nature computational science

Article

https://doi.org/10.1038/s43588-025-00904-8

Discovering physical laws with parallel symbolic enumeration

Received: 25 March 2025

Accepted: 13 October 2025

Published online: 21 November 2025

Check for updates

Kai Ruan[1], Yilong Xu[1], Ze-Feng Gao[1], Yang Liu[2,3], Yike Guo[4], Ji-Rong Wen[1] & Hao Sun[1]

Symbolic regression has a crucial role in modern scientific research owing to its capability of discovering concise and interpretable mathematical expressions from data. A key challenge lies in the search for parsimonious and generalizable mathematical formulas, in an infinite search space, while intending to fit the training data. Existing algorithms have faced a critical bottleneck of accuracy and efficiency over a decade when handling problems of complexity, which essentially hinders the pace of applying symbolic regression for scientific exploration across interdisciplinary domains. Here, to this end, we introduce parallel symbolic enumeration (PSE) to efficiently distill generic mathematical expressions from limited data. Experiments show that PSE achieves higher accuracy and faster computation compared with the state-of-the-art baseline algorithms across over 200 synthetic and experimental problem sets (for example, improving the recovery accuracy by up to 99% and reducing runtime by an order of magnitude). PSE represents an advance in accurate and efficient data-driven discovery of symbolic, interpretable models (for example, underlying physical laws), and improves the scalability of symbolic learning.

For centuries, scientific discovery has been increasingly driven by data. Symbolic regression (SR) stands at the forefront of this movement, aiming to automatically distill interpretable mathematical expressions from observational data without presupposing specific functional forms[1]. This capability has catalyzed scientific advances across multiple domains, including astronomical modeling[2], materials science[3] and physical law discovery[4], among other applications[5,6]. The fundamental challenge in SR arises from the combinatorial explosion of possible expressions, rendering exhaustive search. While conventional regression techniques excel at parameter estimation for predetermined models[7], they offer limited utility when the underlying mathematical structure is unknown.

Evolutionary computation approaches address this structural discovery problem through heuristic search mechanisms. Genetic programming (GP) and related algorithms[1,8] explore expression spaces by applying genetic operations to tree-based representations, while Bayesian methods[9] employ probabilistic sampling to identify promising expressions. These techniques have shown success in uncovering governing equations from data, for example, for dynamical systems[10]. However, they face critical issues in practical applications, such as poor scalability, sensitivity to hyperparameter configurations, and propensity of generating complex expressions that overfit data.

A notable progress lies in the introduction of sparse regression for equation discovery, for example, the sparse identification of nonlinear dynamics (SINDy) approach[11]. By formulating the discovery as a sparse linear regression problem over a predefined function library, such a method achieves remarkable efficiency in identifying parsimonious models. The family of SINDy approaches have shown remarkable success in data-driven discovery of both ordinary and partial differential equations[12–16]. Nevertheless, their effectiveness is contingent on careful library design that balances expressiveness with tractability. More fundamentally, the limitation to linear combinations of basis functions inherently restricts the complexity of discoverable relationships, particularly when variable interactions are nonlinear and absent from the library.

[1]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China. [2]School of Engineering Science, University of Chinese Academy of Sciences, Beijing, China. [3]State Key Laboratory of Nonlinear Mechanics, Institute of Mechanics, Chinese Academy of Sciences, Beijing, China. [4]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. e-mail: haosun@ruc.edu.cn

Deep learning architectures have recently emerged as powerful alternatives to SR, for example, recurrent neural network with risk-seeking policy gradients[17], grammar-based variational autoencoders[18], neural network-based graph modularity[19] and transformer models pre-trained on mathematical expressions[20–22]. Noteworthy, symbolic neural networks represent a particularly relevant specialization, incorporating mathematical operators as activation functions to enable end-to-end equation discovery[23]. A persistent challenge across these neural approaches is their dependence on empirical thresholding for expression simplification, which introduces subjectivity and often fails to yield meaningful equations when the data are limited and noisy.

Very recently, Monte Carlo tree search (MCTS)[24,25], which gained acclaim for powering the decision-making algorithms in AlphaZero[26], has shown great potential in navigating the expansive search space inherent in SR[27]. This method uses stochastic simulations to meticulously evaluate the merit of each node in the search tree and has empowered several SR techniques[28–31]. However, the conventional application of MCTS maps each node to a unique expression, which tends to impede the rapid recovery of accurate symbolic representations. This narrow mapping may limit the strategy's ability to efficiently parse through the complex space of potential expressions, thus presenting a bottleneck in the quest for swiftly uncovering underlying mathematical equations.

SR involves evaluating a large number of complex symbolic expressions composed of various operators, variables and constants. The operators and variables are discrete, while the value of the coefficients is continuous. The NP-hardness of a typical SR process, noted by many scholars[17,19,32], has been formally established[33]. This characteristic requires the algorithm to traverse various possible combinations, resulting in a huge and even infinite search space and thus facing the problem of combinatorial explosion. Unfortunately, all the existing SR methods evaluate each candidate expression independently, leading to substantially low computational efficiency. Although some studies have considered caching evaluated expressions to prevent recomputation[30], they do not reuse these results as subtree values for evaluating deeper expressions. Consequently, the majority of these methods either rely on meta-heuristic search strategies, narrow down the search space based on specific assumptions, or incorporate pre-trained models to discover relatively complex expressions, which make the algorithms prone to producing specious results (for example, local optima). Therefore, the efficiency of candidate expression evaluation is of paramount importance. By enhancing the efficiency of candidate expression evaluation, it becomes possible to design new SR algorithms that are less reliant on particular optimization methods, while increasing the likelihood of directly finding the global optima in the grand search space. Thus, we can improve the SR accuracy (in particular, the symbolic recovery rate) and, meanwhile, drastically reduce the computational time.

To this end, we propose parallel symbolic enumeration (PSE) to automatically distill symbolic expressions from limited data. Such a model is capable of efficiently evaluating potential expressions, thereby facilitating the exploration of hundreds of millions of candidate expressions simultaneously in parallel within a mere few seconds. In particular, we propose a parallel symbolic regression network (PSRN) as the cornerstone search engine, which (1) automatically captures common subtrees of different math expression trees for shared evaluation that avoids redundant computations, and (2) capitalizes on a graphics processing unit (GPU)-based parallel search that results in a notable performance boost. It is notable that, in a standard SR process for equation discovery, many candidate expressions share common subtrees, which leads to repeatedly redundant evaluations and, consequently, superfluous computation. To address this issue, we propose a strategy to automatically reuse common subtrees for shared evaluation, effectively circumventing redundant computation. We further execute PSE on a GPU to perform a large-scale evaluation of candidate expressions in parallel, thereby augmenting the efficiency of the evaluation process. Notably, the synergy of these two techniques could yield up to four orders of magnitude efficiency improvement in the context of expression evaluation. In addition, to expedite the convergence and enhance the capacity of PSRN for exploration and identification of more intricate expressions, we amalgamate it with a token generator strategy (for example, MCTS or GP) that identifies a set of admissible base expressions as tokenized input. The effectiveness and efficiency of the proposed PSE model have been demonstrated on a variety of benchmark and lab test datasets. The results show that PSE surpasses multiple existing baseline methods, achieving higher symbolic recovery rates and efficiency.

Results

We introduce a general-purpose SR method, called PSE, to discover mathematical expressions from data. The core of this framework is the PSRN module that can evaluate hundreds of millions of candidate expressions in parallel by identifying and reusing common subtree computations, dramatically accelerating the search process. To explore complex expressions, PSRN is integrated into an iterative loop with a token generator (for example, GP), which discovers admissible sub-expressions as building blocks. The overall architecture and workflow of the PSE model are illustrated in Fig. 1. A detailed description of the methodology, including the symbol layer design (Fig. 2), token generation strategies and coefficient optimization, is provided in Methods.

To demonstrate the effectiveness and efficiency of the proposed PSE model, we conduct a comprehensive evaluation across a diverse range of challenging tasks. These include standard SR benchmarks, the data-driven discovery of governing equations for chaotic dynamical systems and the modeling of real-world physical phenomena from experimental data. It is noteworthy that, unless specified otherwise in the ablation studies, all experiments presented hereafter utilize the GP variant of our token generator to effectively explore the vast search space. For real-world datasets such as the electro-mechanical positioning system and the turbulent friction experiments, we assess the plausibility of discovered equations using the mean squared error (MSE) and parsimony due to the absence of ground-truth expressions. For other benchmark experiments, we employ the symbolic recovery rate as the key measure of accuracy. Although the generated expression, further simplified by SymPy, is not guaranteed to be the best-fitting solution matching the ground truth, especially for noisy datasets, this metric remains widely adopted in the SR community. It enables direct and standardized comparisons with a substantial body of prior work on established benchmark problem sets that contain ground-truth expressions. The collective results presented as follows consistently highlight the superiority of PSE in both symbolic recovery accuracy and computational performance over existing baseline methods.

SR benchmarks

We first demonstrate the efficacy of PSE on recovering specified mathematical formulas given multiple benchmark problem sets (including Nguyen[34], Nguyen-c[35], R[36], Livermore[32] and Feynman[19], as described in Supplementary Note 2.1), commonly used to evaluate the performance of SR algorithms. Each SR puzzle consists of a ground-truth equation, a set of available math operators and a corresponding dataset. These benchmark data sets contains various math expressions, for example, $x^3 + x^2 + x$ (Nguyen-1), $3.39x^3 + 2.12x^2 + 1.78x$ (Nguyen-1c), $(x+1)^3/(x^2 - x + 1)$ (R-1), $1/3 + x + \sin(x^2)$ (Livermore-1), $x_1^4 - x_1^3 + x_1^2 - x_2$ (Livermore-5) and $x_1x_2x_3 \log(x_5/x_4)$ (Feynman-9), which are listed in detail in Supplementary Tables 1 and 2. Our objective is to uncover the Pareto front of optimal mathematical expressions that balance the equation complexity and error. The performance of PSE is compared with eight baseline methods, for example, symbolic physics learner (SPL)[27], neural-guided genetic programming (NGGP)[32], deep generative symbolic regression (DGSR)[22], PySR[37], Bayesian machine scientist

https://doi.org/10.1038/s43588-025-00904-8

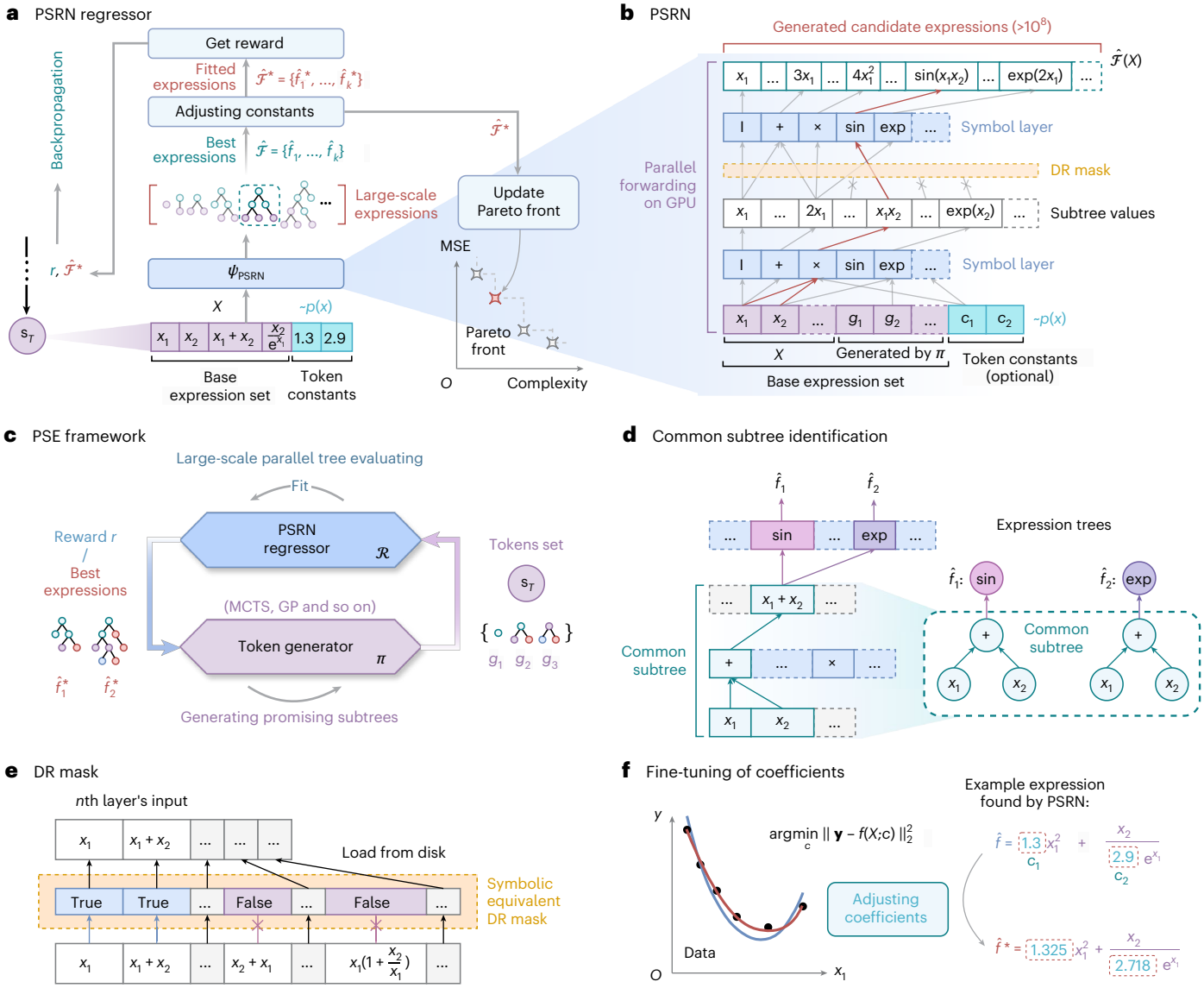


Fig. 1 | Overview of the proposed PSE model. a, Schematic of PSRN regressor for discovering optimal expressions from data. The base expression set $s_{\mathcal{F}}$ from terminal nodes, along with optionally sampled token constant values, is fed into the PSRN (denoted as ψ_{PSRN}) for forward computation. After evaluating the errors of large-scale candidate expressions, ψ_{PSRN} provides the optimal (or top k) expressions denoted as $\hat{\mathcal{F}}$. Subsequently, the least squares method is employed for the identification and fine-tuning of the coefficients, resulting in adjusted expressions $\hat{\mathcal{F}}^*$, which are then utilized for updating the Pareto front as computing the complexity and reward. **b**, Forward computation in PSRN. The base expression set and sampled constants, along with the corresponding data tensor X, are fed into the network. The network comprises multiple symbol layers (for example, typically three, but only two are shown for simplicity). Each layer provides all possible subtree values resulting from one-operator computations among all input expressions. This process can be efficiently parallelized on a GPU for rapid computation. Upon completion of PSRN's forward computation, a myriad of candidate expressions $\hat{\mathcal{F}}$ (for example, up to hundreds of millions) corresponding to the base expression set $s_{\mathcal{F}}$, along with their associated error tensors on the given data, are generated. Then, the expression with the minimal error (or top-k expressions) will be selected. **c**, The token generator π (for example, MCTS, GP) creates promising subtrees from a token set s_T. These subtrees are evaluated by the PSRN regressor $\mathcal{R}$ through large-scale parallel expression evaluation. $\mathcal{R}$ fits the input data, producing a reward r and the best expressions $\hat{\mathcal{F}}^*$, which are fed back to π. This feedback loop drives continuous improvement in generating promising subtrees. The process combines the ability of π to explore the token space with the capacity of $\mathcal{R}$ to assess expression quality based on data fit. This iterative system efficiently discovers high-quality expressions by leveraging both heuristic search and large-scale parallel evaluation. The overall iterative PSE process is detailed in Supplementary Algorithm 2, with specific token generators π described in Supplementary Algorithms 3–5. **d**, Schematic of common subtree identification. Expressions sharing common subtrees, for example, $\sin(x_1 + x_2)$ and $\exp(x_1 + x_2)$, leverage identical subtree computation outcomes. This approach circumvents extensive redundant calculations, thereby increasing the efficiency of SR. **e**, Schematic of duplicate removal mask (DR mask). We designed the DR mask layer to mask the output tensor of the penultimate layer in PSRN, thereby excluding subtree expressions that are symbolically equivalent. This greatly reduces PSRN's usage of GPU memory. **f**, Estimation and fine-tuning of constant coefficients. The least squares method is used to fine-tune the coefficients of preliminary expressions discovered by PSRN using the complete dataset.

https://doi.org/10.1038/s43588-025-00904-8

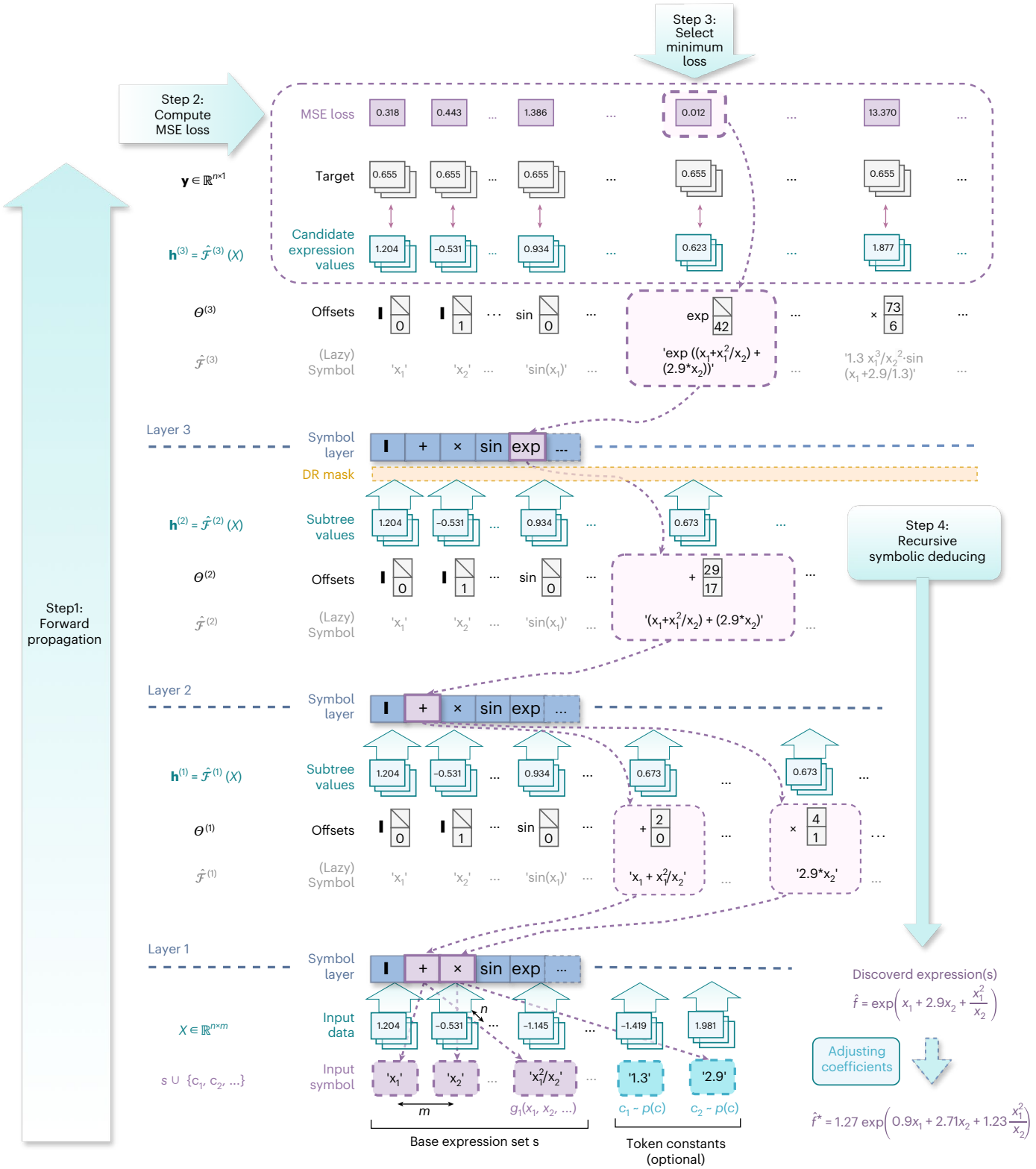


Fig. 2 | The detailed process of PSRN forward propagation for obtaining the optimal expression. Step 1: the input data X and the base expression set s, together with optional token constants sampled from a predefined distribution $p(c)$, are fed into PSRN for forward propagation. Based on the designated operator categories (for example, {**I**, +, ×, sin, exp}, where **I** is the identity operator), a large number of distinct subtree values **h**, for example, $\hat{\mathcal{F}}(X)$ are rapidly calculated layer by layer. Step 2: the MSE is computed between the final layer's output and the broadcast target tensor **y**, resulting in the loss tensor. Step 3: the position of the minimum value in the loss tensor is selected. Step 4: starting from the optimal position, a recursive symbolic deducing is performed using the offset tensor Θ generated when building the network, ultimately yielding the optimal expression. If necessary, the coefficients of this expression will be adjusted in post-processing. The algorithmic procedure for PSRN evaluation and symbolic reconstruction is presented in Supplementary Algorithm 1.

https://doi.org/10.1038/s43588-025-00904-8

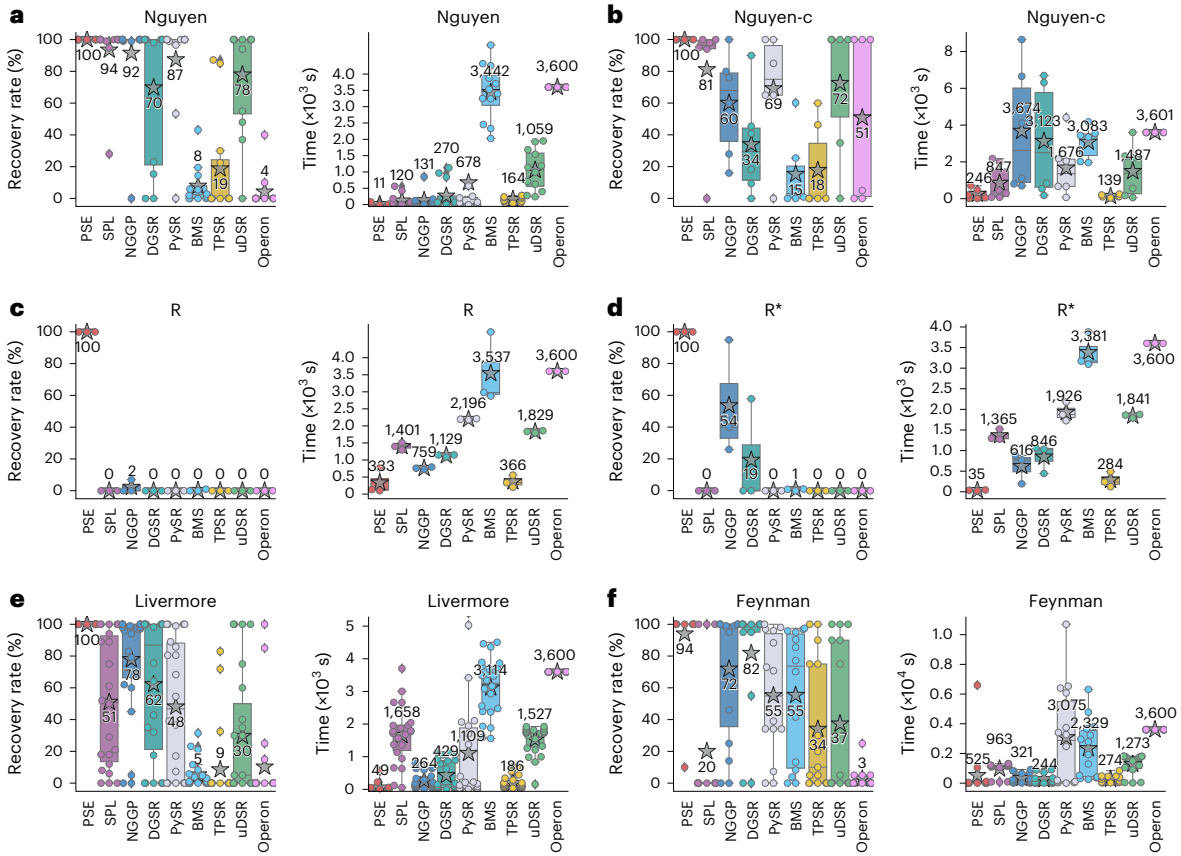


Fig. 3 | Performance of various models on SR benchmarks. a, Nguyen problem set. **b**, Nguyen-c problem set. **c**, R problem set. **d**, R* problem set. **e**, Livermore problem set. **f**, Feynman problem set. We employ two evaluation metrics, namely, symbolic recovery rate (left) and average runtime (right), to assess the performance of each algorithm. Our PSE approach achieves the highest recovery rate across various benchmark problem sets, meanwhile expending the minimal computation time, compared with the baseline methods (for example, SPL[27], NGGP[32], DGSR[22], PySR[37], BMS[9], uDSR[38], TPSR[30] and Operon[39]). This highlights the substantial superiority of PSE. Note that the star marks the mean value over each problem set. As BMS[9] uses priors from Wikipedia, it performs reasonably well on subsets of expressions like Feynman, but has very low recovery rates for other expression subsets as it is not suited for finding artificially designed expressions without constants. The Feynman dataset shown here follows the definition in DGSR's paper, which is a subset of SRBench's Feynman problems under data-limited conditions (fewer than 50 data points). For evaluation on the complete SRBench dataset, see 'SRBench evaluation and performance'. We strictly maintain consistent runtime budgets. The variations in the algorithms' runtime occur because different algorithms trigger their respective early stopping conditions (for example, reaching an MSE below a very small threshold, or discovering symbolically equivalent expressions, and so on). The central line represents the median, the box spans the 25th to 75th percentiles, the whiskers extend to the minimum and maximum values within 1.5 times the interquartile range, and the star indicates the mean recovery rate for each problem set.

(BMS)[9], a unified framework for deep symbolic regression (uDSR)[38], transformer-based planning for symbolic regression (TPSR)[30] and Operon[39]. For the Nguyen-c dataset, each model is run for at least 20 independent trials with different random seeds, while for all other puzzles, 100 distinct random seeds are utilized. We mark the successful case if the ground-truth equation lies in the discovered Pareto front set.

The SR results of different models, in terms of the symbolic recovery rate and computational time, are depicted in Fig. 3. It can be seen that the proposed PSE method outperforms the baseline methods for all the benchmark problem sets in terms of recovery rate, meanwhile maintaining the high efficiency (for example, expending the minimal computation time) that achieves up to two orders of magnitude speedup. The detailed results for each SR problem set are listed in Supplementary Tables 10–15. An intriguing finding is that PSE achieves an impressive symbolic recovery rate of 100% on the R benchmark expressions, while the baseline models almost fail (for example, recovery rate <2%). We attribute this to the mathematical nature of the R benchmark expressions, which are all rational fractions like $(x+1)^3/(x^2-x+1)$ (R-1). Confined to the sampling interval $x \in [-1, 1]$, the properties of the R expressions bear a high resemblance to polynomial functions, resulting in intractable local minima that essentially lead to the failure of NGGP and DGSR. This issue can be alleviated given a larger interval, for example, $x \in [-10, 10]$, as illustrated in the R* dataset (Supplementary Table 12). Notably, the performance of DGSR based on pre-trained language models stems from the prevalence of polynomial expressions in the pre-training corpora, while NGGP collapses on account of its limited search capacity in an enormously large search space. In contrast, owing to the direct and parallel evaluation of multi-million expression structures, PSE has the capability of accurately and efficiently recovering complex expressions.

Discovery of chaotic dynamics

Nonlinear dynamics is ubiquitous in nature and typically governed by a set of differential equations. Distilling such governing equations from limited observed data plays a crucial role in better understanding the fundamental mechanism of dynamics. Here we test the proposed PSE model to discover a series of multi-dimensional autonomous chaotic dynamics (for example, Lorenz attractor[40] and its variants[41]). The synthetic datasets of these chaotic dynamical systems are described in Supplementary Note 2.2. It is noted that we only sample the noisy trajectories while determining the velocity states via smoothed numerical

differentiation. We compare PSE with eight pivotal baseline models (namely, BMS[9], PySR[37], NGGP[32], DGSR[22], wAIC[16], TPSR[30], uDSR[38] and Operon[39]) and run each model on 50 different random seeds to calculate the average recovery rate for each dataset. For weighted Akaike Information Criterion (wAIC), we employed two distinct basis function configurations: one utilizing polynomial basis functions (wAIC1), and another combining polynomial basis functions with the same unary operator basis used by other algorithms (wAIC2). Considering the noise effect, the criterion for successful equation recovery in this experiment is defined as follows: the discovered Pareto front covers the structure of the ground-truth equation (allowing for a constant bias term). As the dynamics of each system is governed by multiple coupled differential equations (for example, three or four as shown in Supplementary Tables 3–6), the discovery is conducted independently to compute the average recovery rate for each equation, among which the minimum rate is taken to represent each model's capability.

Our main focus herein is to investigate whether SR methods can successfully recover the underlying differential equations given a specific set of token operators under the limit of a short period of computational time (for example, a few minutes). In our experiments, we set the candidate binary operators as +, −, × and ÷, while the candidate unary operators include sin, cos, exp, log, tanh, cosh, abs and sign. Figure 4a,b depicts the results of discovering the closed-form governing equations for 16 chaotic dynamical systems. The experiments demonstrate that under four different levels of Gaussian noise (1%, 2%, 5% and 10% of the data's standard deviation), the proposed PSE approach can achieve a much higher symbolic recovery rate (Fig. 4b), enabling to identify more accurately the underlying governing equations, even under noise effect, to better describe the chaotic behaviors (Fig. 4a). This substantiates the capability and efficiency of our method on data-driven discovery of governing laws for more complex chaotic dynamical systems beyond the Lorenz attractor. More detailed results are listed in Supplementary Tables 16–19.

Electro-mechanical positioning system

Real-world data, replete with intricate noise and nonlinearity, may hinder the efficacy of SR algorithms. To further validate the capability of our PSE model in uncovering the governing equations for real-world dynamical systems (for example, mechanical devices), we test its performance on a set of lab experimental data of an electro-mechanical positioning system (EMPS)[42], as shown in Fig. 5a,b. The EMPS set-up is a standard configuration of a drive system used for prismatic joints in robots or machine tools. Finding the governing equation of such a system is crucial for designing better controllers and optimizing system parameters. The dataset was bifurcated into two even parts, serving as the training and testing sets, respectively. The reference governing equation is given by $M\ddot{q} = -F_v\dot{q} - F_c\mathrm{sign}(\dot{q}) + \tau - c$ (ref. 42), where q, $\dot{q}$ and $\ddot{q}$ represent joint position, velocity and acceleration. Here, τ is the joint torque/force; c, M, F_v and F_c are all constant parameters in the equation. Notably, we leveraged the a priori knowledge that the governing equation of such a system follows the Newton's second law, with a general form $\ddot{q} = f(\dot{q}, q, \tau)$. Our objective is to uncover the closed form of f based on a predefined set of token operators. In EMPS, there exists friction that dissipates the system energy. On the basis of this prior knowledge, we include the sign operator to model such a mechanism. Hence, the candidate math operators we use to test the SR algorithms read {+, −, ×, ÷, sin, cos, exp, log, cosh, tanh, abs, sign}.

We compare our PSE model with eight pivotal baseline models, namely, PySR[37], NGGP[32], DGSR[22], BMS[9], uDSR[38], TPSR[30], wAIC[16] and Operon[39]. We execute each SR model 20 trials on the training dataset to ascertain the Pareto fronts. Subsequently, we select the discovered equation from the candidate expression set of each Pareto front based on the test dataset that shows the highest reward value delineated in equation (6). The reward is designed to balance the prediction error and the complexity of the discovered equation, which is crucial to deriving the governing equation that is not only as consistent with the data as possible but also parsimonious and interpretable. Finally, we select among 20 trials the discovered equation with the median reward value to represent each SR model's average performance. The results demonstrate that our PSE model achieves the best performance in successfully discovering the underlying governing equation (Fig. 5c–f and Supplementary Fig. 3). Our model is capable of uncovering the correct governing equation while maintaining low prediction error (Fig. 5g).

Governing equation of turbulent friction

Uncovering the intrinsic relationship between fluid dynamics and frictional resistance has been an enduring pursuit in the field of fluid mechanics, with implications spanning engineering, physics and industrial applications. In particular, one fundamental challenge lies in finding a unified formula to quantitatively connect the Reynolds number (Re), the relative roughness r/D and the friction factor λ, based on experimental data, where r is the absolute roughness and D is the pipe diameter. The Reynolds number, a dimensionless quantity, captures the balance between inertial and viscous forces within a flowing fluid, which is a key parameter in determining the flow regime, transitioning between laminar and turbulent behaviors. The relative roughness, a ratio between the size of the irregularities and the radius of the pipe, characterizes the interaction between the fluid and the surface it flows over. Such a parameter has a substantial influence on the flow's energy loss and transition to turbulence. The frictional force, arising from the interaction between the fluid and the surface, governs the dissipation of energy and is crucial in determining the efficiency of fluid transport systems. The groundbreaking work[43] dated in the 1930s stepped out the first attempt by meticulously cataloging in the lab the flow behavior under friction effect. The experimental data, commonly referred to as the Nikuradse dataset, offers insights into the complex interplay between these parameters (Re and r/D) and the resultant friction factor (λ) in turbulent flows. We herein test the performance of the proposed PSE model in uncovering the underlying law that governs the relationship between fluid dynamics and frictional resistance based on the Nikuradse dataset.

We first transform the data by a data collapse approach[44], a common practice used in previous studies[45]. Our objective is to find a parsimonious closed-form equation given by $\bar{\lambda} = \bar{h}(x)$, where $\bar{\lambda} = \lambda^{-1/2} + 2\log(r/D)$ denotes the transformed friction factor, $x = \mathrm{Re}\sqrt{\lambda/32}(D/r)$ is an intermediate variable, and $\bar{h}$ is the target function to be discovered. We consider seven baseline models for comparison, namely, PySR[37], NGGP[32], DGSR[22], uDSR[38], TPSR[30], BMS[9] and Operon[39]. The candidate operators used in these models read $\{+, \times, -, \div, \sin, \cos, \exp, \log, \tanh, \cosh, \square^2, \square^3\}$. We run each SR model for 20 independent trials with different random seeds and choose the identified expression with the median reward as the representative result. For each trial, we report the expression with the highest reward (equation (6)) on the discovered Pareto front. Figure 5h illustrates discovered governing equations for turbulent friction. The equations discovered by each SR method and the corresponding prediction error versus model complexity are shown in Fig. 5h–j. While several methods achieve comparable MSE performance, our PSE model stands out by producing the best balance between fitting accuracy and expression simplicity (Fig. 5j). It is evident from the results that our method not only matches or surpasses the fitting performance of other approaches but also generates more concise symbolic expressions.

Scalability in high-dimensional space

To further assess the scalability of PSE and its capability for implicit feature selection, we introduced a challenging high-dimensional synthetic benchmark. This benchmark comprises 20 problems where the ground-truth equations, each involving 12 active variables, are embedded in a 50-dimensional input space, with the remaining 38 variables acting as distractors. In this high-dimensional and noisy environment, PSE showed remarkable robustness by successfully recovering 40% of

https://doi.org/10.1038/s43588-025-00904-8

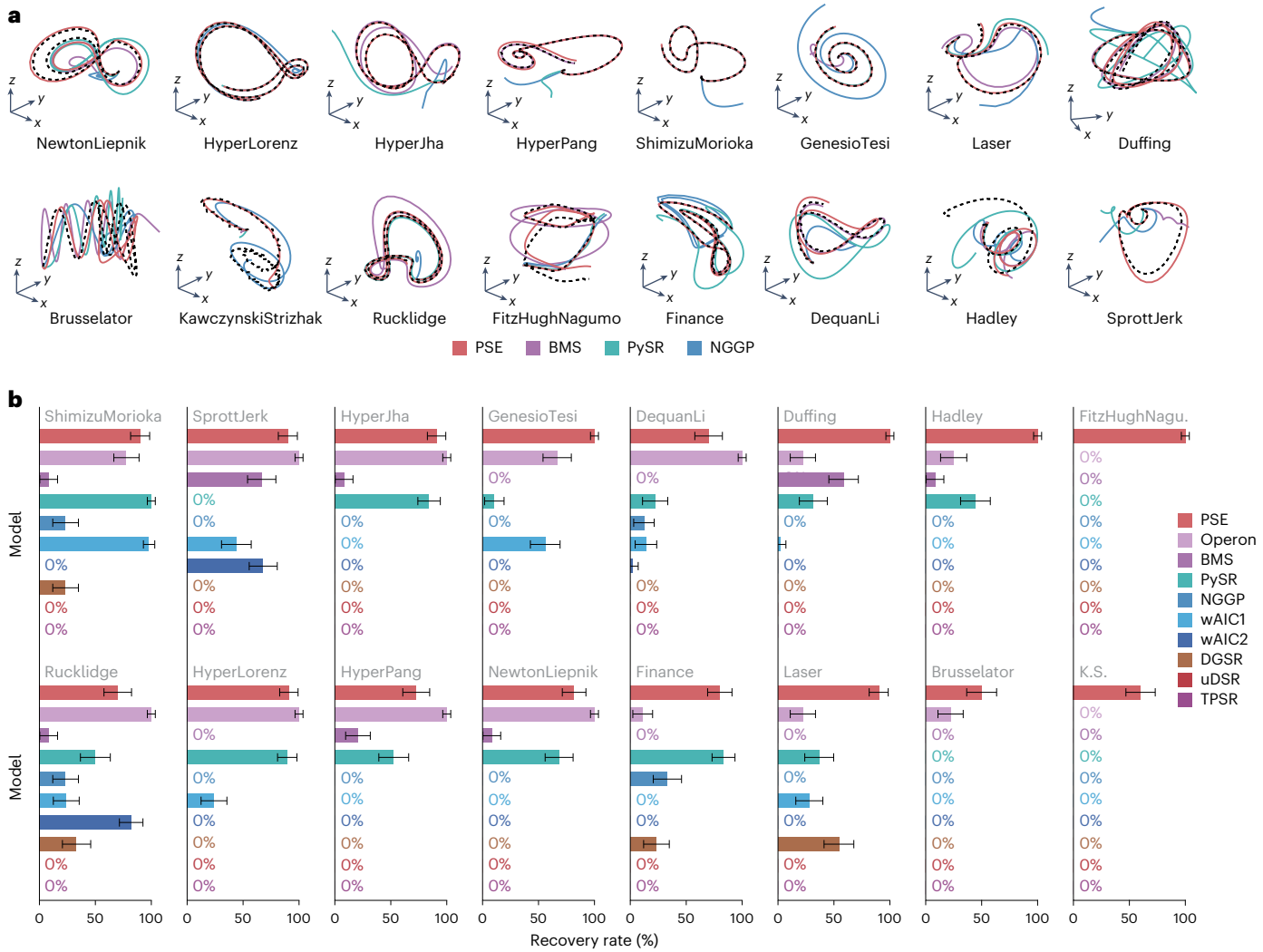


Fig. 4 | Data-driven discovery of nonlinear chaotic dynamics by different models including PSE, BMS, PySR, NGGP, DGSR, wAIC, uDSR, TPSR and Operon. A large number of operators (for example, +, −, ×, ÷, sin, cos, exp, cosh, tanh, abs, sign and so on) are used to simulate the situation in which scientists explore unknown systems in the real world with less prior knowledge about operators. Given the same budget of computational time, PSE has a higher probability of finding the true governing equations among a nearly infinite number of possible expression structures. Each model's runtime is capped at around 10 minutes by limiting the number of iterations. **a**, Examples of predicted trajectories (for example, solid lines) by different models compared with the ground truth (for example, dashed lines). **b**, Average recovery rates of SR algorithms on 16 nonlinear chaotic dynamics datasets. Each bar with the same color represents the recovery rates under a typical case of 2% Gaussian noise. More results under different noise levels (1%, 5% and 10% of the data's standard deviation) can be found in Extended Data Fig. 1 and Supplementary Tables 16–19. The error bars represent the 95% confidence interval of the mean recovery rate for each system. The result is averaged across 50 independent runs.

the true expressions. In stark contrast, leading baselines such as PySR and Operon failed to identify any correct solutions (0% recovery rate) within the same computational time budget. This result highlights PSE's distinct advantage in navigating vast search spaces while effectively discerning relevant variables, a critical capability for tackling complex real-world problems. The detailed description of the methodology, equations and full results can be found in Supplementary Note 7.

Model performance analysis

The performance of the proposed PSE model depends on several factors, including the noise level of the given data, model hyperparameters and whether certain modules are used. Herein, we present an analysis of these factors as well as the model efficiency and the memory footprint.

Model ablation study. We conducted three cases of model ablation study to evaluate the role of certain modules. First, we investigate how much improvement the use of GP or MCTS as the token generator for automatic discovery of admissible tokenized input brings. Second, we conduct a sensitivity analysis of the token constants range. Third, we investigate the extent of the benefit of using DR mask. The results of the ablation experiments are shown in Fig. 6a–c.

First, we evaluate the symbolic recovery rates to compare different token generation methods for PSRN, including random generation, MCTS and GP. The tests are conducted on the entire Nguyen, Nguyen-c, R, R*, Livermore and Feynman problem sets. It can be observed in Fig. 6a that the recovery rate and relative speed diminishes if the token generator is removed from PSE, indicating its vital role in admissible token search that signals the way forward for expression exploration. In addition, we observe that using GP as the token generator leads to faster discovery of ground-truth expressions compared with MCTS, despite their similar symbolic recovery rates. Second, to investigate the sensitivity of our model to the randomly sampled token constants, we set the range to [−1, 1], [−3, 3] and [−10, 10], respectively, and performed the experiments on the Nguyen-c benchmark expressions. The result in Fig. 6b shows that when the range of constant sampling is altered, it substantially affects the time required to find the ground-truth expression,

https://doi.org/10.1038/s43588-025-00904-8

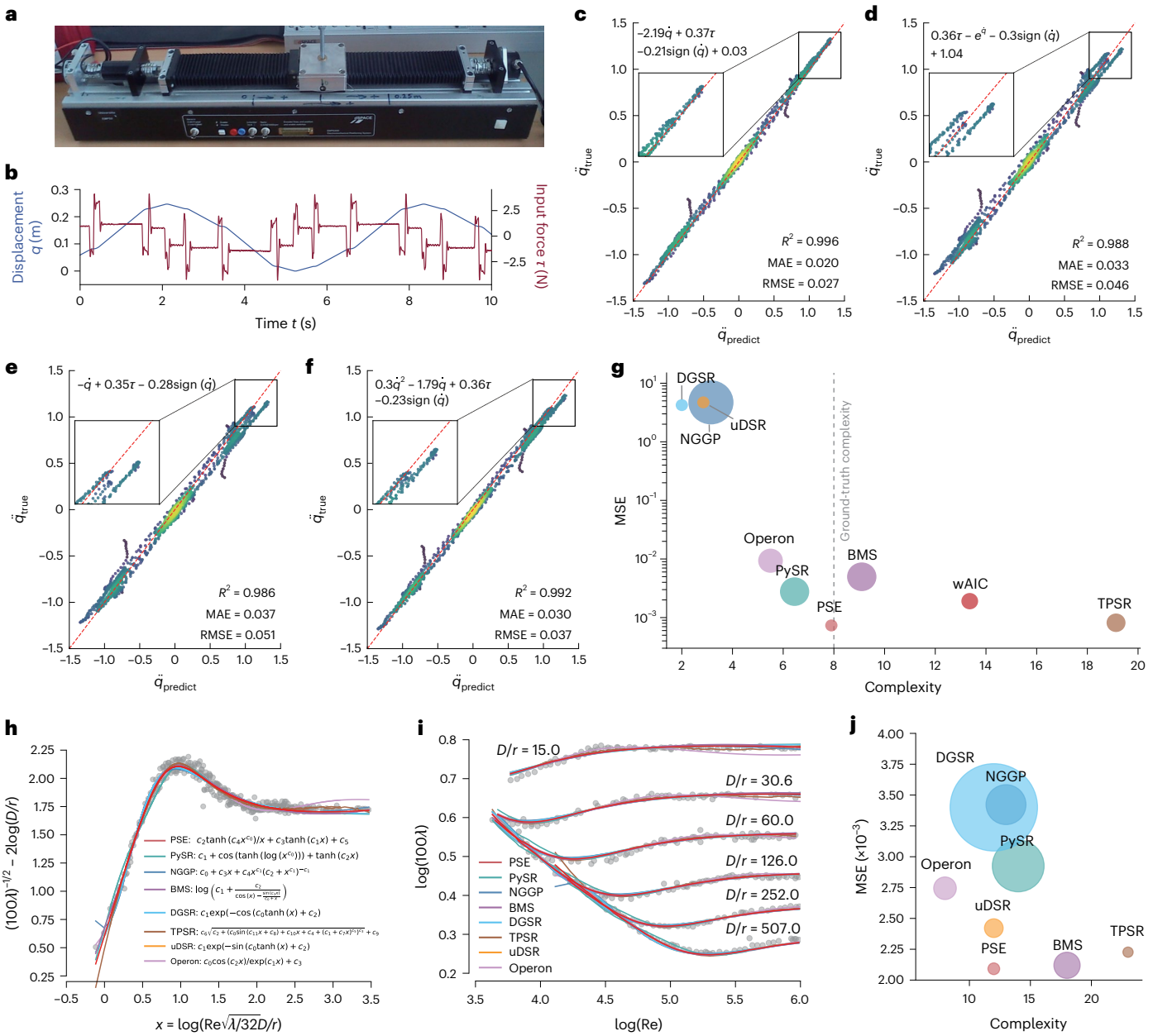


Fig. 5 | Discovering the underlying physical laws with experimental data. a, The set-up of the EMPS experiment. **b**, Collected displacement and input force data. **c**, The prediction performance along with a typical governing equation discovered by PSE. Herein, RMSE stands for root mean square error. **d**, The prediction performance along with a typical governing equation discovered by PySR. **e**, The prediction performance along with a typical governing equation discovered by BMS. **f**, The prediction performance along with a typical governing equation discovered by wAIC. Additional results for other methods on the EMPS experiment can be found in Supplementary Fig. 3. **g**, The prediction error versus model complexity for different SR methods on the EMPS dataset, averaged over 20 independent runs. The circle size indicates MSE uncertainty (interquartile range). **h**, The transformed (or collapsed) Nikuradse's dataset and discovered equations of turbulent friction by different SR methods. The legend shows the median reward models (equation (6)), among 20 trials, obtained by PSE (red), PySR (green)[37], NGGP (blue)[32], DGSR (sky blue)[22], uDSR (orange)[38], TPSR (brown)[30], BMS (purple)[9] and Operon (pink)[39]. **i**, The fitting performance of each SR method on the original Nikuradse's data. **j**, The predicted MSE versus model complexity for different SR methods, averaged over 20 independent runs. The circle size indicates MSE uncertainty (interquartile range). Notably, our PSE method excels at fitting turbulent friction data rapidly (for example, within only 1.5 minutes), meanwhile discovering a more parsimonious and accurate equation.

but has little impact on the symbolic recovery rate. Last but not least, we test the efficacy of the DR mask for memory saving. The result in Fig. 6c illustrates that the DR mask is able to save the graphic memory up to 50%, thus improving the capacity of the operator set (for example, more operators could be included to represent complex expressions).

SRBench evaluation and performance. We further tested the performance of our model on the SRBench problem sets[46], evaluating 133 mathematical expressions on datasets with four levels of Gaussian-type noise (that is, 0, 0.1%, 1% and 10% of the data's standard deviation). The results of symbolic recovery rate, model complexity and computational efficiency for our PSE model in comparison with other 17 baseline models are depicted in Fig. 6d–f. It can be seen that our method outperforms all these existing algorithms in terms of symbolic recovery rate across all noise conditions, while maintaining competitive performance in the context of model complexity and computational efficiency. Note that

https://doi.org/10.1038/s43588-025-00904-8

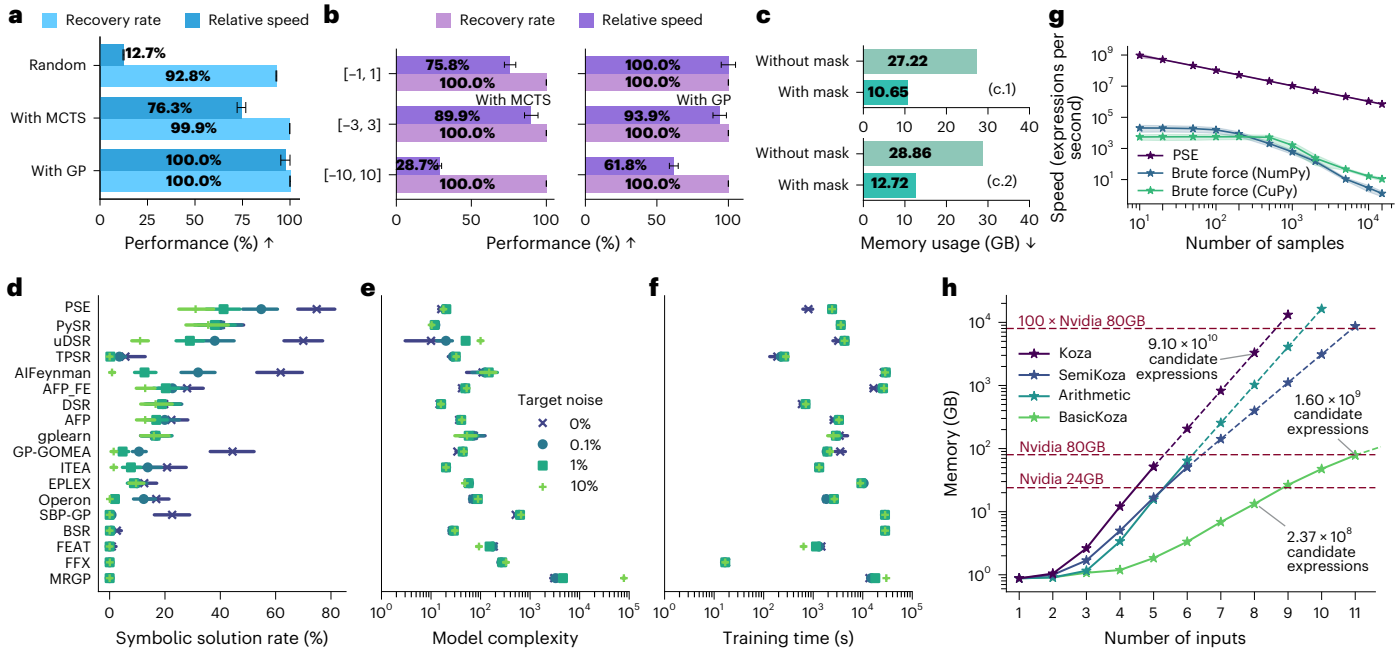


Fig. 6 | Ablation study of the proposed PSE method. a, Comparison of token generators: random, MCTS and GP. **b**, Ablation of token constants range. **c**, Ablation of DR mask on two different PSE configurations: (c.1), a 4-input, 3-layer PSRN with $\mathcal{O}_{\text{Koza}}$ library (for example, {+, ×, −, ÷, identity, sin, cos, exp, log}); and (c.2), a 5-input, 3-layer PSRN with $\mathcal{O}_{\text{SemiKoza}}$ library (for example, {+, ×, SemiSub, SemiDiv, identity, neg, inv, sin, cos, exp, log}). **d**–**f**, SRBench[46] results with GP as the token generator. Our method achieves the highest symbolic recovery rate (**d**) across all noise levels (that is, 0, 0.1%, 1% and 10% of the data's standard deviation) while maintaining competitive performance in model complexity (**e**) and training time (**f**). **g**, Expression search efficiency of the PSRN module. **h**, Space complexity of PSRN with three symbol layers, with respect to the number of input slots and memory footprints. We incorporate four operator sets: $\mathcal{O}_{\text{Koza}}$, $\mathcal{O}_{\text{SemiKoza}}$, $\mathcal{O}_{\text{Arithmetic}}$ (for example, {+, ×, −, ÷, identity}) and $\mathcal{O}_{\text{BasicKoza}}$ (for example, {+, ×, identity, neg, inv, sin, cos, exp, log}) for comparison. With the expansion of the memory footprint, our model demonstrates scalability by evaluating a greater number of candidate expressions in a single forward pass, leading to enhanced performance. We strictly maintain consistent runtime budgets. The variations in the algorithms' runtime occur because different algorithms trigger their respective early stopping conditions (for example, reaching an MSE below a very small threshold, or discovering symbolically equivalent expressions, and so on). The error bars represent the 95% confidence interval of the metrics. The results are averaged across 20 independent runs for each SR problem.

the results were validated using 10 different random seeds following the standard evaluation protocol in ref. 46, with 95% confidence intervals calculated, which demonstrated PSE's robustness to handle real-world noise interference while producing concise expressions with relatively short training time required. These findings underscore the effectiveness of our PSE method.

Expression search efficiency. We compare the efficiency of PSE in the context of evaluation of large-scale candidate expressions, in comparison with two brute-force methods (for example, NumPy[47], which performs central processing unit-based evaluation serially, and CuPy[48], which operates on GPUs in parallel based on batches). Note that PSE has the capability of automatic identification and evaluation of common subtrees, while NumPy and CuPy do not have such a function. Assuming independent input variables $\{x_1, \ldots, x_5\}$ with operators {+, ×, identity, neg, inv, sin, cos, exp, log} for a maximum tree depth of 3, the complete set of generated expressions is denoted by $\hat{\mathcal{F}}$. We consider the computational time required to evaluate the loss values of all the expressions, for example, $||\mathbf{y} - \hat{f}(X)||_2^2$ where $\hat{f} \in \hat{\mathcal{F}}$ denotes one of the candidate expressions, under different sample sizes (for example, 10^1–10^4 data points), with X being the input data matrix and $\mathbf{y}$ the target data.

The result shows that PSE can quickly evaluate all the corresponding expressions, clearly surpassing the brute-force methods (Fig. 6g). When the number of samples is less than 10^4, the search speed of PSE is about 4 orders of magnitude faster, exhibiting an unprecedented increase in efficiency, owing to the reuse of common subtree evaluation in parallel. Notably, when the number of samples is big, downsampling of the data is suggested in the process of uncovering the equation structure to take the speed advantage of PSE during forward propagation. In the coefficient estimation stage, all samples should be used. This could further increase the efficiency of PSE while maintaining accuracy.

Space complexity. We categorize the operators used in PSE into three types: unary, binary-squared and binary-triangled, which are represented by u, b_S, and b_T, respectively. Binary-squared operators represent non-commutative operators (for example, − and ÷) depending on the order of operands, which requires ω_{i-1}^2 space on the GPU during PSRN forward propagation (here, ω_{i-1} represents the input size of the previous symbol layer). Binary-triangled operators represent commutative operators (for example, + and ×) or the memory-saving version of non-commutative operators that only take up $\omega_{i-1}(\omega_{i-1}+1)/2$ space (for example, the SemiSub and SemiDiv symbols, described in 'Symbol layer' in Methods, in the Feynman benchmark that are specifically designed to save memory and support operations in only one direction).

With the number of operators in each category denoted by N_u, N_{b_S} and N_{b_T}, and the number of independent variables and layers of PSE denoted by m and l, respectively, the number of floating-point values required to be stored in PSE can be analyzed. Ignoring the impact of the DR mask, there is a recursive relationship between the tensor dimension of the $(i-1)$th layer (for example, ω_{i-1}) and that of the ith layer (for example, ω_i), namely

$$\omega_i = N_u\omega_{i-1} + N_{b_S}\omega_{i-1}^2 + N_{b_T}\omega_{i-1}\frac{\omega_{i-1}+1}{2} \le \kappa\omega_{i-1}^2,$$

where $\kappa = N_u + N_{b_T} + N_{b_S}$. Thus the complexity of the number of floating-point values required to be stored by an l-layer PSRN can be calculated as $O(\kappa^{2^l-1}\omega_0^{2^l})$, where ω_0 represents the number of input slots.

Clearly, the memory consumption of PSRN increases dramatically with the number of layers. If each subtree value is a single-precision floating-point number (for example, 32-bit), with the input dimension of 20 and operator set {+, −, ×, ÷, identity, sin, cos, exp, log}, the required memory will reach over 10^5 GB when the number of layers is 3, which requires a large compute set. Hence, finding a new strategy to relax the space complexity and alleviate the memory requirement is needed to further scale up the proposed model. Figure 6h illustrates the graphic-memory footprint of various three-layered PSRN architectures, each characterized by a different operator set and the number of input slots. While the rise in memory demands serves as a constraint, this escalation is directly tied to the scalable model's ability to evaluate a greater number of candidate expressions within a single forward pass. This result also shows that PSRN follows the scaling law (for example, the model capacity and size scale with the number of token inputs, given the fixed number of layers). The detailed hardware settings are found in Supplementary Note 2.5.

Discussion

The development of PSE represents a fundamentally different scheme in the search for symbolic models. By shifting the paradigm from independent, sequential evaluation of candidate expressions to a parallelized, shared-computation framework, PSE directly addresses a long-standing efficiency–accuracy bottleneck in SR. This distinction is particularly evident when compared with explicit enumeration methods such as exhaustive SR[49], which, while complete, face a severe computational barrier by constructing and evaluating every possible expression—a process that is both time-consuming and practically limited to simple, often single-variable, problems (Supplementary Note 9). In contrast, the ability of PSE to recognize and exploit common subtrees enables a large-scale parallel enumeration that opens a path toward solving more complex and higher-dimensional discovery problems. When coupled with a token generator (for example, GP or MCTS), the model's ability to delve into intricate expressions is further magnified, showing potential to accelerate data-driven discovery across scientific domains.

Despite its performance, the PSE model faces several challenges that need to be addressed in the future. One of the primary challenges is the rapidly increasing demand for memory in the PSRN module as the number of symbol layers increases (see 'Space complexity' above). Currently, a brute-force implementation of PSRN can only directly handle expressions with an expression tree depth of ≤3 under conventional settings. Otherwise, the method relies on the token generator (for example, GP or MCTS) to extend the PSRN's capacity by providing more complex sub-expressions as tokenized inputs. This bottleneck impedes the PSE's exploration of much deeper expressions, especially for problems requiring extremely deep, monolithic expressions that cannot be easily decomposed into smaller tokens. As detailed in our failure analysis (Supplementary Note 8), this is because the token generator itself may struggle to construct a highly complex sub-expression as a single, coherent unit, leading the search to become trapped in local optima of simpler, 'good enough' solutions, as the marginal accuracy gain from the true, highly complex structure may not provide a sufficient reward signal to justify the difficult evolutionary leap. In addition, our current two-stage approach to handling constants—sampling token constants and then refining them via least squares—is effective but not without limitations. If the true coefficients lie far outside the initial sampling range, the least squares refinement may fail to find the global optimum.

Furthermore, like all data-driven methods, the performance of the PSE is inherently linked to the quality and nature of the data. For instance, its performance degrades under high noise levels, converging to simpler, over-regularized expressions that model the noise rather than the underlying signal. This is related to the PSRN's strong affinity for fractional forms; while this is a notable advantage on certain benchmarks, under high noise it could lead to overfitting by modeling noise as a complex rational function (Supplementary Note 8). Similarly, the model may struggle with deceptive fitness landscapes, where a simple, incorrect expression has a similar MSE to the true, more complex one over the given data domain. In such cases, the algorithm's success hinges on the data being sufficiently informative to distinguish the true model from plausible alternatives. In terms of practical constraints, while the method can operate on various computing platforms, its full acceleration is best realized on hardware that supports massive parallelism, although the subtree reusing benefit remains hardware independent. Like most SR methods, the PSE is also constrained by its predefined, discrete operator set and cannot discover novel mathematical forms without being explicitly extended.

Looking ahead, there are several promising avenues for further optimization and enhancement. For instance, ensemble SR, namely, uDSR[38], integrates a wide array of techniques including large-scale pre-training[22], deep SR[17], sparse regression[11] and AIFeynman[19], and has shown improved SR performance. This suggests substantial potential for enhancing our approach by synthesizing PSE with other SR techniques. Similarly, integrating explicit feature selection methods as a pre-processing step could further focus the search on the most relevant variables, representing another promising avenue for enhancing scalability on extremely large-scale problems. The PSRN framework itself also offers opportunities for efficiency improvements, such as adopting more advanced computational backends that show superior performance in speed and memory utilization. This includes developing more sophisticated token generation strategies, as suggested by our failure analysis (Supplementary Note 8), such as incorporating domain-specific structural priors or employing multi-objective rewards that explicitly value structural novelty. Finally, ongoing work is exploring the integration of prior knowledge, such as the dimensional constraints of physical quantities, to guide the search. This approach promises not only to conserve computational resources but also to expedite the discovery process. We intend to continue addressing these challenges methodically in our forthcoming research.

Methods

The NP-hardness of SR, noted in existing studies[17,19,32], was formally proven recently[33]. This implies the absence of a known solution that can be determined in polynomial time for solving SR problems across all instances, and reflects that the search space for SR is vast and intricate. Almost all the existing SR methods involve a common procedure, which is to assess the quality of candidate expressions $\hat{\mathcal{F}} = \{\hat{f}_1, \hat{f}_2, \ldots\}$ based on the given data $\mathcal{D} = (X, \mathbf{y})$, where $X \in \mathbb{R}^{n\times m}$ and $\mathbf{y} \in \mathbb{R}^{n\times 1}$. Here, $\hat{f}$ is usually constructed by a given operator set $\mathcal{O}$ (for example, {+, ×, −, ÷, sin, cos, exp, log}). Typically, this evaluation is guided by the MSE expressed as:

$$\mathrm{MSE} = \frac{1}{n}||\hat{f}(X) - \mathbf{y}||_2^2. \quad (1)$$

During the search process, a large number of candidate expressions need to be evaluated, while the available operators and variables are limited, leading to the inevitable existence of vast repeated sub-expressions. Existing methods evaluate candidate expressions sequentially and independently. As a result, the common intermediate expressions are repeatedly calculated, leading to considerable computational burden (Supplementary Fig. 1). By reducing the amount of repeated computation, the search process can be substantially accelerated.

To address these challenges and enhance both the efficiency and accuracy of SR, we propose a PSE model, as shown in Fig. 1, to automatically discover mathematical expressions from limited data. The core

of PSE is a PSRN regressor, depicted in Fig. 1b,c, designed to enhance the efficiency of candidate expression evaluation. The architecture of PSRN is inherently parallelism-friendly, enabling rapid GPU-based parallel computation (Fig. 1c). A key innovation of PSRN is its ability to automatically identify and reuse the intermediate calculations of common subtrees (Fig. 1d), thus avoiding redundant computations.

To further bolster the model's discovery capability for more complex expressions, we integrate a token generator (for example, MCTS or GP) that identifies a set of admissible base expressions as tokenized input to the PSRN (Fig. 1a). The PSE model operates iteratively: PSRN continually activates the token generator, starting with a base expression set that includes available independent variables. During these iterations, this set of base expressions is updated by progressively incorporating more complex expressions generated by the token generator. These base expressions, along with sampled token constants, are fed into PSRN for evaluation and the search of optimal symbolic expressions (Fig. 1b). The PSRN can rapidly (for example, within seconds) evaluate hundreds of millions of candidate symbolic expressions, identifying the one with the smallest error or a few best candidates. This represents a substantial speed improvement compared with approaches that evaluate each candidate expression independently. Essentially, PSRN performs a large-scale and PSE of expression trees with scalability. In addition, a duplicate removal mask (DR mask) is designed for PSRN to reduce memory usage (Fig. 1e). Within the PSRN regressor, the coefficients of the most promising expressions are identified and fine-tuned using least squares estimation (Fig. 1f). Rewards computed on the given data are then back-propagated for subsequent token generator updates. As the search progresses, the Pareto front representing the optimal set of expressions is continuously updated to report the final result.

Symbol layer

A mathematical expression can be equivalently represented as an expression tree[8], where the internal nodes denote mathematical operators and the leaf nodes the variables and constants. The crux of the aforementioned computational issue lies in the absence of temporary storage for subtree values and parallel evaluation. Consequently, the common subtrees in different candidate expressions are repeatedly evaluated, resulting in considerable computational wastage.

To this end, we introduce the concept of symbol layer (Fig. 1c), which consists of a series of mathematical operators. The symbol layer serves to transform the computation results of shallow expression trees into deeper ones. From the perspective of avoiding redundant computations, the results of the symbol layer are reused by expression trees with greater heights. From the view of parallel computation, the symbol layer can leverage the power of GPU to compute the results of common subtrees in parallel, improving the overall speed (Supplementary Fig. 1). We categorize the mathematical operators in the symbol layer into three types: (1) unary operators (for example, sin and exp); (2) binary-squared operators (for example, − and ÷), representing non-commutative operators; and (3) binary-triangled operators, representing commutative operators (for example, + and ×) or a variant of non-commutative operators with low-memory footprint (for example, SemiSub and SemiDiv that output $x_i - x_j$ and $x_i \div x_j$ for $i \le j$, respectively). Note that the binary-triangled operators only take up half of the space compared with the binary-squared operators. We denote these three types of operator as u, b_S and b_T, respectively. Mathematically, a symbol layer located at the lth level can be represented as follows:

$$\mathbf{h}^{(l)} = \left(\mathop{\Vert}_{i=1}^{N_u} \mathbf{u}_i\left(\mathbf{h}^{(l-1)}\right)\right) \Vert \left(\mathop{\Vert}_{i=1}^{N_{b_S}} \mathbf{b}_{S_i}\left(\mathbf{h}^{(l-1)}\right)\right) \Vert \left(\mathop{\Vert}_{i=1}^{N_{b_T}} \mathbf{b}_{T_i}\left(\mathbf{h}^{(l-1)}\right)\right), \tag{2}$$

where

$$\mathbf{u}(\mathbf{h}) = \mathop{\Vert}_{i} u(h_i),\ \mathbf{b}_S(\mathbf{h}) = \mathop{\Vert}_{i,j} b_S(h_i, h_j), \mathbf{b}_T(\mathbf{h}) = \mathop{\Vert}_{i \le j} b_T(h_i, h_j). \tag{3}$$

Here, $\Vert$ represents the concatenation operation; N_u, N_{b_S} and N_{b_T} denote the numbers of unary operators, binary-squared operators and binary-triangled operators.

For example, let us consider independent variables $\{x_1, x_2\}$ and dependent variable z, with the task of finding an expression that satisfies $z = f(x_1, x_2)$. When the independent variables $\{x_1, x_2\}$ are fed into a symbol layer, we obtain the combinatorial results (for example, $\{x_1, x_2, \sin(x_1), \sin(x_2), \ldots, x_1 + x_1, x_1 + x_2, x_2 + x_2, \ldots\}$). Notably, each value in the output tensor of the symbol layer corresponds to a distinct sub-expression. This enables PSE to compute the common subtree values just once, avoiding redundant calculations. In addition, the symbol layer can be leveraged on the GPU for parallel computation, further enhancing the speed of expression searches.

When establishing a symbol layer initially, an inherent offset tensor Θ is inferred and stored within the layer for subsequent lazy symbolic deducing during the backward pass (Fig. 2). For the lth symbol layer, its offset tensor can be represented as follows:

$$\Theta^{(l)} = \left(\mathop{\Vert}_{i=1}^{N_u} \Theta_u\right) \Vert \left(\mathop{\Vert}_{i=1}^{N_{b_S}} \Theta_{b_S}\right) \Vert \left(\mathop{\Vert}_{i=1}^{N_{b_T}} \Theta_{b_T}\right), \tag{4}$$

where

$$\Theta_u = \begin{bmatrix} 1 \cdots \omega \\ \varnothing \cdots \varnothing \end{bmatrix}_{2\times\omega}, \tag{5a}$$

$$\Theta_{b_S} = \mathop{\Vert}_{j=1}^{\omega} \begin{bmatrix} 1 & \cdots & \omega \\ j & \cdots & j \end{bmatrix}_{2\times\omega^2}, \tag{5b}$$

$$\Theta_{b_T} = \mathop{\Vert}_{j=1}^{\omega} \begin{bmatrix} 1 & \cdots & \omega - j + 1 \\ j & \cdots & j \end{bmatrix}_{2\times\frac{\omega(\omega+1)}{2}}. \tag{5c}$$

Here, ω denotes the output dimension (the number of symbolic expression trees) of the previous layer.

Each position in the output tensor $\mathbf{h}^{(l)}$ of the symbol layer corresponds to a unique column in the offset tensor $\Theta^{(l)}$. In other words, it corresponds to two child indices (for unary operators, only one), representing the left and right child nodes of the current output tensor position from the previous layer. These offset tensors are used for recursive backward symbol deduction after the position of a minimum MSE value is found.

Parallel SR network

By stacking multiple symbol layers, we construct a PSRN, denoted as ψ_{PSRN} (Fig. 1c). PSRN takes a set of N_{in} input base expressions $s = \{s_1, s_2, \ldots, s_{N_{\text{in}}}\}$ (which can be raw variables, constants, or more complex tokens generated as described in 'PSRN regressor with token generator') and their corresponding data tensor X. It then performs a rapid, parallel forward computation on the GPU. Based on the predefined operator set $\mathcal{O}$, PSRN efficiently evaluates a vast number of distinct candidate expressions by systematically combining its inputs through its layers. For instance, with l symbol layers, PSRN can generate and evaluate all expressions representable as expression trees of depth up to l using the provided base expressions s as leaf nodes. A key advantage is that PSRN computes the values of common subtrees only once, avoiding redundant calculations and substantially speeding up the evaluation of potentially hundreds of millions of expressions.

Once the final layer's computations are complete, PSRN outputs the values $\hat{\mathcal{F}}(X) = \{\hat{f}_1(X), \hat{f}_2(X), \ldots\}$ for all generated candidate expressions. These are then used to compute the MSE (equation (1)) against the target data $\mathbf{y}$, allowing the identification of the expression(s) with the minimum error. The pre-generated offset tensors Θ within each symbol layer are then used to recursively reconstruct the symbolic form of the

optimal candidate expression(s) (Fig. 2). The detailed steps for PSRN's evaluation and symbolic derivation are provided in Supplementary Algorithm 1.

The choice of operator set and the number of input slots for PSRN can be adapted based on the problem complexity and available GPU memory. In the SR benchmark tasks, we employ a 5-input PSRN with the operator set $\mathcal{O}_{\text{Koza}} = \{+, \times, -, \div, \text{identity}, \sin, \cos, \exp, \log\}$, except for the Feynman expression set which uses a 6-input PSRN with the operator set $\mathcal{O}_{\text{SemiKoza}} = \{+, \times, \text{SemiSub}, \text{SemiDiv}, \text{identity}, \text{neg}, \text{inv}, \sin, \cos, \exp, \log\}$. Such a setting aims to conserve GPU memory for handling more input variables. The distinguishing feature of the $\mathcal{O}_{\text{SemiKoza}}$ operator set is that it treats division and subtraction as binary-triangled operators and allows only one direction of operation (see 'Symbol layer'). This trade-off reduces expressive power but conserves GPU memory, which enables to tackle larger scale SR tasks. To discover more deeply nested or complex expressions, PSRN is integrated with a token generator, as described next.

PSRN regressor with token generator

While PSRN can evaluate expressions up to a depth of l (number of symbol layers) from its direct inputs in a single pass, discovering more complex real-world equations often requires exploring deeper expression trees. To extend PSRN's reach and enable the discovery of such intricate symbolic expressions, we integrate the PSRN into a larger iterative framework driven by a token generator, π. This system operates through a synergistic loop. In each iteration, the token generator first proposes a new set of promising base expressions (tokens), denoted as s_T. Subsequently, the PSRN regressor $\mathcal{R}$ takes s_T as input and performs a rapid, large-scale parallel enumeration to evaluate millions of candidate expressions, identifying the top-k expressions, $\{\hat{f}_1, \ldots, \hat{f}_k\}$, that best fit the data. A reward signal is then computed based on the performance (for example, accuracy and complexity) of these discovered expressions. Finally, this reward is fed back to the token generator π, guiding its strategy for generating even better tokens in the subsequent iteration. This iterative process allows PSE to progressively build and refine complex equations that would be unreachable by a single pass of PSRN alone. The overall PSE procedure is detailed in Supplementary Algorithm 2.

Token generation strategies. The token generator, π, is a modular component responsible for exploring the vast space of possible sub-expressions. It is initialized with a base set of operators $\mathcal{O}$ and variables $\mathcal{V}$. In each iteration, it produces a token set, s_T, which can include raw variables, numerical constants and composite expressions (for example, $\{x_1, x_2, x_1^2 + x_2, \frac{x_1}{\exp(x_2)}, 1.3, 2.9\}$) built from previously successful tokens. In this work, we explore several strategies for π. One approach is GP, which maintains and evolves a population of expression trees using operations like crossover and mutation to generate new tokens. The GP-based token generation approach is outlined in Supplementary Algorithm 3. Another method is MCTS, which builds a search tree over the expression space and uses simulation-based feedback to explore promising branches. Our MCTS token generator is detailed in Supplementary Algorithm 4. We also use a random generation strategy as a baseline, which simply combines operators and existing tokens at random. The random token generation strategy is presented in Supplementary Algorithm 5. A detailed comparison of these strategies is presented in our ablation study ('Model ablation study'). As the GP-based generator was found to be the most effective, it is used as the default configuration in our primary experiments.

Reward calculation and Pareto front. To guide the search, each candidate expression $\hat{f}$ is assigned a reward r that balances its accuracy and complexity:

$$r = \frac{\eta^{\alpha}}{1 + \sqrt{\text{MSE}}}, \tag{6}$$

where η is a discount factor (default 0.99) penalizing complexity, and α is the expression's complexity. The set of non-dominated solutions—expressions for which no other expression is better in both accuracy and simplicity—is maintained on a Pareto front. This front is updated in each iteration and represents the final output of the algorithm. The maximum reward achieved is used as the feedback signal for the token generator π.

Coefficients tuning

Many physical laws and mathematical expressions involve numerical coefficients. PSE incorporates a two-stage approach to handle these constants effectively.

Initial constant representation in PSRN. PSRN can incorporate numerical constants during its expression generation phase. As shown in Figs. 1a–b and 2, a predefined number of PSRN input slots can be reserved for token constants. In each iteration, before PSRN's forward propagation, constant values c are sampled from a pre-selected distribution p (for example, $c \sim U(-1, 1)$, $c \sim U(-3, 3)$, where $\sim$ denotes sampled from a distribution) and fed into these constant slots. PSRN can then use these explicit numerical values directly or combine them with operators (for example, 1.3 + 2.9, exp(1.3), 1.3/sin(2.9)) to form more diverse embedded constants within the candidate expressions $\hat{\mathcal{F}}$ it generates. This allows PSRN to explore structures that inherently include numerical values.

Post-PSRN coefficients refinement. After PSRN identifies a set of promising candidate symbolic structures $\hat{\mathcal{F}} = \{\hat{f_1}, \ldots, \hat{f_k}\}$ (which may contain the initially sampled or derived constants), a more precise coefficient optimization is performed (step 4 in Fig. 2). For each selected raw expression $\hat{f}_i$, we parse it to identify all numerical coefficient positions. These coefficients are then treated as free parameters and are collectively optimized using a standard least squares (LS) method against the entire training dataset $(X, \mathbf{y})$:

$$\hat{f}_i^* = \text{LS}\left(\hat{f}_i, X, \mathbf{y}\right), \; i = 1, \ldots, k. \tag{7}$$

This LS step fine-tunes the initial constant values found or incorporated by PSRN, leading to the final optimized expressions $\hat{\mathcal{F}}^* = \{\hat{f}_1^*, \ldots, \hat{f}_k^*\}$. This ensures that the numerical values in the discovered equations are optimally fitted to the data, given the symbolic structure. While we employ the LS method, this refinement step can be generalized to the maximum likelihood estimation for more complex noise models or maximum a posteriori optimization to incorporate regularization. During the warm-up phase of PSE, a simple linear regression step using the independent variables is also performed to expedite the discovery of simpler expressions that might involve basic coefficient fitting.

DR mask

One major limitation of standard PSRN lies in its high demand for graphics memory. When using binary operators such as addition and multiplication, the graphics memory required for each layer grows quadratically with respect to the tensor dimensions of the previous layer. Therefore, reducing the output dimension of the penultimate layer can reduce the required graphics memory. We propose a technique called duplicate removal mask (DR Mask) as shown in Fig. 1e. Before the last PSRN layer, we use the DR mask to remove repeated terms and generate a dense input set. Specifically, assuming the input variables $x_1, x_2, \ldots, x_m$ are independent, we extract the output expressions from the last second layer, parse them using the symbolic computing library SymPy, and remove duplicate expressions to obtain a mask of unique expressions. SymPy's efficient hash value computation facilitates the comparison of mathematical expressions for symbolic equivalence. Typically, this process takes less than a minute. Once the DR mask is identified, it is reusable for PSRNs with the same

architecture (for example, total number of network layers and operator sets). During the search process, the expressions marked by the mask are extracted from the penultimate layer's output tensor and then passed to the final layer for the most graphics-intensive binary operator computations. The percentage of graphics memory saved by the DR mask technique depends on the input size, the number of layers and the operators used in the PSRN architecture. Detailed results are shown in 'Model ablation study'.

Baseline models

We consider nine main baseline SR algorithms used in the comparison analysis: DGSR[22], NGGP[32], PySR[37], BMS[9], SPL[27], wAIC[16], uDSR[38], TPSR[30] and Operon[39]. In addition, we compare with exhaustive SR[49] on single-variable problems only, as it does not support multivariate regression. They represent a selection of SR algorithms, ranging from recent advancements to established, powerful methods. The introduction and detailed settings of the baseline models are found in Supplementary Note 2.3.

Data availability

All the datasets used to test the methods in this study are available at https://github.com/intell-sci-comput/PSE (also see the Zenodo repository at https://doi.org/10.5281/zenodo.17266354 (ref. 50)). Details of our complete data generation methodology are presented in Supplementary Note 2. Source data are provided with this paper.

Code availability

All the source codes used to reproduce the results in this study are available under an MIT license at https://github.com/intell-sci-comput/PSE (also see the Zenodo repository at https://doi.org/10.5281/zenodo.17266354 (ref. 50)).

References

1. Schmidt, M. & Lipson, H. Distilling free-form natural laws from experimental data. *Science* **324**, 81–85 (2009).
2. Wadekar, D. et al. Augmenting astrophysical scaling relations with machine learning: application to reducing the Sunyaev–Zeldovich flux–mass scatter. *Proc. Natl Acad. Sci. USA* **120**, e2202074120 (2023).
3. Li, Y. et al. Electron transfer rules of minerals under pressure informed by machine learning. *Nat. Commun.* **14**, 1815 (2023).
4. Häfner, D., Gemmrich, J. & Jochum, M. Machine-guided discovery of a real-world rogue wave model. *Proc. Natl Acad. Sci. USA* **120**, e2306275120 (2023).
5. Chong, Y. et al. Machine learning of spectra-property relationship for imperfect and small chemistry data. *Proc. Natl Acad. Sci. USA* **120**, e2220789120 (2023).
6. Jung, H. et al. Machine-guided path sampling to discover mechanisms of molecular self-organization. *Nat. Comput. Sci.* **3**, 334–345 (2023).
7. Hosmer, D. W. Jr, Lemeshow, S. & Sturdivant, R. X. *Applied Logistic Regression* Vol. 398 (John Wiley & Sons, 2013).
8. Forrest, S. Genetic algorithms: principles of natural selection applied to computation. *Science* **261**, 872–878 (1993).
9. Guimerà, R. et al. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Sci. Adv.* **6**, eaav6971 (2020).
10. Ly, D. L. & Lipson, H. Learning symbolic representations of hybrid dynamical systems. *J. Mach. Learn. Res.* **13**, 3585–3618 (2012).
11. 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* **113**, 3932–3937 (2016).
12. Rudy, S. H., Brunton, S. L., Proctor, J. L. & Kutz, J. N. Data-driven discovery of partial differential equations. *Sci. Adv.* **3**, e1602614 (2017).
13. Chen, Z., Liu, Y. & Sun, H. Physics-informed learning of governing equations from scarce data. *Nat. Commun.* **12**, 6136 (2021).
14. Sun, F., Liu, Y. & Sun, H. Physics-informed spline learning for nonlinear dynamics discovery. In *Proc. 13th International Joint Conference on Artificial Intelligence* (ed. Zhou, Z.-H.) 2454–2461 (IJCAI Organization, 2021).
15. Rao, C. et al. Encoding physics to learn reaction–diffusion processes. *Nat. Mach. Intell.* **5**, 765–779 (2023).
16. Gao, T.-T. & Yan, G. Autonomous inference of complex network dynamics from incomplete and noisy data. *Nat. Comput. Sci.* **2**, 160–168 (2022).
17. Petersen, B. K. et al. Deep symbolic regression: recovering mathematical expressions from data via risk-seeking policy gradients. In *The 9th International Conference on Learning Representations* (2021).
18. Kusner, M. J., Paige, B. & Hernández-Lobato, J. M. Grammar variational autoencoder. In *Proc. 34th International Conference on Machine Learning* (ed. Precup, D.) 1945–1954 (Proceedings of Machine Learning Research, 2017).
19. Udrescu, S.-M. & Tegmark, M. AI Feynman: a physics-inspired method for symbolic regression. *Sci. Adv.* **6**, eaay2631 (2020).
20. Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A. & Parascandolo, G. Neural symbolic regression that scales. In *Proc. 38th International Conference on Machine Learning* Vol. 139 (eds Meila, M. & Zhang, T.) 936–945 (Proceedings of Machine Learning Research, 2021).
21. Kamienny, P.-A., d'Ascoli, S., Lample, G. & Charton, F. End-to-end symbolic regression with transformers. *Adv. Neural Inf. Process. Syst.* **35**, 10269–10281 (2022).
22. Holt, S., Qian, Z. & van der Schaar, M. Deep generative symbolic regression. In *The 11th International Conference on Learning Representations* (2023).
23. Sahoo, S., Lampert, C. & Martius, G. Learning equations for extrapolation and control. In *Proc. 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) 4442–4450 (Proceedings of Machine Learning Research, 2018).
24. Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games* (eds van den Herik, H. J. et al.) 72–83 (Springer, 2006).
25. Kocsis, L. & Szepesvári, C. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning* (eds Fürnkranz, J. et al.) 282–293 (Springer, 2006).
26. Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**, 1140–1144 (2018).
27. Sun, F., Liu, Y., Wang, J.-X. & Sun, H. Symbolic physics learner: discovering governing equations via Monte Carlo tree search. In *The 11th International Conference on Learning Representations* (2023).
28. Kamienny, P.-A., Lample, G., Lamprier, S. & Virgolin, M. Deep generative symbolic regression with Monte-Carlo-tree-search. In *Proc. 40th International Conference on Machine Learning* (eds Brunskill, E. & Cho, K.) 15655–15668 (Proceedings of Machine Learning Research, 2023).
29. Xu, Y., Liu, Y. & Sun, H. Reinforcement symbolic regression machine. In *The 12th International Conference on Learning Representations* (2024).
30. Shojaee, P., Meidani, K., Farimani, A. B. & Reddy, C. Transformer-based planning for symbolic regression. *Adv. Neural Inf. Process. Syst.* **36**, 45907–45919 (2023).

31. Li, Y. et al. Discovering mathematical formulas from data via gpt-guided monte carlo tree search. *Expert Syst. Appl.* **281**, 127591 (2025).
32. Mundhenk, T. et al. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. *Adv. Neural Inf. Process. Syst.* **34**, 24912–24923 (2021).
33. Virgolin, M. & Pissis, S. P. Symbolic regression is NP-hard. *Trans. Mach. Learn. Res.* https://openreview.net/forum?id=LTiaPxqe2e (2022).
34. Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I. & Galván-López, E. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program. Evolvable Mach.* **12**, 91–119 (2011).
35. McDermott, J. et al. Genetic programming needs better benchmarks. In *Proc. 14th Annual Conference on Genetic and Evolutionary Computation* (eds Soule, T. & Auger, A.) 791–798 (Association for Computing Machinery, 2012).
36. Krawiec, K. & Pawlak, T. Approximating geometric crossover by semantic backpropagation. In *Proc. 15th Annual Conference on Genetic and Evolutionary Computation* (ed. Blum, C.) 941–948 (Association for Computing Machinery, 2013).
37. Cranmer, M. Interpretable machine learning for science with PySR and SymbolicRegression.jl. Preprint at https://arxiv.org/abs/2305.01582 (2023).
38. Landajuela, M. et al. A unified framework for deep symbolic regression. *Adv. Neural Inf. Process. Syst.* **35**, 33985–33998 (2022).
39. Burlacu, B., Kronberger, G. & Kommenda, M. Operon c++: an efficient genetic programming framework for symbolic regression. In *Proc. 2020 Genetic and Evolutionary Computation Conference Companion* 1562–1570 (Association for Computing Machinery, 2020).
40. Lorenz, E. N. Deterministic nonperiodic flow. *J. Atmos. Sci.* **20**, 130–141 (1963).
41. Gilpin, W. Chaos as an interpretable benchmark for forecasting and data-driven modelling. In *The 35th Conference on Neural Information Processing Systems Track on Datasets and Benchmarks* (Curran Associates, 2021).
42. Janot, A., Gautier, M. & Brunot, M. Data set and reference models of EMPS. In *Workshop on Nonlinear System Identification Benchmarks* (2019).
43. Nikuradse, J. et al. *Laws of Flow in Rough Pipes* Technical Report No. NACA-TM-1292 (1950); https://ntrs.nasa.gov/citations/19930093938
44. Prandtl, L. *Recent Results of Turbulence Research* Technical Report No. NACA-TM-720 (1933); https://ntrs.nasa.gov/citations/19930094697
45. Goldenfeld, N. Roughness-induced critical phenomena in a turbulent flow. *Phys. Rev. Lett.* **96**, 044503 (2006).
46. La Cava, W. et al. Contemporary symbolic regression methods and their relative performance. *Adv. Neural Inf. Process. Syst.* **2021**, 1–16 (2021).
47. Harris, C. R. et al. Array programming with NumPy. *Nature* **585**, 357–362 (2020).
48. Okuta, R., Unno, Y., Nishino, D., Hido, S. & Loomis, C. CuPy: a NumPy-compatible library for NVIDIA GPU calculations. In *Proc. 31st Conference on Neural Information Processing Systems* (eds von Luxburg, U. & Guyon, I.) 933–940 (Curran Associates, 2017).
49. Bartlett, D. J., Desmond, H. & Ferreira, P. G. Exhaustive symbolic regression. *IEEE Trans. Evol. Comput.* **28**, 950–964 (2024).
50. Kai, K. Discovering physical laws with parallel symbolic enumeration. *zenodo* https://doi.org/10.5281/zenodo.17266354 (2025).

Acknowledgements

The work is supported by the National Natural Science Foundation of China (No. 92270118, No. 62276269), the Beijing Natural Science Foundation (No. 1232009), and the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDB0620103). H.S. and Y.L. to acknowledge the support from the Fundamental Research Funds for the Central Universities (No. 202230265 and No. E2EG2202X2).

Author contributions

K.R., H.S. and Y.L. contributed to the ideation and design of the research. K.R. performed the research. H.S. and J.-R.W. supervised the project. All authors contributed to the research discussions, writing and editing of the paper.

Competing interests

The authors declare no competing interests.

Additional information

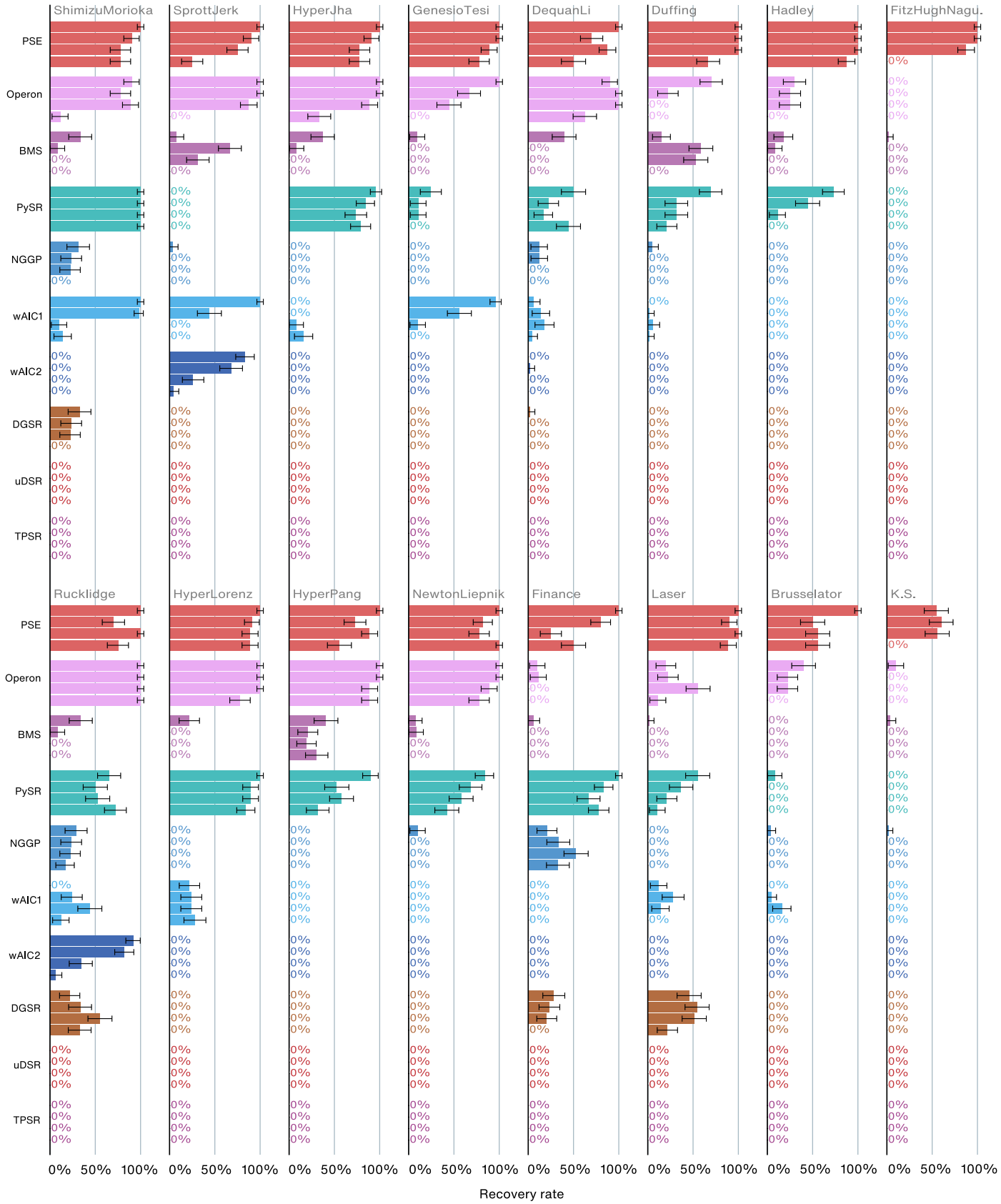
Extended data is available for this paper at https://doi.org/10.1038/s43588-025-00904-8.

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s43588-025-00904-8.

Correspondence and requests for materials should be addressed to Hao Sun.

Peer review information *Nature Computational Science* thanks Fabricio Olivetti de França and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. Peer reviewer reports are available. Primary Handling Editor: Fernando Chirigati, in collaboration with the *Nature Computational Science* team.

https://doi.org/10.1038/s43588-025-00904-8



Extended Data Fig. 1 | Data-driven discovery of nonlinear chaotic dynamics by different models. The average recovery rates of SR algorithms, including PSE, BMS, PySR, NGGP, DGSR, wAIC, uDSR, TPSR, and Operon, are compared on 16 nonlinear chaotic dynamics datasets. Given the same budget of computational time, PSE has a higher probability of finding the true governing equations among a nearly infinite number of possible expression structures. Each group of bars with the same color, from top to bottom, represents the recovery rates under four different levels of Gaussian noise (1%, 2%, 5%, and 10% of the data's standard deviation). Detailed results can be found in Supplementary Tables 16–19. The error bars represent the 95% confidence interval of the mean recovery rate for each system.