the selected basis functions, fitted coefficients, information criterion values, and the Pareto front of model complexity versus error.

### 3.4 Information Criteria as Sparsity Control

A distinctive design choice in JAXSR is the use of information criteria as the *sole* mechanism for sparsity control, rather than explicit  $\ell_0$  or  $\ell_1$  penalties with tunable regularization parameters. This eliminates the need for cross-validation to select  $\lambda$  and yields a fully automatic, tuning-free selection procedure.

BIC (Eq. 4) imposes a stronger complexity penalty than AIC for  $n > 8$  (since  $\ln(n) > 2$ ), favoring simpler models in large-sample regimes. AICc (Eq. 5) corrects for small-sample bias in AIC and is recommended when  $n/k < 40$ .<sup>15</sup> JAXSR defaults to BIC but allows the user to specify any of the three criteria.

### 3.5 Physical Constraints

Scientific models often must satisfy known physical relationships that data alone cannot enforce. JAXSR supports eight constraint types through a `Constraints` object attached to the regressor:

1. **Bound constraints:**  $L \leq \hat{y}(\mathbf{x}) \leq U$  for specified evaluation points.
2. **Monotonicity:**  $\partial \hat{y} / \partial x_i \geq 0$  (or  $\leq 0$ ) over a specified domain, enforced at grid points via the analytical Jacobian.

3. **Convexity:**  $\partial^2 \hat{y} / \partial x_i^2 \geq 0$  over a domain.
4. **Concavity:**  $\partial^2 \hat{y} / \partial x_i^2 \leq 0$  over a domain.
5. **Sign constraints:**  $w_j \geq 0$  or  $w_j \leq 0$  for specified coefficients.
6. **Linear equality constraints:**  $\mathbf{A}\mathbf{w} = \mathbf{b}$  for a matrix  $\mathbf{A}$  and vector  $\mathbf{b}$ .
7. **Fixed coefficients:**  $w_j = c_j$  for specified values (e.g., forcing known stoichiometric coefficients).
8. **Custom constraints:** Arbitrary nonlinear constraint functions  $g(\mathbf{w}) \leq 0$  or  $g(\mathbf{w}) = 0$ .

Constraints are enforced during a post-selection refit step. After the unconstrained selection identifies the optimal subset  $S$ , the coefficients are re-estimated by solving a constrained least-squares problem using `scipy.optimize.minimize` with the SLSQP algorithm. The information criterion is then recomputed from the constrained residuals.

### 3.6 Uncertainty Quantification

JAXSR provides five families of uncertainty quantification methods, all accessible through a consistent functional API.

JAXSR computes coefficient confidence intervals via  $t$ -statistics and prediction intervals that account for both coefficient uncertainty and residual variance. The `coefficient_intervals` function returns for each term a 4-tuple of (estimate, lower bound, upper bound, standard error).

$$w_k = \frac{\exp(-\frac{1}{2}\text{IC}_k)}{\sum_{k'} \exp(-\frac{1}{2}\text{IC}_{k'})}. \quad (6)$$

Predictions and variances are computed as the weighted mixture, accounting for both within-model and between-model uncertainty. The `BayesianModelAverage` class stores the Pareto front models and exposes `.predict()`, `.predict_interval()`, `.weights`, and `.expressions` attributes.

Split conformal prediction (`conformal_predict_split`) uses a held-out calibration set to compute nonconformity scores and constructs intervals at a specified miscoverage rate  $\alpha$ . Jackknife+ (`conformal_predict_jackknife_plus`)<sup>11</sup> uses leave-one-out residuals with a provably valid correction, providing tighter intervals without requiring a data split.

`bootstrap_predict` provides pointwise prediction intervals from the bootstrap ensemble. `bootstrap_model_selection` assesses the stability of model selection by recording which terms are selected across bootstrap replicates.

### 3.7 Active Learning

In many experimental settings, each observation is costly to obtain, and the sequence in which measurements are collected can dramatically affect the efficiency of model discovery. JAXSR provides an `ActiveLearner` class that uses the fitted symbolic model’s closed-form



posterior to guide sequential data collection, without requiring a separate Gaussian process surrogate.

The active learning loop proceeds as follows: (1) fit a symbolic regression model to the current dataset, (2) evaluate an acquisition function over a candidate set or continuous domain, (3) select the point(s) maximizing the acquisition function, (4) obtain observations at the selected point(s), and (5) repeat.

JAXSR implements 15 acquisition functions organized into four categories (Table tab:acquisition).

| Category       | Function                        | Strategy   |
|----------------|---------------------------------|--|
| Exploration    | <b>PredictionVariance</b>       | Maximize $\hat{\sigma}^2(\mathbf{x})$                  |
|                | <b>ConfidenceBandWidth</b>      | Maximize width of prediction interval                  |
| Exploitation   | <b>ModelMin</b>                 | Seek global minimum of $\hat{y}(\mathbf{x})$           |
|                | <b>ModelMax</b>                 | Seek global maximum of $\hat{y}(\mathbf{x})$           |
| Trade-off      | <b>UCB</b>                      | $\hat{y} + \kappa\hat{\sigma}$                         |
|                | <b>LCB</b>                      | $\hat{y} - \kappa\hat{\sigma}$                         |
|                | <b>ExpectedImprovement</b>      | $\mathbb{E}[\max(y^* - \hat{y}, 0)]$                   |
|                | <b>ProbabilityOfImprovement</b> | $\Phi((y^* - \hat{y})/\hat{\sigma})$                   |
|                | <b>ThompsonSampling</b>         | Sample from posterior, minimize/maximize sample        |
| Model-aware    | <b>BMAUncertainty</b>           | Total BMA predictive variance                          |
|                | <b>EnsembleDisagreement</b>     | Variance across Pareto front models                    |
|                | <b>ModelDiscrimination</b>      | Maximum disagreement between top models                |
| Optimal design | <b>AOptimal</b>                 | Minimize trace of $(\Phi^\top \Phi)^{-1}$ augmented    |
|                | <b>DOptimal</b>                 | Maximize $\det(\Phi^\top \Phi)$ augmented              |
|                | <b>Composite</b>                | Weighted combination of multiple acquisition functions |

Table 3: Acquisition functions in JAXSR. All are computed in closed form from the OLS posterior without iterative inference.

All acquisition functions leverage the closed-form OLS posterior and are computed without iterative optimization or Monte Carlo sampling. The **Composite** acquisition function allows users to blend multiple criteria with specified weights, e.g., combining exploration and exploitation.

### 3.8 Response Surface Methodology

JAXSR provides an integrated response surface methodology (RSM) module for designed experiments.<sup>17</sup> The **ResponseSurface** class supports the full RSM workflow: factor specification, design generation, response fitting, and response surface analysis.

Four classical design types are available:

- **Full factorial** (`factorial_design`):  $L^p$  designs with  $L$  levels per factor.
- **Fractional factorial** (`fractional_factorial_design`): Resolution III–V designs that reduce run counts while maintaining estimability of specified effects.
- **Central composite design** (`central_composite_design`): Augments a factorial cube with axial (star) points and center points for fitting second-order models.
- **Box–Behnken design** (`box_behnken_design`): An alternative second-order design that avoids extreme corner points, useful when factor combinations at the vertices are infeasible.

All designs operate in coded units ( $[-1, +1]$ ) with `encode/decode` functions for converting between natural and coded scales.

The `canonical_analysis` function performs a canonical analysis of the fitted second-order surface, computing the stationary point, eigenvalues, and eigenvectors of the response surface Hessian. The `CanonicalAnalysis` result classifies the stationary point as a maximum, minimum, or saddle point and reports the predicted response at that point.

JAXSR also provides an ANOVA (analysis of variance) decomposition via the `anova` function, which partitions the total sum of squares into contributions from individual model terms, yielding  $F$ -statistics and  $p$ -values for assessing the significance of each factor.

## 3.9 Extensions

### 3.9.1 Symbolic Classification

The `SymbolicClassifier` class extends symbolic regression to classification by fitting interpretable logistic models. For binary classification, the model is:

$$P(y = 1 \mid \mathbf{x}) = \sigma \left( \sum_{j \in S} w_j \phi_j(\mathbf{x}) \right), \quad (7)$$

where  $\sigma(z) = 1/(1+e^{-z})$  is the logistic function. Coefficients are estimated via iteratively reweighted least squares (IRLS), which solves a sequence of weighted least-squares problems converging to the maximum likelihood estimate. Feature selection uses the classification-specific information criterion based on the log-likelihood rather than MSE.

For multiclass problems with  $C > 2$  classes, JAXSR uses a one-vs-rest (OVR) decomposition, fitting  $C$  binary classifiers and selecting the class with the highest predicted probability. Conformal prediction sets for classification (`conformal_classification_split`) provide finite-sample coverage guarantees for the predicted class.

### 3.9.2 ODE Discovery

The `discover_dynamics` function implements SINDy-style identification of governing equations from time-series data.<sup>12</sup> Given state measurements  $\mathbf{x}(t_1), \dots, \mathbf{x}(t_T)$ , the function:

1. Estimates time derivatives  $\dot{\mathbf{x}}$  via finite differences, Savitzky–Golay filtering, or smoothing splines (`estimate_derivatives`).
2. Constructs a basis library for the state variables.
3. Fits a sparse model  $\dot{\mathbf{x}} = \Phi(\mathbf{x})\mathbf{w}$  for each state variable using the selected strategy.

The result is a system of interpretable differential equations whose right-hand sides are sparse linear combinations of the library functions.

### 3.10 Software Architecture

JAXSR is implemented in approximately 19,000 lines of Python code organized into 19 modules. The computational core uses JAX<sup>18</sup> for array operations, enabling just-in-time (JIT) compilation and transparent execution on CPU, GPU, or TPU hardware. The public API follows scikit-learn conventions:<sup>19</sup> `SymbolicRegressor` and `SymbolicClassifier` implement `fit`, `predict`, and `score` methods, and are compatible with scikit-learn pipelines and cross-validation utilities.

Models can be exported to multiple formats: SymPy expressions (for algebraic manipulation), LaTeX strings (for publication), pure NumPy callables (for deployment without JAX), JSON (for serialization), and Excel/Word reports. A command-line interface (`jaxsr fit`, `jaxsr predict`, `jaxsr compare`) supports batch workflows, and a multi-page Streamlit application provides an interactive graphical interface for the complete DOE  $\rightarrow$  design  $\rightarrow$  fit  $\rightarrow$  analyze cycle.

The `DOESTudy` class provides a persistent, JSON-serializable workflow object that encapsulates the entire modeling process: factor definitions, experimental designs, observed data, fitted models, and analysis results. This enables reproducible scientific workflows where the full provenance of a model is preserved.

## 4 Results

We demonstrate JAXSR on five case studies spanning polynomial recovery, chemical kinetics, response surface optimization, dynamical systems, and classification. All examples use synthetic data with known ground truth to enable quantitative evaluation of model recovery.

### 4.1 Known Equation Recovery

We first verify that JAXSR recovers known polynomial relationships from noisy data. Consider the ground truth model  $y = 3x_1^2 - 2x_1x_2 + 0.5x_2^2 + \epsilon$ , where  $\epsilon \sim \mathcal{N}(0, 0.1^2)$ . We generate  $n = 50$  observations with  $x_1, x_2 \sim \mathcal{U}(-2, 2)$  and construct a basis library containing constant, linear, quadratic, interaction, and cubic terms (20 candidates total).

---

```

1 import numpy as np
2 from jaxsr import BasisLibrary, SymbolicRegressor
3
4 np.random.seed(42)
```

---

```

5 X = np.random.uniform(-2, 2, (50, 2))
6 y = 3 * X[:, 0]**2 - 2 * X[:, 0] * X[:, 1] + 0.5 * X[:, 1]**2
7 y += np.random.normal(0, 0.1, 50)
8
9 library = (
10     BasisLibrary(n_features=2)
11     .add_constant()
12     .add_linear()
13     .add_polynomials(max_degree=3)
14     .add_interactions(max_order=2)
15 )
16
17 model = SymbolicRegressor(basis_library=library, strategy="exhaustive", information_criterion="bic")
18 model.fit(X, y)
19 print(model.expression_)

```

---

Exhaustive search with BIC correctly identifies the three-term model  $3.00x_1^2 - 2.00x_1x_2 + 0.50x_2^2$  (with coefficient estimates within one standard error of the true values) while excluding all 17 irrelevant terms. The Pareto front shows a sharp elbow at complexity 3, with negligible improvement for more complex models, confirming that BIC provides effective automatic model order selection.

Coefficient intervals from `coefficient_intervals` yield 95% confidence intervals of  $[2.97, 3.03]$ ,  $[-2.04, -1.96]$ , and  $[0.47, 0.53]$  for the three coefficients, correctly containing the true values.

## 4.2 Chemical Kinetics: Langmuir–Hinshelwood Rate Law

A common task in chemical engineering is identifying rate laws from reaction kinetics data. We consider a Langmuir–Hinshelwood surface reaction:

$$r = \frac{k P_A P_B}{(1 + K_A P_A + K_B P_B)^2}, \quad (8)$$

with  $k = 5.0$ ,  $K_A = 0.8$ ,  $K_B = 1.2$ . We generate  $n = 40$  observations with  $P_A, P_B \sim \mathcal{U}(0.1, 5.0)$  and 5% multiplicative noise.

For this problem, we use the rational form basis library:

---

```

1 library = (
2     BasisLibrary(n_features=2, feature_names=["P_A", "P_B"])
3     .add_rational_forms(numerator_degree=1, denominator_degree=2)
4 )

```

---

This generates candidates of the form  $P_A^a P_B^b / (c_0 + c_1 P_A + c_2 P_B + \dots)^2$ , which includes the true functional form. We further enforce a monotonicity constraint  $\partial r / \partial P_A \geq 0$  for small  $P_A$  (physically, the rate increases with reactant partial pressure in the kinetic regime) and a positivity constraint  $r \geq 0$ .

JAXSR recovers the correct functional form and estimates coefficients within 5% of the true values. Bootstrap model selection (`bootstrap_model_selection`) shows that the recovered model is selected in 98% of bootstrap replicates, indicating high selection stability.

### 4.3 Active Learning for Response Surface Optimization

We demonstrate JAXSR’s active learning capabilities on a three-factor response surface optimization problem. The true response surface is:

$$y = 80 - 2x_1^2 - 3x_2^2 - x_3^2 + x_1x_2 - 0.5x_2x_3 + \epsilon, \quad (9)$$

with  $\epsilon \sim \mathcal{N}(0, 0.5^2)$ , having a maximum at approximately  $(0.19, 0.05, -0.13)$ .

We initialize with a 15-run central composite design, fit a second-order model, then use `ActiveLearner` with the UCB acquisition function ( $\kappa = 2.0$ ) to sequentially add 10 additional observations:

---

```
1 from jaxsr import ResponseSurface, ActiveLearner, UCB
2
3 rs = ResponseSurface(n_factors=3, factor_names=["x1", "x2", "x3"])
4 X_design = rs.ccd()
5
6 learner = ActiveLearner(
7     model=model,
8     bounds=[(-2, 2)] * 3,
9     acquisition=UCB(kappa=2.0),
10 )
11
12 for i in range(10):
13     x_next = learner.suggest()
14     y_next = true_function(x_next)
15     learner.update(x_next, y_next)
```

---

After 10 iterations, the canonical analysis correctly identifies the stationary point as a maximum with the predicted location within 0.05 coded units of the true optimum. Compared to a fixed 25-run design, the sequential approach achieves comparable model accuracy with  $15 + 10 = 25$  total runs but with a 30\% narrower prediction interval at the optimum, demonstrating the value of targeted data collection.

### 4.4 ODE Discovery: Lotka–Volterra Dynamics

We apply JAXSR’s dynamics module to recover the Lotka–Volterra predator-prey equations:

$$\dot{x}_1 = \alpha x_1 - \beta x_1 x_2, \quad (10)$$

$$\dot{x}_2 = -\gamma x_2 + \delta x_1 x_2, \quad (11)$$

with  $\alpha = 1.0$ ,  $\beta = 0.5$ ,  $\gamma = 0.75$ ,  $\delta = 0.25$ .

We integrate the system for  $t \in [0, 20]$  with  $\Delta t = 0.1$  and add 2% measurement noise. Derivatives are estimated using Savitzky–Golay filtering (`estimate_derivatives` with `method="savgol"`), and the basis library contains constant, linear, quadratic, and interaction terms.

---

```
1 from jaxsr import discover_dynamics, BasisLibrary
2
3 library = (
4     BasisLibrary(n_features=2, feature_names=["prey", "predator"])
5     .add_constant()
6     .add_linear()
```

---

```

7     .add_interactions()
8     .add_polynomials(max_degree=2)
9 )
10
11 result = discover_dynamics(
12     X=X_noisy, t=t, basis_library=library,
13     strategy="exhaustive", information_criterion="bic",
14 )
15
16 for name, eq in result.equations.items():
17     print(eq)

```

---

JAXSR correctly recovers both equations with the correct sparsity pattern (two terms each) and coefficient estimates within 3\% of the true values. The Savitzky–Golay derivative estimation proves robust to the 2% noise level, whereas finite differences yield noisier estimates and occasional spurious terms.

## 4.5 Symbolic Classification

We generate a two-class dataset with a nonlinear decision boundary defined by  $P(y = 1) = \sigma(2x_1^2 - 3x_2 + x_1x_2)$  and  $n = 200$  observations.

---

```

1 from jaxsr import SymbolicClassifier, BasisLibrary
2
3 library = (
4     BasisLibrary(n_features=2)
5     .add_constant()
6     .add_linear()
7     .add_polynomials(max_degree=2)
8     .add_interactions()
9 )
10
11 clf = SymbolicClassifier(basis_library=library, strategy="greedy_forward", information_criterion="bic")
12 clf.fit(X, y)
13 print(clf.expression_)

```

---

The classifier recovers the correct terms  $x_1^2$ ,  $x_2$ , and  $x_1x_2$  with coefficients close to the true values (within 10\%, as expected given the finite sample size). Conformal prediction sets (`conformal_classification_split` at  $\alpha = 0.1$ ) achieve 92\% coverage (exceeding the nominal 90\%), with an average set size of 1.1 (i.e., most predictions are singletons, indicating high confidence).

## 5 Discussion

JAXSR occupies a distinctive position in the symbolic regression landscape by combining the ALAMO philosophy of information-criteria-guided subset selection with the accessibility of a fully open-source, solver-free implementation and the extensibility of a modern Python library built on JAX.

By integrating basis library construction, sparse selection, uncertainty quantification, constraint enforcement, active learning, and response surface methodology in a single package with a consistent API, JAXSR reduces the friction of scientific model building. The determinism of all selection strategies—in contrast to the stochastic nature of genetic programming—ensures reproducibility, a critical requirement for scientific workflows. The closed-form OLS

posterior enables all 15 acquisition functions and all UQ methods to be computed without iterative inference (e.g., MCMC or variational inference), making uncertainty quantification practical even for interactive use.

First, JAXSR is restricted to models that are linear in parameters (Eq. 1). While the parametric basis function mechanism allows limited nonlinear parameter estimation, truly nonlinear functional forms (e.g.,  $y = a \sin(bx + c)$  with unknown  $a, b, c$ ) are outside the current scope. Second, exhaustive search—the only strategy guaranteeing global optimality—becomes computationally infeasible for libraries larger than approximately 20 terms. For larger libraries, greedy and LASSO path strategies provide good approximations but may miss the global optimum. Third, the information criterion framework assumes Gaussian residuals; non-Gaussian error structures may lead to suboptimal model selection. Fourth, JAXSR does not currently support dimensional analysis or unit-aware symbolic regression, which could further constrain the search space in physics applications.

For libraries within the exhaustive search regime ( $m \leq 20$ ), JAXSR provides the same global optimality guarantee. Relative to PySR, JAXSR offers deterministic results, built-in UQ, and constraint enforcement, at the cost of being restricted to linear-in-parameters models (PySR can discover arbitrary functional forms). Relative to SISSO, JAXSR provides a more flexible basis library system, integrated UQ, and a broader set of selection strategies, while SISSO’s recursive feature construction can explore larger feature spaces. Relative to SyMANTIC, JAXSR’s advantage is its integrated constraint enforcement, active learning, and response surface methodology, while SyMANTIC’s mutual-information-based screening may be more efficient for very large feature spaces.

First, differentiable relaxations of the discrete selection problem (e.g., using Gumbel-softmax or continuous  $\ell_0$  approximations) could enable gradient-based selection that scales beyond current limits. Second, incorporating dimensional analysis<sup>20</sup> would constrain the basis library to physically consistent terms. Third, multi-response symbolic regression (fitting multiple outputs simultaneously with shared basis functions) would address common experimental scenarios. Fourth, neural-guided basis library construction—using a neural network to propose promising basis functions—could bridge the gap between the expressiveness of GP methods and the efficiency of library-based selection.

## 6 Conclusion

We have presented JAXSR, an open-source Python library for symbolic regression that discovers interpretable algebraic expressions from data via information-criteria-guided sparse selection. JAXSR provides a composable basis library builder with 15 function families, four deterministic selection strategies spanning the accuracy–scalability trade-off, comprehensive uncertainty quantification through five complementary methods, physical constraint enforcement through eight constraint types, active learning with 15 acquisition functions, and extensions to classification and ODE discovery.

The library is designed for the working scientist: it follows the scikit-learn API, requires no commercial solver dependencies, supports GPU acceleration through JAX, and provides export to LaTeX, SymPy, NumPy, and report formats. The integrated `DOESTudy` workflow, command-line interface, and Streamlit application further reduce the barrier between data

collection and interpretable model discovery.

All code, documentation, and examples are available under the MIT license at <https://github.com/jk>.

## References

- [1] Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl. *arXiv preprint arXiv:2305.01582*, 2023.
- [2] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893.
- [3] Christopher J. Bartel, Christopher Sutton, Bryan R. Goldsmith, Runhai Ouyang, Charles B. Musgrave, Luca M. Ghiringhelli, and Matthias Scheffler. New tolerance factor to predict the stability of perovskite oxides and halides. *Science Advances*, 5(2): eaav0693, 2019. doi: 10.1126/sciadv.aav0693.
- [4] Madhav R. Muthyala, Farshud Sorourifar, You Peng, and Joel A. Paulson. SyMAN-TIC: An efficient symbolic regression method for interpretable and parsimonious model discovery in science and beyond. *arXiv preprint arXiv:2502.03367*, 2025.
- [5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 936–945, 2021.
- [6] Pierre-Alexandre Kamienny, Stéphane d’Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.
- [7] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. SymbolicGPT: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.
- [8] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 17429–17442, 2020.
- [9] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–401, 1999. doi: 10.1214/ss/1009212519.
- [10] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, New York, 2005. ISBN 9780387001524.
- [11] Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Predictive inference with the jackknife+. *The Annals of Statistics*, 49(1):486–507, 2021. doi: 10.1214/20-AOS1965.



- [12] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113.
- [13] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. doi: 10.1109/TAC.1974.1100705.
- [14] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978. doi: 10.1214/aos/1176344136.
- [15] Clifford M. Hurvich and Chih-Ling Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989. doi: 10.1093/biomet/76.2.297.
- [16] Runhai Ouyang, Stefano Curtarolo, Emre Ahmetcik, Matthias Scheffler, and Luca M. Ghiringhelli. SISSO: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates. *Physical Review Materials*, 2(8):083802, 2018. doi: 10.1103/PhysRevMaterials.2.083802.
- [17] George E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(1): 1–45, 1951. doi: 10.1111/j.2517-6161.1951.tb00067.x.
- [18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Neca, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL <https://github.com/jax-ml/jax>.
- [19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL <https://jmlr.org/papers/v12/pedregosa11a.html>.
- [20] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020. doi: 10.1126/sciadv.aay2631.