arXiv:2301.11919v1 [cs.LG] 27 Jan 2023

INCORPORATING BACKGROUND KNOWLEDGE IN SYMBOLIC REGRESSION USING A COMPUTER ALGEBRA SYSTEM

A PREPRINT

Charles Fox[2], Neil Tran[1], Nikki Nacion[1], Samiha Sharlin[1], and Tyler R. Josephson[1,2]

[1]Department of Chemical, Biochemical, and Environmental Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250
[2]Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250

ABSTRACT

Symbolic Regression (SR) can generate interpretable, concise expressions that fit a given dataset, allowing for more human understanding of the structure than black-box approaches. The addition of background knowledge (in the form of symbolic mathematical constraints) allows for the generation of expressions that are meaningful with respect to theory while also being consistent with data. We specifically examine the addition of constraints to traditional genetic algorithm (GA) based SR (PySR) as well as a Markov-chain Monte Carlo (MCMC) based Bayesian SR architecture (Bayesian Machine Scientist), and apply these to rediscovering adsorption equations from experimental, historical datasets. We find that, while hard constraints prevent GA and MCMC SR from searching, soft constraints can lead to improved performance both in terms of search effectiveness and model meaningfulness, with computational costs increasing by about an order-of-magnitude. If the constraints do not correlate well with the dataset or expected models, they can hinder the search of expressions. We find Bayesian SR is better these constraints (as the Bayesian prior) than by modifying the fitness function in the GA.

1 Introduction

1.1 Symbolic Regression for Scientific Discovery

Symbolic Regression (SR) is our tool of choice, as it generates mathematical expressions that are optimized for complexity and accuracy to a given dataset. Since John Koza pioneered the paradigm of programming by means of natural selection, many applications for SR in scientific discovery have emerged [1]. Unlike other applications of machine learning techniques, scientific research demands explanation and verification, both of which are made more feasible by the generation of human-interpretable mathematical models (as opposed to fitting a model with thousands of parameters) [2–4]. Furthermore, SR can be effective even with very small datasets ($\sim$10 items) such as those produced by difficult or expensive experiments which are not easily repeated. The mathematical expressions produced by SR can easily be extrapolated to untested or otherwise unreachable domains within a dataset (such as extreme pressures or temperatures).

For decades, SR has discovered interesting models from data in many unique applications including inferring process models at the Dow Chemical company [5], rainfall-runoff modelling [6] and rediscovering equations describing double-pendulum motion [7]. Symbolic regression has been applied across all scales of scientific investigation, including the atomistic (interatomic potentials [8]), macroscopic (computational fluid dynamics [9]), and cosmological (dark matter overdensity [10]) scales. Some techniques facilitate search through billions of candidate expressions, such as the space of nonlinear descriptors of material properties [11]. While most applications of SR in science focus on identifying empirical patterns in data, such "data-only" approaches do not account for potential insights from background theory. In fact, some SR works emphasize their capabilities of discovery “without any prior knowledge about physics, kinematics, or geometry” [7]. Nonetheless, we posit that prior knowledge need not be discarded, and in this work, we explore how

theory may be incorporated into symbolic regression to demonstrate machine learning in the context of background knowledge.

1.2 Incorporating Background Knowledge into Symbolic Regression

One particularly important step towards effective use of SR in specific domains is the addition of prior knowledge. This step has the potential to take a general purpose SR algorithm and use it to find novel models with physical meaning. For example, AI-DARWIN is uses prior knowledge of chemical reaction mechanisms in the form of predefined functions that a genetic algorithm may use in its search of equation space, ensuring that each generated model is mechanistically meaningful [12]. This approach specifically encodes the prior knowledge in the form of functions available instead of limitations on functions generated. In another recent example, Engle and Sahinidis use a deterministic symbolic regression algorithm that constrains the space of possible equations, not to those constructed from a library of meaningful function components, but to those functions that obey derivative constraints from theory. This improves the quality of generated expressions for thermodynamic equations of state [13]. Another approach to incorporating background knowledge in symbolic regression is the Bayesian Machine Scientist (BMS) [14]. BMS rigorously incorporates background knowledge in the form of a Bayesian prior on symbolic expressions; expressions are *a priori* more likely if their distribution of mathematical operators aligns with the distribution of operators in a corpus of prominent equations. However, their approach to the Bayesian prior does not incorporate meaning from particular scientific domains.

Checking consistency of equations *after* the search is complete is also possible. Previously, we showed that generated expressions can be compared to rich background knowledge (expressed as axioms for the environment under study), by posing generated expressions as conjectures to an automated theorem prover (ATP) [15]. However, state-of-the-art ATPs are too slow to incorporate this logical check as symbolic expressions are generated, and therefore cannot be easily used to bias the search for equations in light of that background knowledge. Moreover, translating scientific theories into a computer-interpretable form is not straightforward.

We address these specific drawbacks by combining symbolic regression systems (both genetic algorithm and Bayesian approaches) with a computer algebra system that checks constraints as an equation search is conducted. This is similar to Logic Guided Genetic Algorithms (LGGA), which uses "auxiliary truths" (ATs) corresponding to datasets in order to weigh items in a dataset as well as augment it with more information [16]. LGGA follows an iterative approach of training an arbitrary genetic algorithm with some dataset, augmenting that dataset with ATs, and training that algorithm again with more informative data. An important distinction between our work and LGGA is that the dataset is not altered in any way and the addition of extra information is performed during the execution of the GA.

1.3 Adsorption

Adsorption, the phenomenon in which molecules bind to a surface, enables chemical processes including carbon capture, humidity control, removal of harmful pollutants from water, and hydrogen production [17] [18] [19] [20]. Models of adsorption enable prediction and design of engineered adsorption processes, and many have been proposed over the years (selected equations are shown in Table 1) [21–24]. These models relate the amount adsorbed at equilibrium as a function of pressure or concentration and are commonly expressed as equations that are either empirical or derived from theory. For example, the Freundlich isotherm [25], is an empirical function designed to fit observed data, the Langmuir [26] and BET [27] isotherms are derived from physical models, and the Sips [28] isotherm is Langmuir-inspired with empirical terms added for fitting flexibility. We wonder, "What kinds of models could be generated by a machine learning system, and what role can background knowledge play in the search for accurate and meaningful expressions?"

Table 1: Some well-known isotherms written as SR might find them, and their complexities.

Isotherm	Literature Expression	Symbolic Regression Form	SR Complexity
Langmuir [26]	$\frac{q_{max}K_{eq}p}{1+K_{eq}p}$	$\frac{c_1 p}{c_2+p}$	7
Dual-Site Langmuir [26]	$\frac{q^a_{max}K^a_{eq}p}{1+K^a_{eq}p} + \frac{q^b_{max}K^b_{eq}p}{1+K^b_{eq}p}$	$\frac{c_1 p}{c_2+p} + \frac{c_3 p}{c_4+p}$	15
BET [27]	$\frac{c_1(p/p_0)}{(1-p/p_0)(1-p/p_0+c_2(p/p_0))}$	$\frac{c_1 p}{c_2+c_3 p+c_3 p^2}$	15
Freundlich [25]	$c_1 p^{\frac{1}{n}}$	$c_1 p^{c_2}$	5
Sips [28]	$\frac{c_1 p^{\frac{1}{n}}}{1+c_1 p^{\frac{1}{n}}}$	$\frac{p^{c_2}}{c_1+p^{c_2}}$	9

1.4 Thermodynamic Constraints

We consider models to be more *meaningful* when they satisfy thermodynamic constraints on the functional forms appropriate for modeling these phenomena. That is, a random equation that fits data, but does not approach zero loading correctly, is less trustworthy outside the training data than an equation constrained to follow thermodynamics. We have identified three constraints relevant for single-component adsorption [15]:

$$\lim_{p\to 0} f(p) = 0 \tag{1}$$

$$\lim_{p\to 0} f'(p) < \infty \tag{2}$$

$$\forall p > 0 \qquad f'(p) \geq 0 \tag{3}$$

Constraint 1 ensures that, in the limit of zero pressure, all molecules must desorb, and loading cannot be negative. Constraint 2 requires that in the limit of zero pressure, the slope of the isotherm must be a positive finite constant. Talu and Myers show that, as pressure approaches zero, the slope of the adsorption isotherm equals the adsorption second virial coefficient B_{1S}, which characterizes the interaction between one molecule and the surface, and must be a finite positive number [29] [30]:

$$\lim_{p\to 0} \frac{df}{dp} = \frac{B_{1S}}{RT} = c \tag{4}$$

Constraint 3 requires that loading does not decrease with increasing pressure (the isotherm is monotonically non-decreasing) for all ($\forall$) positive values of pressure. Note that this does not hold for mixture adsorption (in which competition plays a role), nor in BET adsorption, which exhibits a discontinuity at the saturation pressure, instead of a monotonic increase.

1.5 PySR: Symbolic Regression using Genetic Algorithms

PySR, Python for Symbolic Regression, is a Python library that uses a genetic algorithm for symbolic regression [31]. PySR is a Python wrapper that calls a Julia library by the same author, SymbolicRegression.jl (SR.jl), for numerical performance. Due to the nature of the modifications needed to the algorithm for this work, the base Julia library was used, but all added functionality should be inherited by the Python wrapper library as well.

The basic premise is that one or more populations of models move towards more optimal solutions via random mutations. At each generation, some members of a population are removed based on their fitness, age, or some other criteria (PySR replaces the oldest members). Beneficial solutions are encouraged by having more optimal members of a population mutate and reproduce.

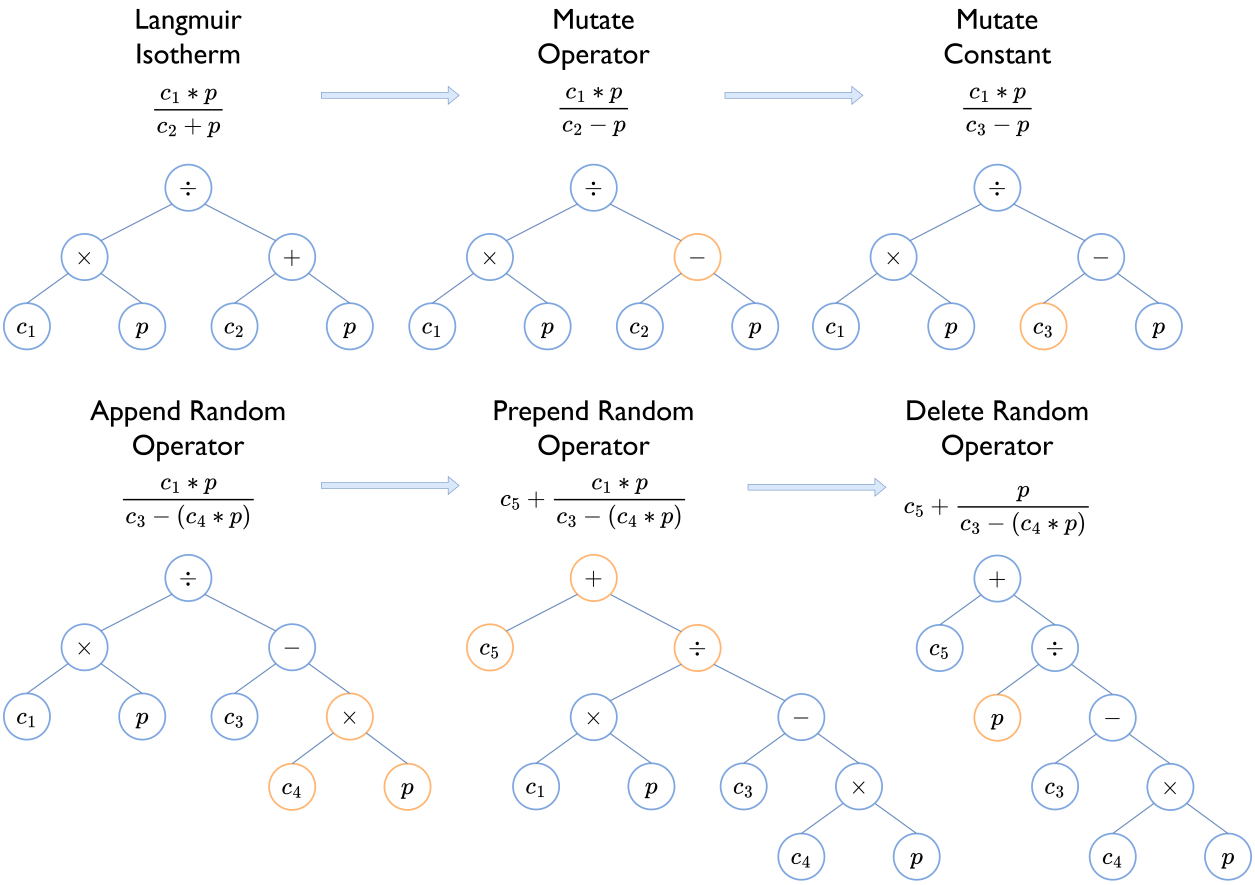

Figure 1: All mutations (except for random tree generation and simplification) in PySR in succession (read from left to right, top to bottom). Changes from each previous expression tree are shown in orange.

Changes include mutating a single constant or operator, simplifying the expression, or performing crossover between two expressions (Fig. 1 and Fig. 2). PySR uses multiple populations in a method similar to the island methodology [32]. This aims to allow for specialization by separately evolving unique populations, occasionally allowing some members to move between them to share that specialization. Specifically, PySR implements the so-called Hall of Fame (HOF), which is a Pareto front built from the best members across each population. After a number of generations, each population submits its top 10 best members (based on score) which are then compared and pared down via Pareto front. Expressions that remain in the HOF are used for future mutations in each of the populations.

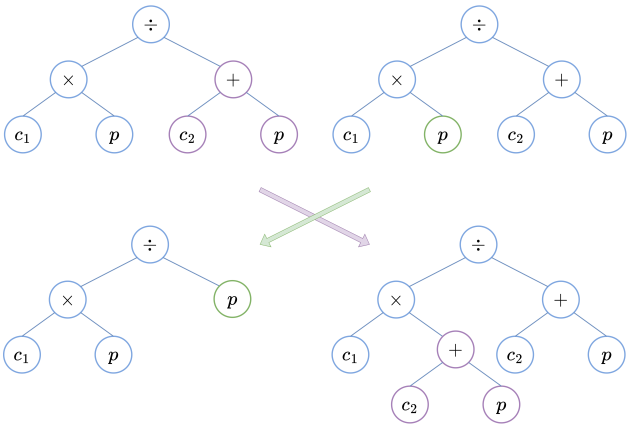

Figure 2: An example of the crossover mutation between two expression trees.

1.6 Bayesian Symbolic Regression

The Bayesian Machine Scientist (BMS) by Guimera et. al. [14] approaches symbolic regression from a Bayesian perspective. Bayesian Symbolic Regression (BSR) frames the search for accurate, concise and informed models as sampling the marginal posterior distribution of symbolic models with respect to a prior and fit to a dataset. Markov chain Monte Carlo (MC) is used to generate new expression trees (Fig. 3), which are accepted or rejected based on their likelihood. The authors define three MC moves: node replacement, root addition/removal, and elementary tree replacement, which together enable construction of expression trees while maintaining detailed balance, ensuring proper sampling of the posterior.

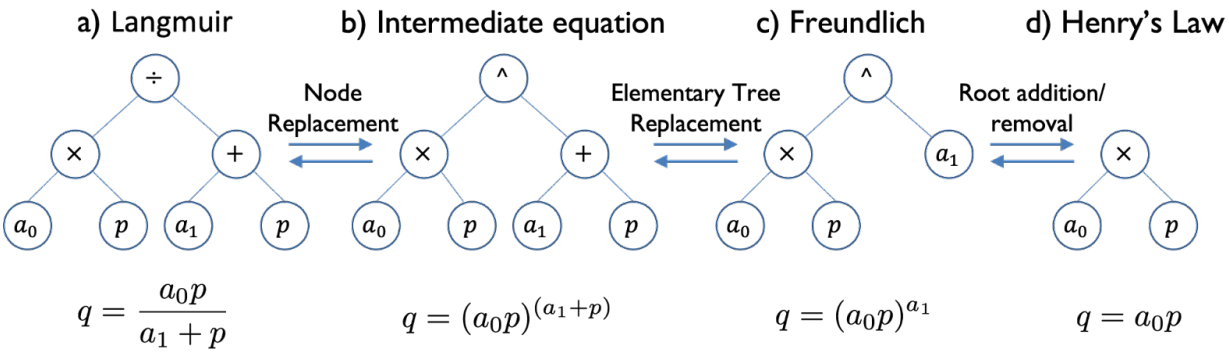

Figure 3: Illustrating the moves available to the BMS algorithm, as applied to adsorption equations. In contrast to the mutations available in PySR, these transformations satisfy detailed balance [14].

Specifically, the probability of some model given some data is defined as:

$$p(f_i|D) = \frac{1}{Z}\int_{\Theta_i} d\theta_i p(D|f_i,\theta_i)p(\theta_i|f_i)p(f_i) = \frac{\exp[-\mathcal{L}(f_i)]}{Z} \tag{5}$$

where Z is the probability of the dataset $p(D)$, Θ_i is the space of possible values for parameters θ_i and $\mathcal{L}$ is the description length of the model.

A central idea in BSR is the inclusion of a prior to emphasize expressions that are *a priori* more likely than others, regardless of the data. Guimera, et al. based their prior off of a corpus of 4080 mathematical expressions collected from Wikipedia (from the "list of scientific equations named after people"), and assigned the prior likelihood using the counts of each unique operator (n_o) in the corpus, by fitting parameters α and β like so:

$$\text{EP} = -\log(p(f_i)) = \sum_{o\in O}\left[\alpha_o n_o(f_i) + \beta_o n_o^2(f_i)\right] \tag{6}$$

While this method leads to a distribution of expressions that resembles the corpus prior when run with no data, $p(f_i)$ can also be set to a constant value so that there is no bias based on operators present in the search process. For our problem, we crafted a prior especially for adsorption thermodynamics (see details in Methods).

2 Methods

2.1 Checking Thermodynamic Constraints

Three constraint checking functions for the thermodynamic constraints described in Section 1.4) were developed using the Python library SymPy, an open-source computer algebra system [33]. Each function returns either true or false, depending on if its constraint is met or not (if a time limit is exceeded, the constraint is returned as false). For both PySR and BSR, we found that hard constraints (rejecting every expression that fails any constraint) severely hinder the search process, cutting off intermediate expressions between better expressions that may also pass the constraints. Consequently, we impose these as "soft" constraints, penalizing expressions for constraint violation, without outright rejecting them. This approach (as implemented in PySR) is detailed in algorithm 1.

Constraints 1 and 2 could be checked using SymPy's limit and derivative functionality, but Constraint 3 was more challenging. Though SymPy can check if an expression is strictly increasing in a given range, the check for monotonicity returns false if any change in curvature (critical point) exists for the expression – thus preventing functions such as x^3 from being considered monotonically non-decreasing. To allow for zero slope, we implemented a custom monotonic non-decreasing check function (see alg. 2). Instead of just checking the slope in one range, it checks the ranges between all critical points (as well as to the start and end of the original range in question).

We hypothesize that the "equation space" explored by SR includes accurate, but not thermodynamically consistent expressions that can be rejected through the incorporation of background knowledge, guiding the search to more theory-informed expressions.

2.2 PySR Modifications

In PySR, each member in a population has a score to be minimized, which combines the loss and complexity (defined by total nodes in the expression tree). When a thermodynamic constraint is violated, we multiply the loss function by a penalty, raising the score and making the expression less fit. This allows any number of constraints to be checked in any order (as multiplication is commutative), and confers larger penalties to expressions that violate multiple constraints.

$$\text{Loss: } L = \ell_2^R * \prod_{i=1,2,3} c_i^{\delta_i} \text{ where } \delta_i = \left\} \begin{matrix} 1 & \text{if constraint } i \text{ passed} \\ 0 & \text{if constraint } i \text{ failed} \end{matrix} \right(\tag{7}$$

$$\text{Member Score: } S = L + n_{nodes} * c_l \tag{8}$$

The above equations detail how the loss and score are calculated in PySR. ℓ_2^R is the L2 norm, c_i is the penalty for constraint i, δ_i indicates if constraint i is passed and c_l is the penalty for the length / complexity of an expression.

PySR also has the option to take any operators defined in Julia or Python, including custom user-defined operators. For this work only the operators $+$, $-$, $*$ and $\div$ were used to manage the size of the search space. Expressions written in their canonical form may use other operators such as exponents but these are only due to simplification of generated expressions.

2.3 BMS Modifications

The Bayesian prior used in the Bayesian Machine Scientist code Bayesian Machine Scientist code by Guimera et al. [14] incorporates "background knowledge" in its equation search, through the use of a Bayesian prior based on mathematical operation frequency among named equations in Wikipedia. Because the majority of these equations are unrelated to adsorption, they may lead the search in a less optimal direction. Instead, we consider the thermodynamic constraints described above to be our "prior knowledge," and construct the following expression:

$$\text{EP} = \sum_{o \in O} \left[c_{ops} n_o(f_i) \right] + \sum_{i=1,2} c_i * \delta_i \text{ where } \delta_i = \left\} \begin{matrix} 1 & \text{if constraint } i \text{ passed} \\ 0 & \text{if constraint } i \text{ failed} \end{matrix} \right(\tag{9}$$

where c_{ops} is the constraint penalty for operators (analogous to the parsimony parameter in PySR), and n_o is the count of each operator in expression f_i. This expression directly replaces Eq. 6, changing the prior distribution. Note that we checked all three constraints with PySR, and only the first two constraints with BSR (omitting the monotonic non-decreasing check).

3 Results

3.1 Datasets

To examine the effects of adding constraints to SR during a search, four experimental adsorption datasets were identified: adsorption of nitrogen and methane on mica [26], adsorption of isobutane in silicalite, [34] and adsorption of nitrogen on Fe-Al_20_3 catalyst [27]. The first and second datasets come from the landmark paper introducing the Langmuir isotherm model [26]. This model assumes there are discrete loading "sites" that do not interact with each other, and that each site can either be occupied or not. The isobutane dataset is well-described by a dual-site Langmuir model which has two unique types of sites. The fourth dataset (referred to as the BET dataset) was used by the authors of BET theory

to support their model for multilayer adsorption. These data and the respective ground truth model fits are shown in Fig. 4.

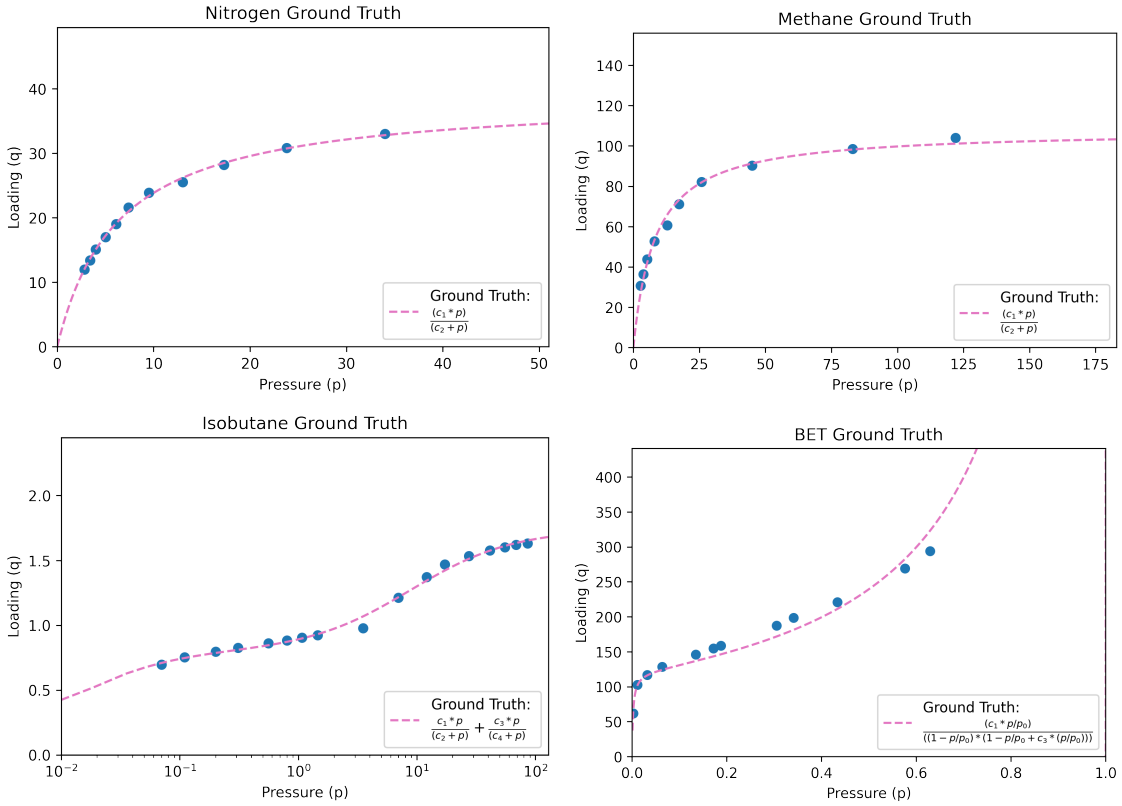

Figure 4: Each dataset and corresponding ground truth model with constants fit using SciPy. Isobutane is shown with log scaled pressure so that the two separate curves are visible. BET is shown with pressure increasing to 1 so the asymptote is visible.

3.2 Langmuir Datasets

The main results of this work are shown in two plot types. The left column contains Pareto fronts which show the best expressions based on complexity and accuracy. In these, the horizontal axis shows increasing complexity (defined as the total number of nodes in an expression tree), and the vertical axis shows loss, which is logarithmically scaled so the trend of the Pareto front is more apparent. The best expression at each complexity is taken from each of 8 runs (gray curves), with the overall Pareto front shown in orange. The “ground truth" expression for each dataset is also shown in the form it would likely be expressed by SR, along with loss found using fit constants. The right column of each figure shows the dataset and select expressions from the overall Pareto front for that test. Only some expressions are shown so plots remain readable and because expressions longer than the ground truth are usually overfit and overlay the ground truth expression too closely for distinction. The ground truth is plotted with a dotted line so that expressions with similar accuracy can still be seen. Plotting the generated expressions on the data helps to illustrate how they may or may not follow the thermodynamic constraints and how similar they are to the ground truth.

Figure 5 shows the results from both SR algorithms with constraints on and off on the Langmuir nitrogen dataset. The first and second rows show BSR and PySR respectively with constraints off and clearly show that BSR finds the ground truth while PySR does not. The expression that defines the corner at complexity 7 in the BSR Pareto front plot (Fig. 5a) is indistinguishable from the ground truth (both written mathematically and drawn on the data) when viewed in the isotherm plot (Fig. 5b). The BSR plot (Fig. 5a) has a much larger variance in terms of best Pareto fronts across 8 runs (as shown by the grey lines) than PySR, but this may indicate longer time needed for the algorithm to converge. The corresponding isotherm plots (the right column) show how expressions fit the data better as they become more complex, following the general trend of the Pareto fronts. These plots also show how some expressions can fit the data reasonably well while violating the constraints from theory, as is the case in the plot for PySR (Fig. 5d). In fact, only 2.2% of

expressions generated by PySR (without enforcing constraints) pass the first constraint and only 33% pass the second constraint (Table 2). Without constraints enforced, BSR finds more consistent expressions than PySR, with 37% of its expressions passing the first constraint and 67% passing the second.

When the thermodynamic constraints are enabled, the effect is clearly shown in the Pareto fronts (bottom two rows). Both SR methods find the ground truth and achieve the same or similar accuracy (accuracy is less for the same expression when the constants were not optimized as thoroughly in the search). Datasets that are well represented by the Langmuir isotherm show the effects of the constraints well because it is typically very accurate as well as being concise. The isotherm plots show, as before, how the expressions fit the data better as they become more complex but showing anything beyond a complexity of 7 is redundant as the ground truth is discovered and matches the pre-fit ground truth almost perfectly. The trend of slightly more variation across BSR runs also continues here to some extent and the variation across PySR runs appears roughly similar to with constraints disabled. Importantly, PySR sees a 5x increase in expressions passing the first constraint (though still only 10%) and a marginal improvement across the other two constraints (8% and 13%). The change is more stark in BSR where twice as many expressions now pass the first constraint (up to 72%) and a significant portion pass the third constraint (up to 19% from 0.46%) even though it was not included in the Bayesian prior.

While the results are mostly similar for the methane dataset, there are some important differences. Like with the nitrogen dataset, BSR finds the ground truth without constraints enabled while PySR does not. This is apparent in the Pareto fronts (Fig. 6a and Fig. 6c). In this case, PySR finds an expression with complexity 9 with more accuracy than the ground truth, though with an extra constant in the numerator, it violates the thermodynamic constraints (Fig. 6d). Imposing the constraints penalized the loss for this expression relative to the ground truth, but not enough to overcome the increased accuracy (Fig. 6h). As with nitrogen, BSR does a better job of finding expressions that pass the constraints, even when they are not enabled, as it finds 33% passing the first and 51% passing the second (where PySR finds 4.1% and 48% respectively).

3.3 Isobutane Dataset

Unlike the methane and nitrogen datasets (Fig. 5 and 6) which are best modeled by the Langmuir isotherm, the isobutane dataset (Fig. 7) is best modeled by the dual-site Langmuir isotherm, which has twice the complexity. Despite this significant complexity, the dual-site Langmuir isotherm is not significantly more accurate than many expressions shorter than it. This is best seen in Fig. 7c and 7g which show the Pareto fronts for PySR with constraints off and on respectively. In both plots, expressions with half the complexity reach almost the same accuracy, creating a plateau from complexity 7 onward. This is also shown well in the corresponding isotherm plots which show that the expressions found at complexity 7 match the data as well as the ground truth. Importantly, these expressions do not satisfy the thermodynamic constraints.

Unlike PySR, BSR does not find expressions with accuracy close to the ground truth until the same complexity.

For BSR, including constraints shifts the whole Pareto front down (Fig. 7a to Fig. 7e), indicating that more accurate expressions were found at many complexity levels. While PySR did not find accurate expressions consistent with the constraints, BSR did. In this case, BSR finds the ground truth expression while PySR does not. This is not apparent on either the Pareto fronts or isotherm plots however, because the accuracy of the expression found is about 10x worse than the fit ground truth and the best expressions found at that complexity. This is likely because, while the ground truth is found, the form it was originally produced in (before being simplified) is much more complex.

In PySR, penalizing expressions that violate constraints actually led to populations of equations that violated constraints two and three more often, with a decrease of about 10% in each case (see Table 2). This was surprising – we anticipated that imposing penalties would lead to fewer violating expressions, but the opposite occurred. For BSR as well, including constraints in the prior actually led to a decrease in expressions satisfying the second constraint (from 46% to 36%), and a slight increase in the first and third constraints.

3.4 BET Dataset

The BET dataset is unique because the ground truth expression diverges to infinity as the pressure approaches 1 (pressure in this case is relative vapor pressure, p/p^{sat}; the vapor being adsorbed becomes a liquid as $p/p^{\mathrm{sat}} \to 1$. So in this case, the third constraint (that it is monotonically non-decreasing) no longer holds for all pressure (seen in Fig. 8). Nonetheless, we found that whether or not constraints were enabled, many of the most accurate expressions generated by PySR for this dataset pass the third constraint (78.65% without constraints and 81.29% with), contrary to the ground truth theory. Furthermore, PySR satisfies the first two constraints less frequently with constraints on compared to with constraints off. One possible explanation for this behavior is that the dataset itself is more easily fit by expressions with

expressions that are monotonically non-decreasing, at least from the perspective of the PySR algorithm. Overall, while PySR can find accurate expressions for the BET dataset, it fails to find expressions that also follow the constraints, even when they are enabled.

In contrast, BSR did not generate many expressions that were monotonically nondecreasing, and the incorporation of constraints had a substantial effect on the search. Specifically, the second constraint is passed about 92% of the time both with it enabled and disabled and the portion passing the first constraint increases dramatically from 16% to 85% once it is enabled. This leads to a large number of models which agree with the requisite constraints for BET, but none of these are the ground truth rediscovered. Instead, many expressions with close to (or better than) the accuracy of the ground truth are found by both algorithms in both cases, none of the isotherms plotted appear similar. The asymptote at a partial pressure of 1 is not replicated by any similarly accurate expressions and the slight curve of the ground truth in the middle of the dataset is also absent. These results together seem to indicate that the constraints, while thermodynamically correct, do not provide enough information (or even provide contradictory information) for rediscovering the BET ground truth expression.

Dataset	Constraints Active	BSR C1	BSR C2	BSR C3	PySR C1	PySR C2	PySR C3
Nitrogen	False	37%	67%	0.46%	2.2%	33%	46%
Nitrogen	True	72%	73%	19%	10%	41%	59%
Methane	False	33%	51%	0.51%	4.1%	48%	61%
Methane	True	59%	59%	2.5%	5.7%	54%	62%
Isobutane	False	24%	46%	0.45%	3.4%	65%	68%
Isobutane	True	36%	36%	1.3%	5.5%	56%	58%
BET	False	16%	92%	0.09%	6.2%	35%	79%
BET	True	85%	92%	2.4%	4.7%	30%	81%

Table 2: Percentage of expressions generated passing each of the three constraints. Results are shown across both SR methods, all datasets and with constraints active and disabled.

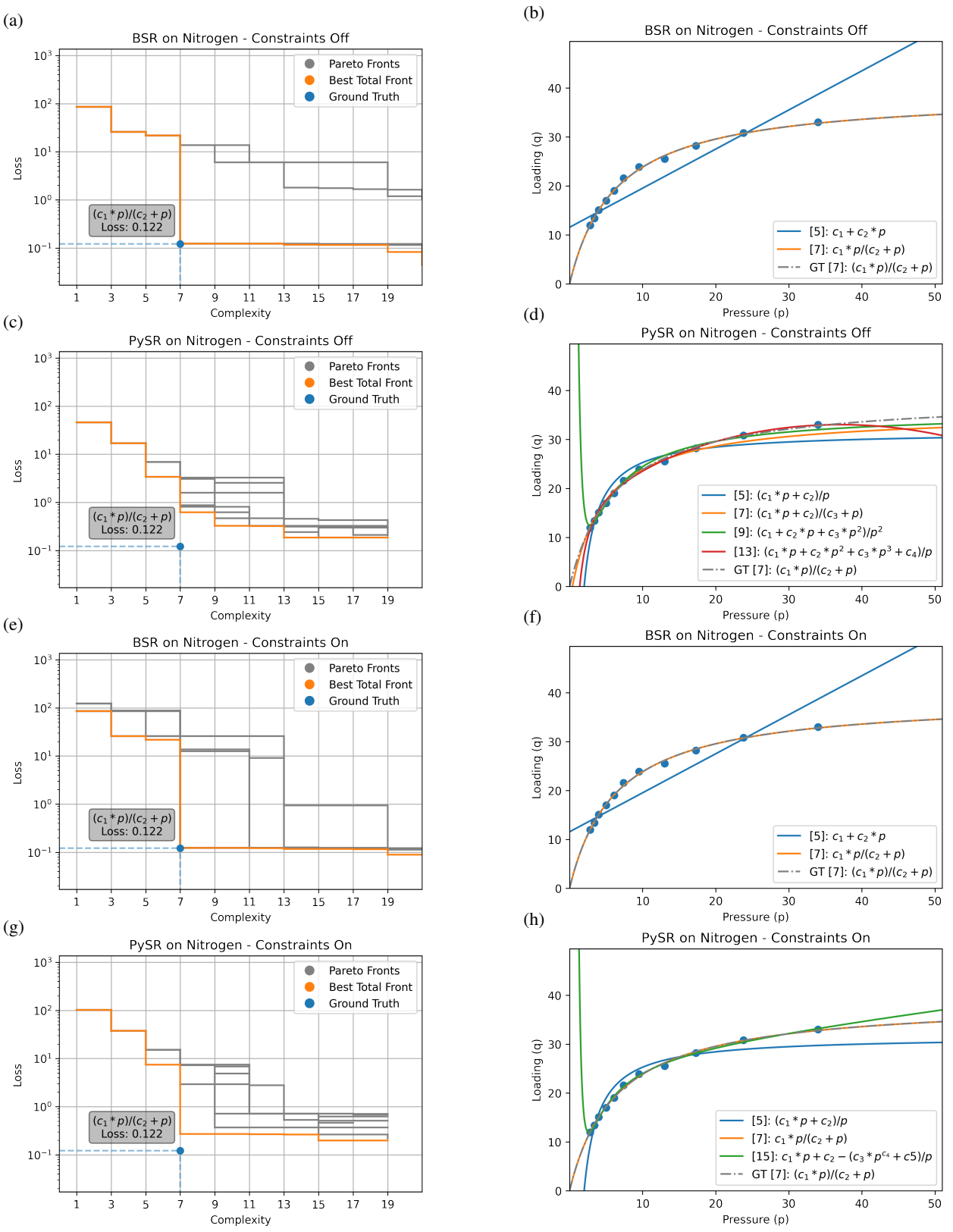

Figure 5: BSR and PySR on the Nitrogen dataset. The left column shows combined Pareto fronts across 8 runs and the right column shows interesting isotherms found at the defining corners of those Pareto fronts. The constraints are disabled in the top four subplots and enabled in the bottom four. The rows alternate between BSR and PySR.

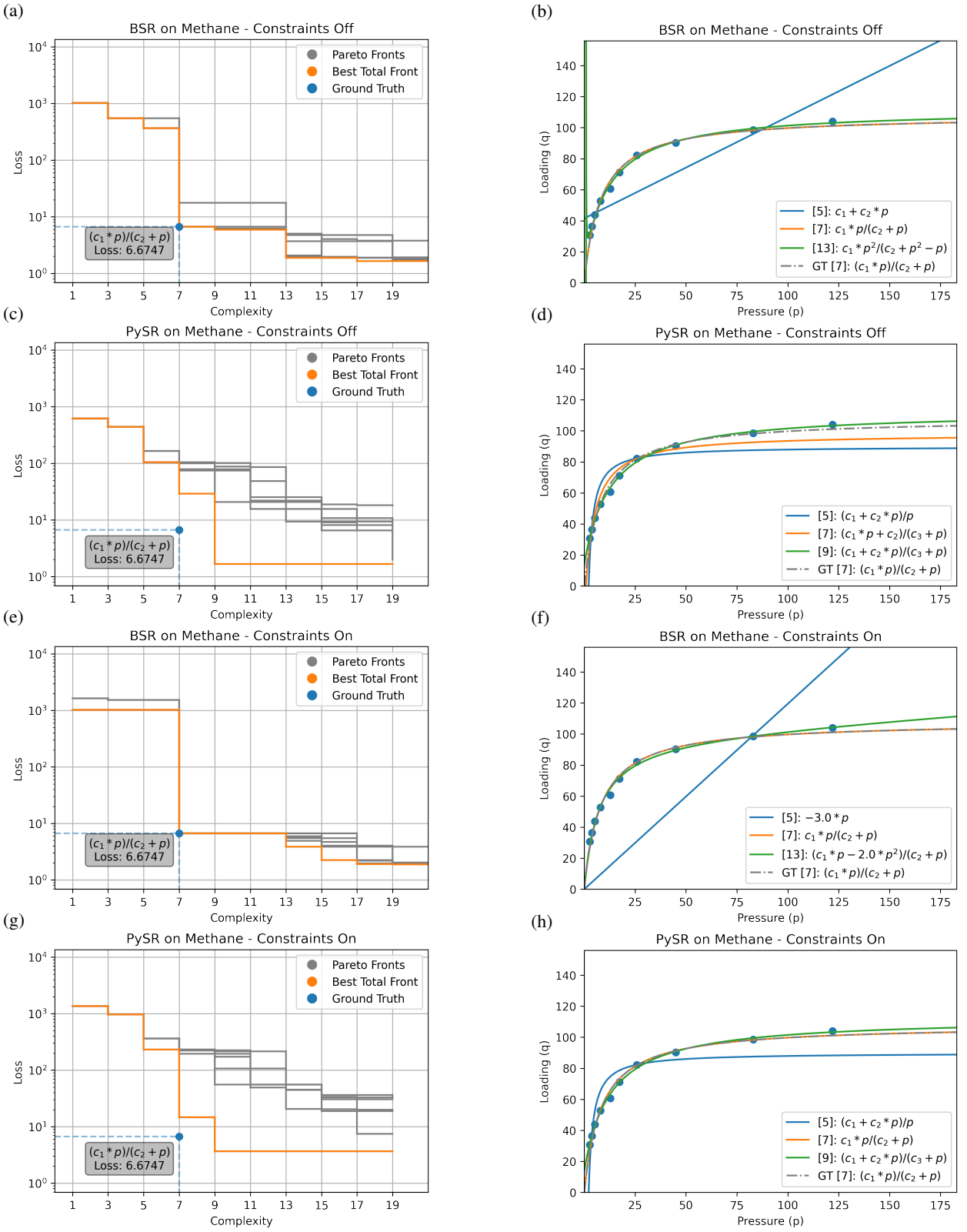

Figure 6: BSR and PySR on the Methane dataset. The left column shows combined Pareto fronts across 8 runs and the right column shows interesting isotherms found at the defining corners of those Pareto fronts. The constraints are disabled in the top four subplots and enabled in the bottom four. The rows alternate between BSR and PySR.

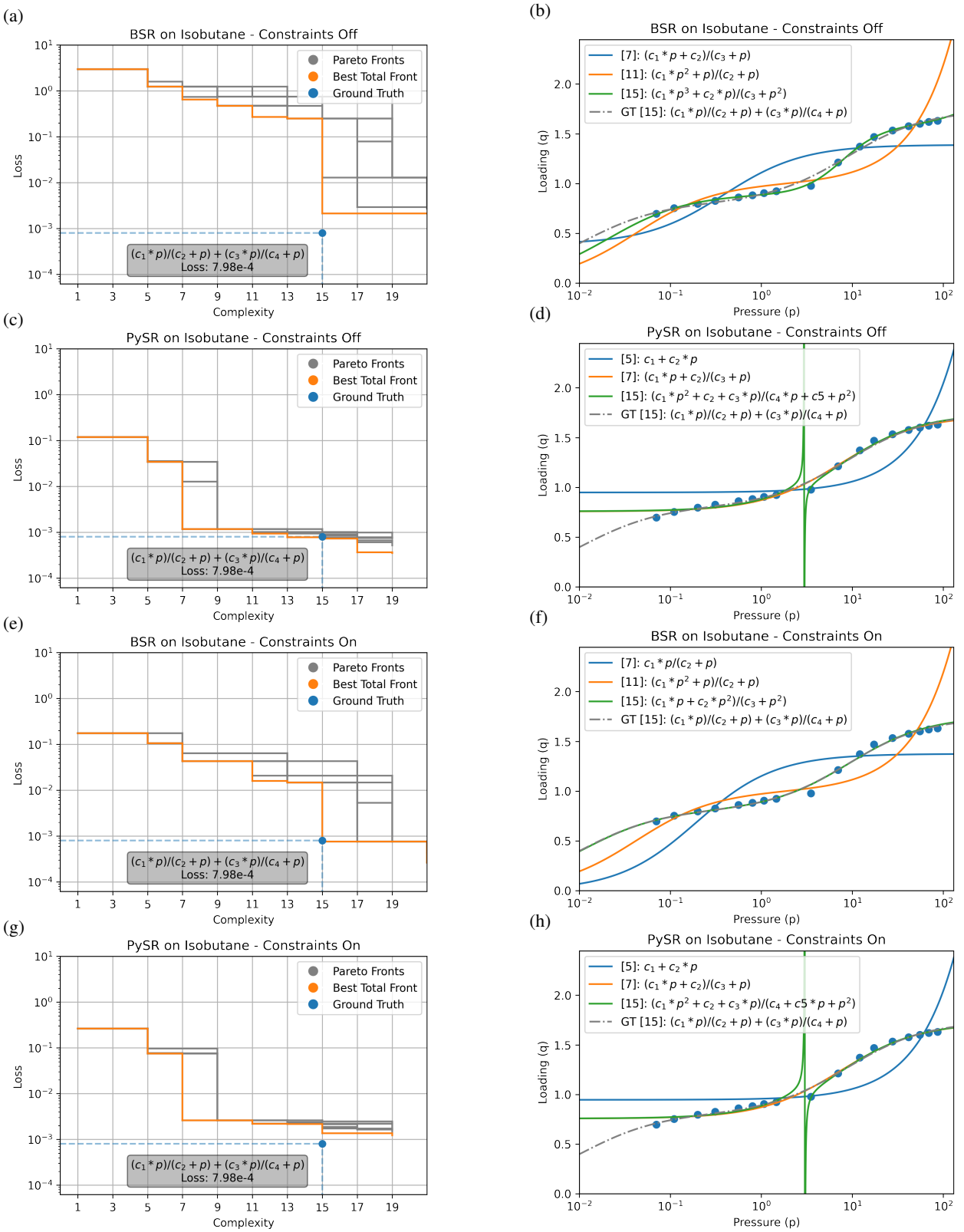

Figure 7: BSR and PySR on the Isobutane dataset. The left column shows combined Pareto fronts across 8 runs and the right column shows interesting isotherms found at the defining corners of those Pareto fronts. The constraints are disabled in the top four subplots and enabled in the bottom four. The rows alternate between BSR and PySR.

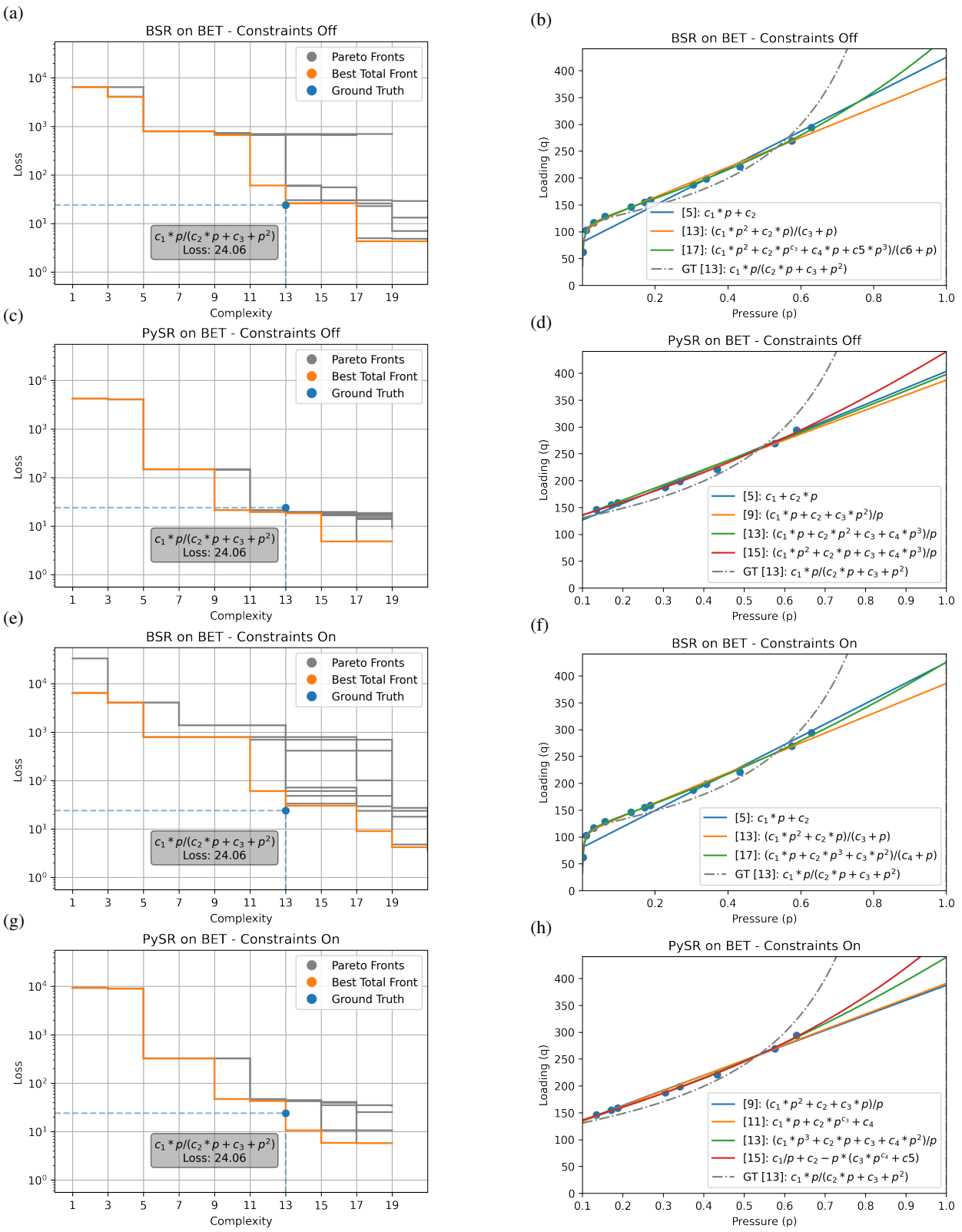

Figure 8: BSR and PySR on the BET dataset. The left column shows combined Pareto fronts across 8 runs and the right column shows interesting isotherms found at the defining corners of those Pareto fronts. The constraints are disabled in the top four subplots and enabled in the bottom four. The rows alternate between BSR and PySR.

4 Discussion

4.1 Effectiveness

This work highlights that sometimes, a stochastic search through equation space finds equations that are superior to ground truth expressions – in our case achieving comparable accuracy to the ground truth expressions while also being less complex (shorter expression length). This is particularly observed in the case of isobutane (Fig. 7); both with constraints on and off, PySR finds the expression $\frac{c_1 p + c_2}{c_3 + p}$, which fits the data well but diverges from the ground truth as it approaches 0. But many of these expressions, while consistent with the data, are inconsistent with thermodynamics, as they violate our tested constraints. We have demonstrated that accounting for these constraints in the search process can guide the population or distribution of expressions toward thermodynamically consistent expressions, and aid in the identification of the ground truth expression. Sometimes this leads to more consistent equations, and sometimes it doesn't improve the search at all.

4.2 Exceptions To Constraints

While the three constraints presented in this work do follow from a broader thermodynamic theory, not all isotherm models in this work satisfy all the constraints. Specifically, the BET isotherm does not satisfy the third constraint because it reaches an asymptote as p/p_0 approaches 1. While this does break the monotonicity constraint, it also seems reasonable when considering what p_0 represents (the pressure at which the adsorbate becomes a liquid and the interaction fundamentally changes). This raises the question: what constraints to include and why? The decision to test if expressions pass the third constraint necessarily excludes the BET model because, while it is theoretically grounded, it only applies from in the range $(0, p_0)$. Furthermore, the data used does not extend past that range so expressions that do pass the third constraint may not appear much different in terms of what is relevant. We recommend carefully examining what constraints one may want to test and in what places they should actually be checked. Introducing incorrect constraints may hinder the search with our biases, and prevent algorithms from discovering phenomena outside our assumptions.

4.3 Computational Complexity / Runtime

As seen in Fig. 10), consideration of constraints increases runtime by an order of magnitude; this is even after we carefully integrated the computer algebra system into the SR algorithms to reduce overhead, and leveraged memory to avoid redundant checks on expressions previously visited. On average, numerically checking models is much faster than manipulating them symbolically (especially for larger expressions) – checking *every* new expression is quite expensive. This is unfortunately necessary if the constraints are to be considered as an integral part of the search. If a cheaper solution is needed, the search can be performed without constraints, and constraints checked after the fact. In fact, this approach enables even more elaborate methods of considering background knowledge, such as comparing against complex, multi-premise background theories using an automated theorem prover [15].

4.4 Challenges around complexity, simplification, and canonical form

In this work, simplification is necessary in order to identify whether a generated expression matches the ground truth and to assign generated expressions an appropriate complexity. We augmented SymPy's "simplify" function, to shorten the numerous rational expressions we generated into a "canonical form" (details in the Supporting Information). While some methods such as BSR attempt simplification during runtime, PySR does not because of the added computation needed per expression. Generating a "canonical form" for expressions generated by PySR sometimes increases, and sometimes decreases, the complexity. Some expressions are generated as complex expression trees that are much more complex than their canonical forms (Fig. 9).

Simplification is a crucial challenge of this work because complexity plays a significant role in SR. After all, models are compared via accuracy and complexity to make decisions during the search. A single model may very well take on different scores / likelihoods because of how it is written, influencing not just its standing, but subsequent steps in the search. Ideally, every model would always be written in the simplest form, but this is computationally intractable in some circumstances [35]. Because of this, comparing functions based on behavior (symbolic constraints) may be more appropriate, because limiting behavior is invariant to the numerous ways an expression can be written.

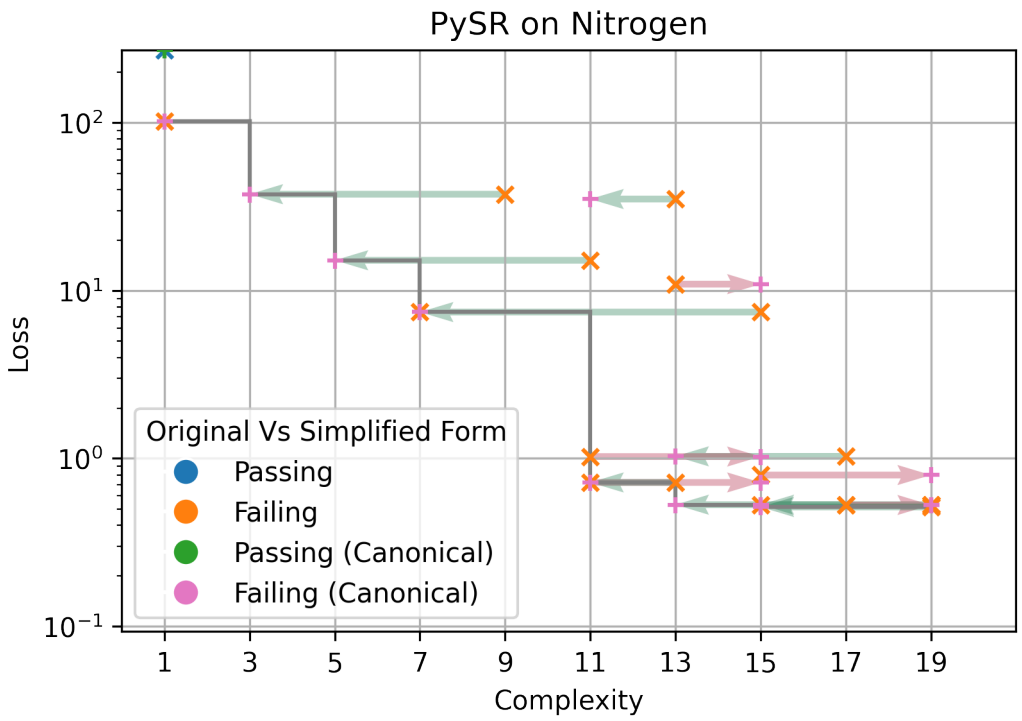

Figure 9: The effect of the canonical form checking function on a Pareto front showing the results from PySR on the Nitrogen dataset. Points on this plot are marked as “Passing" only if they pass all constraints checked.

4.5 Reducing Underspecification Through Inductive Biases

Machine learning researchers at Google recently highlighted the role of underspecification in machine learning pipelines [36]. They suggested that one way to combat underspecification is to use credible inductive biases to allow for selection from otherwise similarly effective models, and that these constraints should have little negative effect on accuracy if selected correctly. In this work, we find expressions that are roughly equivalent in terms of accuracy and complexity but have different functional forms, leading to different behavior outside the range of the data – signatures similar to those discussed in [36]. We find that adding thermodynamic constraints can help improve the search for good expressions, but this doesn't necessarily restrict the hypothesis space in the same way that inductive biases do; we were unable to effectively search with hard constraints, and so our hypothesis space still included expressions that are inconsistent with constraints. Instead, we can reduce the hypothesis space after the search is complete; by rejecting accurate-but-inconsistent expressions using our background knowledge, we improve on the issues of underspecification. Nonetheless, for datasets with reasonably complex behavior, there still exist multiple distinct thermodynamically-consistent expressions of similar accuracy and complexity. The space of equations defined by the limited number of operators considered here, even for one dimensional datasets, is just that vast!

5 Conclusions

In this work, we couple a computer algebra system to two symbolic regression algorithms in order to check the consistency of generated expressions with background knowledge. We find that including appropriate mathematical constraints can improve search effectiveness or break the search entirely, depending on the dataset and implementation details. Although computational costs increase by an order of magnitude, tightly integrating SR with a computer algebra system is a practical way to check for constraints on each expression generated during the search.

We have shown that consideration of constraints helps in rediscovering ground-truth isotherm models from experimental data, including the Langmuir and the dual-site Langmuir isotherms (though the dual-site Langmuir isotherm was not identified on the Pareto front, it was present in the generated models). In contrast, the BET isotherm was not rediscovered; more accurate and concise models were generated instead, and the most meaningful model (BET) was consequently missed. We found that Bayesian Symbolic Regression is a more effective and intuitive platform for

incorporating symbolic constraints in a Bayesian prior, rather than by modifying the fitness function in traditional genetic algorithms; the resulting populations of expressions were more attuned to the constraints with BSR. Finally, though background knowledge can screen out accurate yet inconsistent solutions, symbolic regression pipelines remain underspecified in our context, capable of generating multiple distinct solutions with similar performance and adherence to constraints.

6 Acknowledgements

We thank Marta Sales-Pardo and Roger Guimerà for discussions about the Bayesian Machine Scientist, and Miles Cranmer for assistance with PySR. This material is based upon work supported by the National Science Foundation under Grant No. (NSF #218938), as well as startup funds from the University of Maryland, Baltimore County.

7 Supporting Information

The modified version of PySR and the code used to run it are both available on GitHub. The PySR code was forked from the original repository on June 6th, 2020 and is available at https://github.com/CharFox1/SymbolicRegression.jl. The code for running PySR, parsing its output, and plotting the results, and is available at https://github.com/ATOMSLab/pySR_adsorption. The modified version of BMS code used in this paper is available at https://github.com/ATOMSLab/BayesianSymbolicRegression.

The Supporting Information includes 1) further description of the adsorption models considered here, 2) further discussion of the changes we implemented in PySR and BMS codes to implement thermodynamic constraint checking, including pseudocode for new algorithms, 3) description of our pipeline for collecting and analyzing generated expressions, 4) further discussion of the nuances around identifying of "interesting" expressions in automated pipelines and algorithms for simplification and pattern-matching, 5) details of constant fitting, 6) experiments comparing runtime of algorithms on different datasets, and 7) details of the testing environment on the UMBC supercomputer.

References

[1] John R. Koza. Genetic Programming, 1992.

[2] Felipe Oviedo, Juan Lavista Ferres, Tonio Buonassisi, and Keith T Butler. Interpretable and explainable machine learning for materials science and chemistry. *Accounts of Materials Research*, 3(6):597–607, 2022.

[3] Xiaoting Zhong, Brian Gallagher, Shusen Liu, Bhavya Kailkhura, Anna Hiszpanski, and T Han. Explainable machine learning in materials science. *npj Computational Materials*, 8(1):1–19, 2022.

[4] Jacques A Esterhuizen, Bryan R Goldsmith, and Suljo Linic. Interpretable machine learning for knowledge generation in heterogeneous catalysis. *Nature Catalysis*, 5(3):175–184, 2022.

[5] Arthur Kordon, Flor Castillo, Guido Smits, and Mark Kotanchek. Application Issues of Genetic Programming in Industry. pages 241–258. June 2006.

[6] Dragan A. Savic, Godfrey A. Walters, and James W. Davidson. A Genetic Programming Approach to Rainfall-Runoff Modelling. *Water Resources Management*, 13(3):219–231, June 1999.

[7] Michael Schmidt and Hod Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85, April 2009. Publisher: American Association for the Advancement of Science.

[8] Alberto Hernandez, Adarsh Balasubramanian, Fenglin Yuan, Simon Mason, and Tim Mueller. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression, August 2019. arXiv:1904.01095 [cond-mat, physics:physics].

[9] Mehrad Ansari, Heta A. Gandhi, David G. Foster, and Andrew D. White. Iterative symbolic regression for learning transport equations. *AIChE Journal*, 68(6):e17695, 2022. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.17695.

[10] 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*, volume 33, pages 17429–17442. Curran Associates, Inc., 2020.

[11] 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, August 2018. Publisher: American Physical Society.

[12] Arijit Chakraborty, Abhishek Sivaram, and Venkat Venkatasubramanian. AI-DARWIN: A first principles-based model discovery engine using machine learning. *Computers & Chemical Engineering*, 154:107470, November 2021.

[13] Marissa R. Engle and Nikolaos V. Sahinidis. Deterministic symbolic regression with derivative information: General methodology and application to equations of state. *AIChE Journal*, 68(6):e17457, 2022. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.17457.

[14] Roger Guimerà, Ignasi Reichardt, Antoni Aguilar-Mogas, Francesco A. Massucci, Manuel Miranda, Jordi Pallarès, and Marta Sales-Pardo. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Science Advances*, January 2020. Publisher: American Association for the Advancement of Science.

[15] Cristina Cornelio, Sanjeeb Dash, Vernon Austel, Tyler Josephson, Joao Goncalves, Kenneth Clarkson, Nimrod Megiddo, Bachir El Khadir, and Lior Horesh. AI Descartes: Combining Data and Theory for Derivable Scientific Discovery, October 2021. arXiv:2109.01634 [cs].

[16] Dhananjay Ashok, Joseph Scott, Sebastian J. Wetzel, Maysum Panju, and Vijay Ganesh. Logic Guided Genetic Algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):15753–15754, May 2021. Number: 18.

[17] R. Ben-Mansour, M. A. Habib, O. E. Bamidele, M. Basha, N. A. A. Qasem, A. Peedikakkal, T. Laoui, and M. Ali. Carbon capture by physical adsorption: Materials, experimental investigations and numerical modeling and simulations – A review. *Applied Energy*, 161:225–255, January 2016.

[18] James A. Ritter and Armin D. Ebner. State-of-the-Art Adsorption and Membrane Separation Processes for Hydrogen Production in the Chemical and Petrochemical Industries. *Separation Science and Technology*, 42(6):1123–1193, April 2007. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01496390701242194.

[19] M. H. Stenzel. Remove organics by activated carbon adsorption. *Chemical Engineering Progress; (United States)*, 89:4, April 1993.

[20] Douglas Ruthven. Principles of Adsorption and Adsorption Processes | Wiley, June 1984.

[21] G Limousin, J-P Gaudet, L Charlet, Stephanie Szenknect, V Barthes, and M Krimissa. Sorption isotherms: A review on physical bases, modeling and measurement. *Applied geochemistry*, 22(2):249–275, 2007.

[22] Keng Yuen Foo and Bassim H Hameed. Insights into the modeling of adsorption isotherm systems. *Chemical engineering journal*, 156(1):2–10, 2010.

[23] Nimibofa Ayawei, Augustus Newton Ebelegi, and Donbebe Wankasi. Modelling and interpretation of adsorption isotherms. *Journal of chemistry*, 2017, 2017.

[24] Jianlong Wang and Xuan Guo. Adsorption isotherm models: Classification, physical meaning, application and solving method. *Chemosphere*, 258:127279, November 2020.

[25] Herbert Freundlich. *Über die Adsorption in Lösungen. Habilitationsschrift durch welche... zu haltenden Probevorlesung" Kapillarchemie und Physiologie" einladet Dr. Herbert Freundlich*. W. Engelmann, 1906.

[26] Irving Langmuir. The Adsorption of Gases on Plane Surfaces of Glass, Mica and Platinum. *Journal of the American Chemical Society*, 40(9):1361–1403, September 1918. Publisher: American Chemical Society.

[27] Stephen Brunauer, P. H. Emmett, and Edward Teller. Adsorption of Gases in Multimolecular Layers. *Journal of the American Chemical Society*, 60(2):309–319, February 1938. Publisher: American Chemical Society.

[28] Robert Sips. On the structure of a catalyst surface. *The journal of chemical physics*, 16(5):490–495, 1948.

[29] Orhan Talu and Alan L. Myers. Rigorous thermodynamic treatment of gas adsorption. *AIChE Journal*, 34(11):1887–1893, 1988. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690341114.

[30] J Toth. Some Consequences of the Application of Incorrect Gas/Solid Adsorption Isotherm Equations. *Journal of Colloid and Interface Science*, 185(1):228–235, January 1997.

[31] Miles Cranmer, Dhananjay Ashok, and Johann Brehmer. MilesCranmer/PySR: v0.6.0, June 2021.

[32] Z. Konfrst. Parallel genetic algorithms: advances, computing trends, applications and perspectives. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 162–, April 2004.

[33] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[34] T. J. H. Vlugt, W. Zhu, F. Kapteijn, J. A. Moulijn, B. Smit, and R. Krishna. Adsorption of Linear and Branched Alkanes in the Zeolite Silicalite-1. *Journal of the American Chemical Society*, 120(22):5599–5600, June 1998. Publisher: American Chemical Society.

[35] Dan Richardson and John Fitch. The Identity Problem for Elementary Functions and Constants. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '94, pages 285–290, New York, NY, USA, August 1994. Association for Computing Machinery.

[36] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. Underspecification Presents Challenges for Credibility in Modern Machine Learning, November 2020. arXiv:2011.03395 [cs, stat].

[37] T. J. H. Vlugt, R. Krishna, and B. Smit. Molecular Simulations of Adsorption Isotherms for Linear and Branched Alkanes and Their Mixtures in Silicalite. *The Journal of Physical Chemistry B*, 103(7):1102–1118, February 1999. Publisher: American Chemical Society.

[38] Jeff Reback, jbrockmendel, Wes McKinney, Joris Van den Bossche, Matthew Roeschke, Tom Augspurger, Simon Hawkins, Phillip Cloud, gfyoung, Sinhrks, Patrick Hoefler, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, J. H. M. Darbyshire, Richard Shadrach, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Marco Edward Gorelli, Fangchen Li, Torsten Wörtwein, Matthew Zeitlin, Vytautas Jancauskas, Ali McMaster, and Thomas Li. pandas-dev/pandas: Pandas 1.4.3, June 2022.

[39] Patrick Kofod Mogensen, John Myles White, Asbjørn Nilsen Riseth, Tim Holy, Miles Lubin, Christof Stocker, Andreas Noack, Antoine Levitt, Christoph Ortner, Blake Johnson, Dahua Lin, Kristoffer Carlsson, Yichao Yu, Christopher Rackauckas, Josua Grawitter, Alex Williams, Ben Kuhn, Benoît Legat, Jeffrey Regier, cossio, Ron Rock, Thomas R. Covert, Benoit Pasquier, Takafumi Arakaki, Alexey Stukalov, Andrew Clausen, Arno Strouwen, and Benjamin Deonovic. JuliaNLSolvers/Optim.jl: v1.7.0, May 2022.

[40] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, March 2020. Number: 3 Publisher: Nature Publishing Group.

8 Supporting Information

8.1 Langmuir

The Langmuir isotherm model was originally presented in 1918 and remains in common use today [26]. While written and discovered in a simplified form in this work, the original model has a few specific terms with important meaning for the materials they pertain to.

$$q_e = \frac{q_m K_L p}{1 + K_L p} \text{ or } \theta_A = \frac{K_L p}{1 + K_L p} \tag{10}$$

The original Langmuir isotherm relates the volume adsorbed onto the surface (q_e), the adsorption strength (K_L), the gas pressure p and the maximum adsorption capacity (q_m) [24] [26]. The isotherm is often more simply written with only the fractional adsorption θ_A (what fraction of the surface is occupied by adsorbed molecules). However, in order to understand the expression that SR is likely to find, the isotherm should be rewritten as:

$$q_e = \frac{q_m K_A P}{1 + K_A P} \rightarrow q = \frac{c1 * p}{(c2 + p)} \tag{11}$$

This is because the nitrogen and methane datasets used in this work relate pressure to volume adsorbed, not including the total possible volume that could be adsorbed. Including that quantity in the numerator and dividing through by the Langmuir Constant leads to two unique constants that an SR algorithm might try to fit.

The basic premise of the Langmuir model is to represent the adsorbent (the surface atoms or molecules are adsorbing onto) as a simple surface with some number of free and full spots where the adsorbate could stick. The model is not concerned with rough or porous surfaces, molecules that may take up multiple “sites" or otherwise interact with each other after being adsorbed or, most importantly, any stacking of molecules. While this may seem somewhat idealistic and restricted, the Langmuir model has been shown to apply well to various adsorbents (surfaces) and adsorbates (gasses). Wang et. al. classify the Langmuir model as a chemical adsorption model because it is only concerned with mono-layer adsorption in which a chemical bond is formed between the adsorbent and adsorbate [24].

8.2 Dual-Site

In some cases, the Langmuir model can be very simply extended to fit new phenomena. Specifically, if there are two types of adsorption sites (with different properties) on a surface, adding another term of the Langmuir model may be enough to describe the adsorption process again. This new model would be called a “dual-site Langmuir" model and would be written as:

$$q = \frac{q_a K_A p}{(1 + K_A p)} + \frac{q_b K_B p}{(1 + K_B p)} \rightarrow q = \frac{c1 * p}{(c2 + p)} + \frac{c3 * p}{(c4 + p)} \tag{12}$$

In this case, because there are two unique Langmuir terms, there are also two unique terms for both the maximum volume that can be adsorbed (q_a and q_b) as well as the “Langmuir Constants" (K_A and K_B). Furthermore, this model is considered to be the ground truth for the isobutane dataset because the adsorbent material, the MFI zeolite, has two distinct adsorption sites for this adsorbate [37]. That is, the material that molecules are sticking to has two unique types of places for them to stick.

8.3 BET

Unlike the Langmuir model, the BET model is specifically designed with multi-layer adsorption in mind. Beyond the first layer, the Van der Waals force is what pulls the adsorbate towards the surface (and itself). Because of this, Wang et. al. classify the BET isotherm as a physical model as opposed to a chemical one [24]. The BET model can actually be written in a few ways based on what the maximum number of layers that can be adsorbed is believed to be. At one layer, it simplifies to the Langmuir model (which only models one layer) and at an arbitrary number of layers n it becomes significantly more complex (and unlikely to be found via SR). Fortunately the form at $n = \infty$ is more concise and can be simplified further:

$$\frac{c1 * (p/p_0)}{(1 - p/p_0) * (1 - p/p_0 + c_2(p/p_0))} \tag{13}$$

Note that while the p_0 term is written here, this constant is not provided to the SR algorithms, requiring that they fit a third constant (if they were to find this functional form).

8.4 Thermodynamic Constraint Functions

Three constraint checking functions (which follow from the three thermodynamic constraints introduced previously in section 1.4) were developed using the Python library SymPy which is designed for symbolic math [33]. Each function returns either true or false, depending on if its constraint is met or not. While these functions are useful for examining the expressions generated after a run, simply discarding an expression for failing one or more constraint during a run can severely hinder search potential by cutting off intermediary steps between better expressions that may also pass the constraints. Because of this, each function has a corresponding weight that allows it to act as a "soft constraint". Specifically, a constraint will not affect the score of an expression if it is passed but will multiply (worsen) that score if it is not. This approach (as implemented in PySR) is detailed in algorithm 1.

Algorithm 1 Modified genetic algorithm

Input: $tree, dataset, options$
Output: $score, loss$
  function SCOREFUNC($tree, dataset, options$)
    $loss \leftarrow$ evalLoss($tree, dataset, options$) ▷ Traditional error calculation
    $penalties \leftarrow options.penalties$
    if $penalties$ not empty **then** ▷ Skip slow calculation if possible
      $expr, var \leftarrow$ parseTree($tree$)
      for $p \in penalties$ and $tf \in thermoFunctions$ **do**
        if $tf(expr, var) =$ False **then**
          $loss \leftarrow loss * p$
        end if
      end for
    end if
    $score \leftarrow$ scoreFunc($loss, tree, options$) ▷ Factor in complexity
    return $score, loss$
  end function

Algorithm 2 Monotonic Non-decreasing Check

Input: $expr, var, start, stop$
Output: $passing$ (If the expression is monotonically non-decreasing)
function MONTONICNONDECREASING($expr, var, start, stop$)
  if $expr$ is constant **then**
    return True
  end if
  $turningPoints \leftarrow$ inflection points (zeros) of $expr$ ▷ Calculated using SymPy
  if $turningPoints$is empty **then**
    if $expr(stop) - expr(start) \geq 0$ **then** ▷ Measure slope
      return True
    else
      return False
    end if
  end if
  $turningPoints \leftarrow [start, turningPoints, stop]$ ▷ Include bounds
  for each sequential pair of points $p, q \in turningPoints$ **do**
    if $expr(q) - expr(p) \leq 0$ **then** ▷ Measure slope between zeros
      return False
    end if
  end for
  return True
end function

8.5 Implementation of Thermodynamic Constraints

In order to maintain parity between PySR and BMS, the Julia library PyCall was used. This allowed the same Python code that checked thermodynamic constraints in BMS to be used within the Julia code of the SymbolicRegression.jl library (the back-end of PySR). Beyond the concern of a fair comparison across platforms, PyCall showed itself to be somewhat necessary for this work. As of writing, there is no Julia library close to as full-featured as SymPy (and SymPy can have some serious limitations/difficulties). One method explored was using the Julia library MATLAB.jl to call MATLAB code from within Julia in a similar way to how PyCall works with Python code. While this did eventually work (and MATLAB does have robust symbolic math capabilities) it showed itself to be very difficult to set up due to intricacies in how the two languages store data. In the end, using MATLAB was not significantly faster than Python in this work (which is somewhat surprising).

One downside to the current structure of our version of SR.jl and the languages it relies on is that it must be run in distributed mode instead of threaded mode (meaning multiple processes must be created, requiring more overhead). This is due to the fact that PyCall expects only one Julia instance to attempt to use Python at once. With multiple processes, a new instance of the PyCall Julia package is created for each process, thus avoiding memory access conflicts.

8.6 PySR Data Collection

Although PySR handles many expressions, populations of those expressions and the Hall of Fame of the best expressions across all populations so far, accessing that information has proven difficult. PySR's goal each iteration is to produce a Pareto front showing what it has found to be the most accurate expression at each complexity level. This means the vast majority of expressions in a population are not shown (although some of them may be duplicates). While this is an issue, on the other hand, if an expression never makes it the Pareto front at any point during the run, by definition of the search it is not as important. That said, this limitation with output means that an expression such as the Langmuir isotherm may be generated and never shown due to poor accuracy relative to its peers.

In order to transform a few hundred Pareto front printouts to usable data, the full output text is parsed using Python and stored in a Pandas DataFrame [38]. The values collected from the raw text are the expression itself, score, loss, complexity, runtime, iteration and run number. While this is a significant amount of data already, there are a few interesting attributes yet to be calculated – specifically those relating to the simplified form of the expressions and those related to the thermodynamic constraints. In order to make further parsing manageable, once the simplified form of each expression is generated, only the most accurate expression with that form is retained. This step often reduces the number of rows in the DataFrame by about 1000x. The simplified form of each expression is calculated both with original (optimized/fit) and substituted constants. The new complexity of the simplified form may or may not be smaller

than the original complexity so both attributes are kept. Finally, the thermodynamic constraints that may have been used to guide the search are checked for each expression.

8.7 Identifying “interesting" expressions

An important component of this work is identification of interesting or meaningful expressions generated by the SR algorithms. An expression is most obviously meaningful if it is, or can be simplified to, the canonical form of an already known expression such as the Langmuir or BET adsorption isotherms. Unfortunately, determining if one expression is equivalent to another is a very difficult, and sometimes undecidable problem. In fact, while not true for this work due to limits on operators and constants used, Richardson's theorem shows that given the right set of operators and the transcendental numbers π and e, it may be impossible to show that one expression is equivalent to another [35]. It is also important to note that there is no incentive for SR to generate expressions that are easily read or understood – if it even had a concept of what expressions fit those criteria. Even when ground truth expressions are found, they can often be in unrecognisable forms. Furthermore, while SymPy does have a reasonably capable simplification function, it is of course not perfect and may encounter expressions it cannot handle. Because of these facts, this work uses a more complex function built on top of SymPy's simplification function and also accepts that much analysis will have to be done manually, at least for now.

8.8 Simplification Function

While SymPy is capable of simplifying symbolic math expressions and equations, there are definitely expressions that it may not simplify to their shortest form. Furthermore, while constants may be optimized per expression within both PySR and BMS, the expressions themselves are the true object of the search so constants are often left symbolic (e.g. c1, c2...). Because of this, in the interest of effectively simplifying without naively filling all symbolic constants with random numbers, symbolic constants are substituted with prime numbers. This avoids situations where one constant can completely remove another through multiplication or division. While this method has some advantages, there are also reasons to simplify using fit constants. Primarily, some expressions such as the BET isotherm have the same constant in multiple places, meaning it can be factored out. This should also be the case with well fit constants for newly generated expressions (especially those that turn out to be the same structure as previous isotherms). The conversion from the original BET isotherm to the form the simplification function finds is shown below:

$$\frac{c_1 * (p/p_0)}{(1 - p/p_0) * (1 - p/p_0 + c_2(p/p_0))} \rightarrow \frac{c_1 * p}{(c_2 * p + c_3 + p^2)} \tag{14}$$

Once constants are decided (either fit or primes), ordinary SymPy simplification is used to compress the expression as much as is reasonable. Then, if the resulting expression is rational (has only integer exponents and no division by zero), it can be written as a fraction and possibly simplified further. Specifically, if there is a common factor between the numerator and denominator because they are both degree 1 or larger, it may be possible to remove up to the difference in their degrees (see example below). While this situation is not guaranteed, having the capability to simplify it is still useful, especially when examining many generated expressions and differences between them.

$$\frac{3x}{2x + 3x^2} \rightarrow \frac{3}{2 + 3x} \tag{15}$$

Algorithm 3 Simplification Function

Input: $expr, vars, pars$
Output: can (Canonical form of expression)
function SIMPLIFY($expr, vars, pars$)
$expr \leftarrow$ expression
$vars \leftarrow$ variables
$pars \leftarrow$ parameters
for p **in** $pars$ **do**
$p \leftarrow$ prime number ▷ Substitute constants with unique primes
end for
Simplify using SymPy
if $expr$ is rational **then**
$num, denom \leftarrow expr$ as fraction ▷ Calculated using SymPy
if degree$(num) >$ degree$(denom)$ **then**
$factor \leftarrow$ leading term of num
else
$factor \leftarrow$ leading term of $denom$
end if
$expr \leftarrow \frac{num/factor}{denom/factor}$ ▷ Remove common factor
end if
return $expr$
end function

It is important to note that the simplification function will not always return the simplest possible form. While it does obviously reduce complexity, the original goal of the function was to have a canonical form for each expression such that whenever it is found, in whatever form, it will be simplified to that form. This effect is achieved by always writing the expression as a rational function if possible (and not simplifying further from that point). Because of this, when expressions are plotted on Pareto fronts (or their complexities are compared in general) the minimum complexity between the generated form and the simplified form is used.

8.9 Fitting Constants in PySR

By default, PySR uses Nelder-Mead optimization to fit constants for expressions as they are generated [31] [39]. Nelder-Mead is well suited to optimizing parameters for arbitrary generated expressions because it does not require any derivative or gradient information. Simply put, the algorithm evaluates the function at $n + 1$ points where n is the number of parameters. Then the next point is selected by finding the point with the highest function evaluation and looking to the opposite side of the rest of the points (a simplex). This iteratively moves the worst point to a likely better location, eventually moving towards a local optimum regardless of how the function is evaluated.

One downside to this method is that it is not guaranteed (or even expected) to find global minima. This issue is usually rectified by allowing for multiple starts from random locations, and selecting the best result. In PySR, each expression gets 8 attempts by default, randomizing the parameters between each. This is typically enough but there are occasionally cases where an expression should be much more accurate than it is due to poorly fitted constants.

To fit ground-truth expressions and check constants in post-processing, we also applied Nelder-Mead optimization using SciPy [40]. In this case, many more iterations were allowed to ensure ground truths and interesting functional forms had the best chance to show their effectiveness (or lack thereof).

8.10 Runtime

An important consideration when examining the effectiveness of the thermodynamic constraints in guiding SR is the impact on computation time. While not extreme, symbolic math (in SymPy) can be slow, especially compared to the otherwise efficient and optimized Julia code running behind the PySR front-end. The following plot shows the difference in iteration time (the time between one Pareto front and the next being printed by PySR) across different datasets, and more importantly, across different constraint penalties. It shows that if SymPy can avoid being loaded, runtime is not affected but that in all other cases, it is increased by about an order of magnitude.

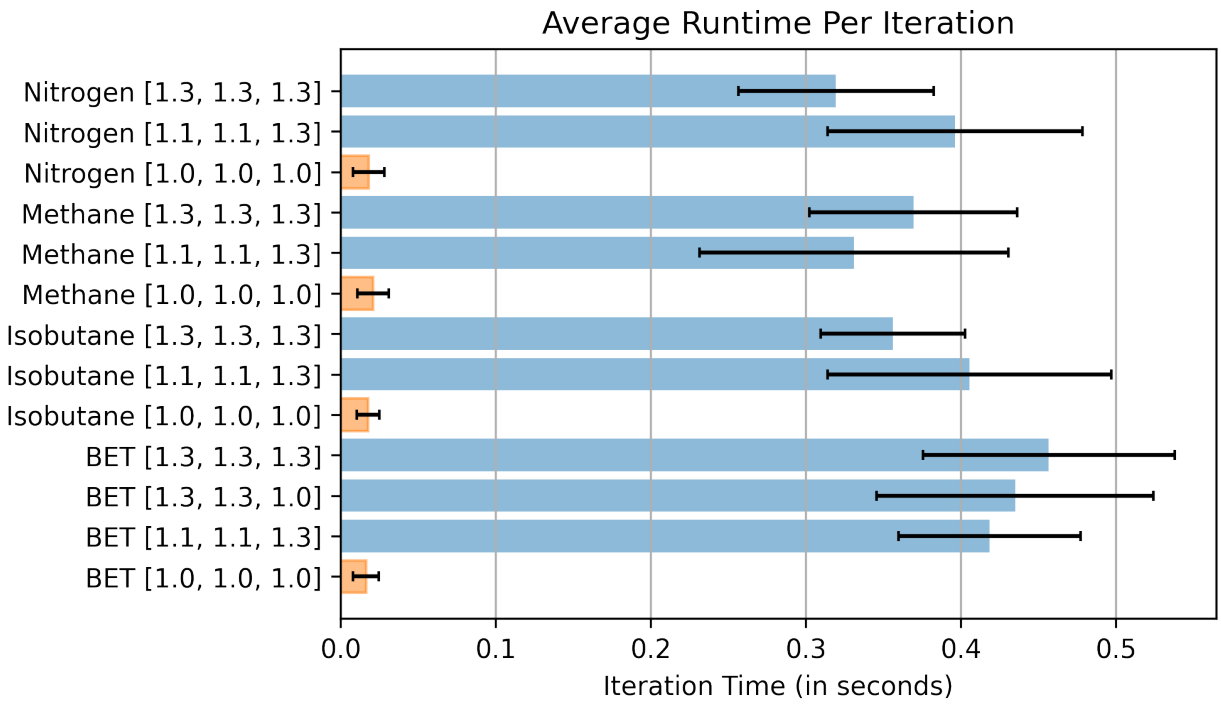

Figure 10: Average runtimes across all datasets and combinations of thermodynamic constraint penalties. Runs with all penalties set to 1.0 are highlighted in orange. Standard deviation is shown by error bars at the top of each bar.

8.11 Environment

Testing was done on both the batch and cpu2021 partitions of the UMBC High Performance Computing Facility (https://hpcf.umbc.edu/). This allowed for the longer runtimes and larger parameter exploration necessitated by adding thermodynamic constraints with variable penalties. It was also important to be able to run long term because BMS requires some build-up time to converge to the desired distribution, since it is based off of MCMC.

While BMS is entirely based in Python, SymbolicRegression.jl (the backend for PySR) is entirely in Julia. Because of the need to use SymPy, the Julia package PyCall.jl was used to allow Python code and libraries to be run by Julia (at the time this work was completed, symbolic math libraries in Julia could not evaluate the constraints considered in this work). To allow for parallel use of Python / SymPy from within Julia, PySR was run in distributed mode, necessitating more overhead than threaded mode.