Shape Arithmetic Expressions: Advancing Scientific Discovery Beyond Closed-Form Equations

Krzysztof Kacprzyk
University of Cambridge

Mihaela van der Schaar
University of Cambridge
The Alan Turing Institute

Abstract

Symbolic regression has excelled in uncovering equations from physics, chemistry, biology, and related disciplines. However, its effectiveness becomes less certain when applied to experimental data lacking inherent closed-form expressions. Empirically derived relationships, such as entire stress-strain curves, may defy concise closed-form representation, compelling us to explore more adaptive modeling approaches that balance flexibility with interpretability. In our pursuit, we turn to Generalized Additive Models (GAMs), a widely used class of models known for their versatility across various domains. Although GAMs can capture non-linear relationships between variables and targets, they cannot capture intricate feature interactions. In this work, we investigate both of these challenges and propose a novel class of models, Shape Arithmetic Expressions (SHAREs), that fuses GAM's flexible shape functions with the complex feature interactions found in mathematical expressions. SHAREs also provide a unifying framework for both of these approaches. We also design a set of rules for constructing SHAREs that guarantee transparency of the found expressions beyond the standard constraints based on the model's size.

1 INTRODUCTION

Throughout the centuries, scientists have used mathematical equations to describe the world around us.

From the groundbreaking work of Johannes Kepler on the laws of planetary motion in the 17th century to Albert Einstein's theory of relativity, mathematical equations have been instrumental in shaping the scientific landscape. They provide a concise representation of reality that is open to rigorous mathematical analysis. Recently, machine learning has been used to accelerate the process of discovering equations directly from data (Schmidt and Lipson, 2009).

Symbolic Regression Symbolic regression (SR) is an area of machine learning that aims to construct a model in the form of a *closed-form expression*. Such an expression is a combination of variables, arithmetic operations $(+, -, \times, \div)$, some well-known functions (trigonometric functions, exponential, etc), and numeric constants. For instance, $3 \sin(x_1 + x_2) \times e^{2x_3^2}$. Such equations, if concise, are interpretable and well-suited to mathematical analysis. These properties have led to applications of SR in many areas such as physics (Schmidt and Lipson, 2009), medicine (Alaa and van der Schaar, 2019), material science (Wang et al., 2019), and biology (Chen et al., 2019). Traditionally Genetic Programming (Koza, 1994) has been used for this task (Bongard and Lipson, 2007; Schmidt and Lipson, 2009; Cranmer, 2020; Stephens, 2022). Recently, this area attracted a lot of interest from the deep learning community. Neural networks have been used to prune the search space of possible expressions (Udrescu and Tegmark, 2020; Udrescu et al., 2021) or to represent the equations directly by modifying their architecture and activation functions (Martius and Lampert, 2017; Sahoo et al., 2018). A different approach is proposed by Biggio et al. (2021), where a neural network is pre-trained using a curated dataset. A similar approach is employed by D'Ascoli et al. (2022); Kamienny et al. (2022). Methods using deep reinforcement learning (Petersen et al., 2021) have also been proposed, as well as a hybrid of the two approaches (Holt et al., 2023). Symbolic regression is usually validated on synthetic datasets with closed-form ground truth equations (Udrescu and Tegmark,

2020; Petersen et al., 2021; Biggio et al., 2021). However, as we investigate in Section 2.1, closed-form functions are often inefficient in describing some relatively simple relationships. They either fit the data poorly or produce overly long expressions. They are also not compatible with categorical variables.

Generalized Additive Models Generalized Additive Models (GAMs) (Hastie and Tibshirani, 1986; Lou et al., 2012) are widely used transparent methods. They model the relationship between the features x_i and the label y as

$$g(y) = f_1(x_1) + \ldots + f_n(x_n) \qquad (1)$$

where g is called the *link functions* and the f_k's are called *shape functions*. These models allow arbitrary complex shape functions but exclude more complicated interactions between the variables. They are deemed transparent as each function f_k can be plotted; thus, the contribution of x_k can be understood. Initially, splines and other simple parametric functions were used as shape functions. In recent years, many different classes of functions were proposed, including tree-based (Lou et al., 2012), deep neural networks (Agarwal et al., 2021; Radenovic et al., 2022) or neural oblivious decision trees (Chang et al., 2022). GAMs have also been extended to include pairwise interactions (known as GA^{2}M) that can be visualized using heat maps (Lou et al., 2013). The main disadvantage of GAMs is their inability to model more complicated interactions, for instance, $\frac{x_1 x_2}{x_3}$ (see Section 2.2).

Transparency of Closed-Form Expressions A model is considered *transparent* if by itself it is understandable—a human understands its function (Barredo Arrieta et al., 2020). The transparency of symbolic regression can be compromised if the found expressions become too complex to comprehend. In many scenarios, an arbitrary closed-form expression is unlikely to be considered transparent. Most of the current works limit the complexity of the expression by introducing a constraint based on, e.g., the number of terms (Stephens, 2022), the depth of the expression tree (Cranmer, 2020), or the description length (Udrescu et al., 2021). Although these metrics often correlate with the difficulty of understanding a particular equation, size does not always reflect the equation's complexity as it does not focus on its semantics. Some recent works introduce a recursive definition of complexity that takes into account the type and the order of operations performed (Vladislavleva et al., 2009; Kommenda et al., 2015). Although they are a step in the right direction, they are not grounded in how the model will be analyzed, and thus, it is not clear if they capture how comprehensible the model is (further discussion in Appendix D).

Contributions and Outline In Section 2, we investigate the limitations of SR and GAMs. In Section 3, we introduce a novel class of models called **SH**ape **AR**ithmetic **E**xpressions (SHAREs) that combine GAM's flexible shape functions with the complex feature interactions found in closed-form expressions thus providing a unifying framework for both approaches. In Section 4, we introduce a new kind of *rule-based* transparency that goes beyond the standard constraints based on the model's size and is grounded in the way the model is analyzed. We also investigate the theoretical properties of transparent SHAREs. Finally, we demonstrate their effectiveness through experiments in Section 5.

2 LIMITATIONS OF CURRENT APPROACHES

2.1 Symbolic Regression Struggles With Expressions That Are Not Closed-Form.

Symbolic regression excels in settings where the ground truth is a closed-form expression (Udrescu and Tegmark, 2020). However, its effectiveness becomes less certain when applied to scenarios with no underlying closed-form expressions. Some phenomena do not have a closed-form expression (e.g., non-linear pendulum), and many functions in physics are determined experimentally rather than derived from a theory and are not inherently closed-form (e.g., current-voltage curves, drag coefficient as a function of Reynolds number, phase transition curves). This is even more relevant in life sciences, where the complexity of the studied phenomena makes it more difficult to construct theoretical models. We claim symbolic regression struggles to find compact expressions for certain relatively simple univariate functions.

Example: Stress-Strain Curves To illustrate our point, we try to fit a symbolic regression model to an experimentally obtained stress-strain curve. We use data of stress-strain curves in steady-state tension of aluminum 6061-T651 at different temperatures obtained by Aakash et al. (2019). Figure 1 shows a sample of these curves. These functions are relatively simple as they can be divided into a few interpretable segments representing different behaviors of the material. The first part is linear and corresponds to elastic deformation. It ends at a point called yield strength. The rest corresponds to plastic deformation. For aluminum at 20°C, we can distinguish the part of the curve when the stress increases called strain hardening. It achieves a maximum called the ultimate strength, and then it starts decreasing in a process called necking until fracture.
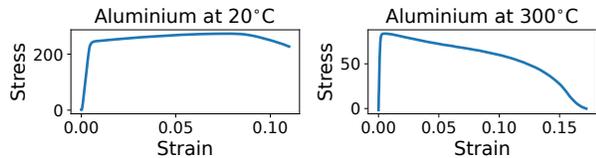
Figure 1: Stress-strain curves of aluminum at different temperatures

We use a symbolic regression library PySR (Cranmer, 2020) to fit the stress-strain curve of aluminum at 300°C. We fit the model and present several of the found expressions in Table 1. The size of a closed-form expression is defined as the number of terms in its representation. For instance, $\sin(x + 1)$ has complexity 4 as it contains four terms: $\sin$, $+$, x, and 1. We can see that small programs do not fit the data well. A good fit is achieved only by bigger expressions. However, such expressions are much less comprehensible when analyzed based on their symbolic representation, and thus, their utility is diminished.

Table 1: A few of the equations discovered by Symbolic Regression when fitted to the stress-strain curve of aluminum at 300°C. Equations in the last three rows are too long to fit in the table. We reproduce them below the table

Equation	Size	R^2 score
$y = 63.3e^{-x}$	4	0.163
$y = 78.8 - 285x$	5	0.529
$y = 74.9 \cos\left(7.78x\right)$	6	0.733
$y = 71.2 \cos\left(\frac{x}{x-0.277}\right)$	8	0.750
$y = 147 \cos\left(8.58x - 0.429\right) - 71.5$	10	0.770
Equation 2	11	0.836
Equation 3	15	0.933
Equation 4	18	0.970

$$y = -428x + 428 \cos\left(0.0711 \log\left(x\right)\right) - 324 \quad (2)$$

$$y = 428 \cos\left(3.31x - 0.0751 \log\left(1.16x\right)\right) - 320 \quad (3)$$

$$y = 168 \cos\left(\left(\left(7.23 - \cos\left(e^{-421x}\right)\right)\left(x - 2.03\right)\right)\right) - 88.1 \quad (4)$$

2.2 GAMs Cannot Model Complex Interactions

The main disadvantage of GAMs is that they are poor at modeling more complicated, non-additive interactions (involving 3 or more variables). Such interactions occur frequently in real life. For instance, many equations from physics involve multiplying a few variables together. To illustrate this point, we choose a few simple equations from the Feynman Symbolic Re-

gression Database (Udrescu and Tegmark, 2020) and compare the performance of GAMs and GA^{2}Ms with a black-box machine learning model. We implement GAMs and GA^{2}Ms using Explainable Boosted Machines (EBMs) (Lou et al., 2012, 2013) and choose XGBoost (Chen and Guestrin, 2016) for a black-box model.

Choice of Equations We choose equations so that they represent a variety of non-additive interactions between variables (see Table 2). Equations I.8.14 and I.29.16 describe the Euclidean distance in two dimensions and the Law of Cosines. Both of them involve a square root of a sum of terms. Equations I.12.2, I.18.12, I.32.5, II.2.42 can be used to describe an electric force between charged bodies, a torque, a rate of radiation of energy, and a heat flow. All of them are either products or fractions of products. Equations I.6.20b and I.40.1 describe a Gaussian distribution and a particle density. They both contain exponential functions. Lastly, equation I.12.11 describes a Lorentz force via a sum of products, one of which contains a trigonometric function.

Results We report the results in Table 2. The performance of GAMs is much lower than the performance of a full-capacity model (whose R^2 score is close to 1.0 as no noise was added to the dataset). The gap between GAM and XGBoost is partially closed by adding pairwise interactions in GA^{2}Ms. This dramatically improves the score in some cases (e.g., equation I.8.14) but still underperforms in others (e.g., equation I.32.5). It is important to note that pairwise interactions decrease the comprehensibility of the model. In particular, 2D heatmaps are more challenging to understand than plots of univariate functions, and the individual shape functions cannot be analyzed independently. As the shape functions have overlapping sets of arguments, we may have to analyze many shape functions at the same time to understand the model.

3 SHAPE ARITHMETIC EXPRESSIONS

In this section, we introduce a new type of machine learning model that connects symbolic regression's ability to model interactions with GAM's power of efficiently describing univariate functions by plots. This new family of models addresses the issues of both GAMs and symbolic regression that we discussed in the previous section.

Inspired by the GAM literature, we define a set of *shape functions* $\mathcal{S}$, where each $s \in \mathcal{S}$ is a univariate function $s : \mathbb{R} \to \mathbb{R}$. This might be, for instance, a

Table 2: Performance of additive models (GAM and GA^{2}M) compared to a full capacity model (XGBoost) on datasets from the Feynman Symbolic Regression Database containing complex variable interactions. We show the mean R^2 score and a standard deviation in the brackets over 10 cross-validation splits.

Eq. Number	Equation	GAM	GA^{2}M	XGBoost
I.6.20b	$f = e^{-\frac{(\theta-\theta_1)^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$	0.731 (.010)	0.896 (.004)	0.997 (.000)
I.8.14	$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$	0.229 (.011)	0.966 (.000)	0.989 (.000)
I.12.2	$F = \frac{q_1 q_2}{4\pi\epsilon r^2}$	0.676 (.011)	0.950 (.003)	0.993 (.001)
I.12.11	$F = q(E_f + Bv\sin(\theta))$	0.675 (.004)	0.955 (.001)	0.996 (.000)
I.18.12	$\tau = rF\sin(\theta)$	0.760 (.002)	0.981 (.000)	0.999 (.000)
I.29.16	$x = \sqrt{x_1^2 + x_2^2 - 2x_1 x_2 \cos(\theta_1 - \theta_2)}$	0.298 (.007)	0.902 (.002)	0.983 (.001)
I.32.5	$P = \frac{q^2 a^2}{6\pi\epsilon c^3}$	0.444 (.015)	0.835 (.009)	0.988 (.001)
I.40.1	$n = n_0 e^{-\frac{magx}{k_b T}}$	0.736 (.003)	0.899 (.003)	0.981 (.001)
II.2.42	$P = \frac{\kappa(T_2-T_1)A}{d}$	0.615 (.006)	0.937 (.002)	0.990 (.000)

set of cubic splines or univariate neural networks. Let $\mathcal{B} = \{+, -, \div, \times\}$ be a set of binary operations. Let us denote real variables as x_i. We introduce *Shape Arithmetic Expression* (SHARE) as a mathematical expression that consists of a finite number of shape functions, binary operations, variables, and numeric constants. For instance, see Equation 5, where $s_1, s_2, s_3 \in \mathcal{S}$ are the shape functions and need to be plotted next to the equation to understand the whole model.

$$s_1(x_4 s_2(x_2)) + \frac{x_1}{s_3(x_3) - 2.3} \qquad (5)$$

Remark 1. Any GAM is an example of a SHARE. If we choose $\mathcal{S}$ to be a set of some well-known functions (e.g., $\mathcal{S} = \{\sin, \cos, \exp, \log\}$) then closed-form expression can also be considered SHAREs. In general, however, $\mathcal{S}$ is supposed to be a flexible family of functions that are fitted to the data and are meant to be understood visually.

Formally, we represent SHAREs as expression trees (types of graphs) where each node is either a binary operation $b \in \mathcal{B}$ (with two children), a univariate function $s \in \mathcal{S}$ (with one child), a variable or a numeric constant (as leaves). Equation 5 represented as a tree can be seen in Figure 2. We borrow the terminology from SR literature and define the size of a SHARE as the number of nodes in its expression tree. The depth of a SHARE is defined as the depth of its expression tree.

Why Univariate Functions? We decided to use only univariate functions for two reasons: they are easy to understand, and they are sufficient. Firstly, they are easy to comprehend because they can always be plotted. While analyzing them, we have to keep track of only one variable, and we can characterize them using monotonicity (i.e., where the function is increasing or
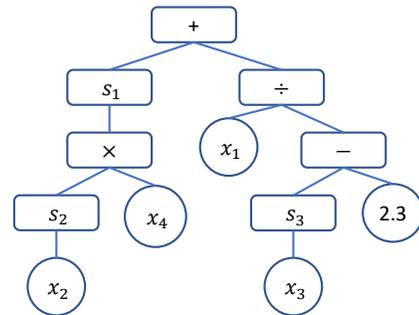
Figure 2: Shape Arithmetic Expression represented as a tree.

decreasing). Univariate functions are also much easier to edit in case we want to fix the model. Secondly, the Kolmogorov–Arnold representation theorem (Kolmogorov, 1957) states that for any continuous function $f : [0,1]^n \to \mathbb{R}$, there exist univariate continuous functions $g_q, \phi_{p,q}$ such that

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} g_q\left(\sum_{p=1}^{n} \phi_{p,q}(x_p)\right) \qquad (6)$$

That means in principle, for expressive enough shape functions, SHAREs should be able to approximate any continuous function. However, SHAREs of that form would not necessarily be very transparent. We discuss the transparency of SHAREs in the next section.

4 TRANSPARENCY

As explained in Section 1, the transparency of symbolic regression can be compromised if the found expressions become too complex to comprehend. In many scenarios, an arbitrary closed-form expression is unlikely to be considered transparent. Note that

any fully connected deep neural network with sigmoid activation functions is technically a closed-form expression. As SHAREs extend SR, they inherit the same problem. Current works introduce constraints that are not grounded in how the model will be analyzed. Although they are correlated with the difficulty of understanding the model, they are not based on any assumptions of how the model is actually understood. Therefore, it is unclear whether they capture how comprehensible the model is. That includes constraints based on model size (Stephens, 2022; Cranmer, 2020; Udrescu et al., 2021) and even recent semantic constraints (Vladislavleva et al., 2009; Kommenda et al., 2015) (further discussion on SR constraints in Appendix D).

Understanding by Decomposing: Rule-based Transparency We approach the problem more systematically. Motivated by research on human understanding and problem solving (Newell et al., 1958; Simon, 1962; Navon, 1977; Simon, 1996), we assume that in certain scenarios *understanding a complex expression involves decomposing it into smaller expressions and understanding them and the interactions between them.* Thus, the model can be decomposed into simpler terms and understood from the ground up—provided the expressions remain transparent throughout. This is in agreement with recent research in XAI that highlights *decomposability* as a crucial factor for transparency, enabling more interpretable and explainable machine learning methods (Barredo Arrieta et al., 2020). Thus, we define transparency implicitly by proposing two general rules for building machine learning models in a *transparency-preserving* way, and we justify why they may be sufficient for achieving transparency in certain scenarios. These rules, in turn, allow us to define a subset of transparent SHAREs. Note, we use $\circ$ to denote standard function composition, i.e., $(f \circ g)(x) = f(g(x))$.

Rule 1 (Univariate composition)**.** Let s be any univariate function. $s(x_i)$ is transparent, where x_i is any variable. If f is transparent then $s \circ f$ is also transparent.

Rule 2 (Disjoint binary operation)**.** Let $b \in \mathcal{B}$ be a binary operation. If f and g are transparent and have disjoint sets of arguments, then $b \circ (f, g)$ is also transparent.

Rationale for Rule 1 Let s be any univariate function. Then $s(x_i)$ is transparent because we can visualize it and create a mental model of its behavior. Let us now consider a transparent function f. As it is transparent, we should have a fairly good understanding of the properties of f. For instance, what range of values it attains or whether it is monotonic for certain sub-

sets of the data. As we can visualize s, it is reasonable to expect that we can infer these properties about $s \circ f$ as well. We can analyze s and f separately and then use that knowledge to analyze $s \circ f$.

Rationale for Rule 2 Let $b \in \mathcal{B}$ be a binary operation and let f and g be transparent functions with non-overlapping sets of arguments. As these functions are transparent, we can understand their various properties. As they do not have any common variables, they act independently. Thus, we can combine them using the binary operation b and directly use the previous analysis to understand the new model $b \circ (f, g)$. Thus, it is considered transparent. An example of a property that conforms to this rule is the image of the function. If we know the images of two functions as two intervals and their sets of arguments are disjoint, then we can straightforwardly calculate the image of any combination of these functions (i.e., sum, difference, product, or ratio) using interval arithmetic. This is not possible if the variables overlap—interval arithmetic can only guarantee us a superset of the function's image. See Appendix D for a detailed discussion.

Disjoint Binary Operations in Physics Although Rule 2 may seem like a strong constraint, many common closed-form equations used to describe natural phenomena can be constructed by following the two rules. In particular, 86 out of 100 equations from Feynman Symbolic Regression Database (Udrescu and Tegmark, 2020) can be expressed in that form. Thus, in many cases, the space of transparent models should be rich enough to find a good fit.

Restricting the Search Space The current definition of SHAREs contains certain redundancies. For instance, it allows for a direct composition of two shape functions. This unnecessarily complicates the model as the composition of two shape functions is approximately just another shape function (given that the class of shape functions is expressive enough). As any binary operation applied to a function and a constant can be interpreted as applying a linear function, we can remove the constants without losing the expressivity of SHAREs (given that the shape functions can approximate linear functions).

We can now use these two rules and the above observations to define transparent SHAREs.

Definition 1. A transparent SHARE is a SHARE that satisfies the following criteria:

- Any binary operator is applied to two functions with disjoint sets of variables.
- The argument of a shape function cannot be an output of another shape function, i.e., $s_1(s_2(x))$ is not

allowed.

- It does not contain any numeric constants.

Remark 2. By this definition, any GAM is a transparent SHARE. This is consistent with the fact that GAMs are generally considered transparent models (Hastie and Tibshirani, 1986; Caruana et al., 2015).

Transparent SHAREs have several useful properties. The following proposition demonstrates that there is no need to arbitrarily limit the size of the expression tree (as might be the case for many SR algorithms) as the depth and the number of nodes of a transparent SHARE are naturally constrained.

Proposition 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a transparent SHARE. Then*

- *Each variable node appears at most once in the expression tree.*
- *The number of binary operators is $d-1$, where d is the number of variable nodes (leaves)*
- *The depth of the expression tree of f is at most $2n$.*
- *The number of nodes in the expression tree of f is at most $4n-2$.*

Proof. Appendix A. □

For comparison, the expression tree of a GAM has $3n - 1$ nodes. That demonstrates that transparent SHAREs are not only naturally constrained, but even the largest possible expressions are not significantly longer than the expression for a GAM, even though it can capture much more complicated interactions. The immediate corollary of this proposition is useful for the implementation.

Corollary 1. *SHARE f satisfies Rule 2 if and only if each variable appears at most once in its expression tree.*

Closed-Form Equations Considered as Transparent SHAREs. Consider equation I.34.14 from the Feynman Symbolic Regression Database (Udrescu and Tegmark, 2020),

$$\omega = \frac{1 + v/c}{\sqrt{1 - v^2/c^2}} \omega_0, \qquad (7)$$

where we consider ω, v, c, ω_0 as variables. Note that this representation violates Rule 2, and it may be difficult to understand this equation in this format. For instance, comprehending how changing v impacts ω is challenging as we have two terms (numerator and denominator) that are not independent. Instead, we can rewrite this equation as a transparent SHARE

$$\omega = s_1 \left(\frac{v}{c} \right) \omega_0, \qquad (8)$$
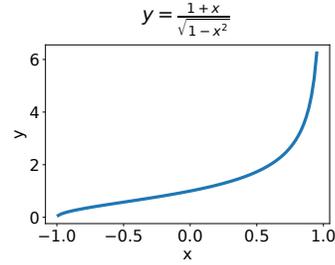
Figure 3: Plot of a function $s_1(x) = \frac{1+x}{\sqrt{1-x^2}}$.

where $s_1(x) = \frac{1+x}{\sqrt{1-x^2}}$ and can be visualized as in Figure 3.

We can now understand the properties of s_1. For instance, it is defined on $(-1, 1)$; it is increasing, starts concave, and has an inflection point at $x = 0$. After that, it is convex until an asymptote at $x = 1$, where it approaches $+\infty$. By understanding the properties of s_1, we can easily understand the behavior of the whole Equation 8.

5 SHARES IN ACTION

In this section, we perform a series of experiments to show how SHAREs work in action.[1] First, we justify our claim that SHAREs extend GAMs (Section 5.1) and SR (Section 5.2). Then, we show an example that cannot be fitted by GAM or by SR. For every experiment, we show the Pareto frontier of the found expressions with respect to R^2 score and the number of shape functions, i.e., the best expression for a given number of shape functions. For details about the experiments, including additional experiments on real datasets, see Appendix C.

Implementation For illustrative purposes, we propose a simple implementation of SHAREs utilizing nested optimization. The outer loop employs a modified genetic programming algorithm (based on gplearn (Stephens, 2022) used for symbolic regression), while the inner loop optimizes shape functions as neural networks via gradient descent. Although not a key contribution due to its limited scalability, this implementation demonstrates SHAREs' potential to outperform existing transparent methods and enhance interpretability, given more efficient optimization algorithms. We note that optimization of transparent models is usually harder than that of black boxes (Rudin et al., 2022). For further implementation details, see Appendix B.

---

[1]The code for all experiments can be found at https://github.com/krzysztof-kacprzyk/SHARES

5.1 SHAREs Extend GAMs

As we discussed earlier, GAMs (without interactions) are examples of SHAREs. That means that, in particular, if we have a dataset that can be modeled well by a GAM, SHAREs should also model it well. To verify this, we generate a semi-synthetic dataset inspired by the application of GAMs to survival analysis described by Hastie and Tibshirani (1995). In this work, GAMs are used to model the risk scores of patients taking part in a clinical trial for the treatment of node-positive breast cancer. We choose three of the covariates considered and assume that the risk score (log of hazard ratio) can be modeled as a GAM of age, body mass index (BMI), and the number of nodes examined. We recreate the shape functions to resemble the ones reproduced in the original paper. Then we choose the covariates uniformly from the prescribed ranges and calculate the risk scores.

We fit SHAREs to this dataset and show the results in Figure 4. Each row shows the best equation with the corresponding number of shape functions, and the shape functions of the equation with three shape functions are shown at the bottom of the figure.

#s	Equation	R^2 score
0	$y = \frac{x_{\text{nodes}}}{x_{\text{age}} x_{\text{bmi}}}$	-0.267
1	$y = x_{\text{age}} s_3(x_{\text{bmi}})$	0.630
2	$y = s_2(x_{\text{age}}) + s_3(x_{\text{bmi}})$	0.847
3	$y = s_1(x_{\text{nodes}}) + s_2(x_{\text{age}}) + s_3(x_{\text{bmi}})$	0.999
4	$y = s_0(s_1(x_{\text{nodes}}) + s_2(x_{\text{age}}) + s_3(x_{\text{bmi}}))$	0.989
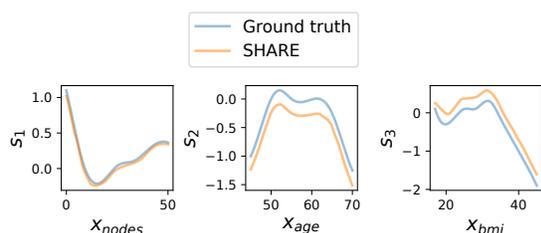
Figure 4: Results of fitting SHAREs to the risk score data. Each row in the table shows the best found equation with the corresponding number of shape functions (#s). At the bottom, shape functions from the fourth row compared to the ground truth.

We see that the equation in the fourth row achieves a high R^2 score. It is also in the desired form. When we plot the shape functions in Figure 4 we see that they match the ground truth well (the vertical translation is caused by the fact that shape functions can always be translated vertically).

5.2 SHAREs Extend SR

Torque Equation Consider equation I.18.12 (Table 2) used to calculate torque, given by $\tau = rF\sin(\theta)$. We sample 100 rows from the Feynman dataset corresponding to this expression and we run our algorithm. Each row of the table in Figure 5 shows the best equation with the corresponding number of shape functions. The bottom part of the figure shows the shape functions of the equations in the second and fourth rows.

# s	Equation	R^2 score
0	$\tau = r$	-0.115
1	$\tau = rFs_3(\theta)$	0.999
2	$\tau = s_1(r)Fs_3(\theta)$	0.999
3	$\tau = s_1(r)s_2(F)s_3(\theta)$	0.999
4	$\tau = s_0(s_1(r)s_1(F)s_3(\theta))$	0.999
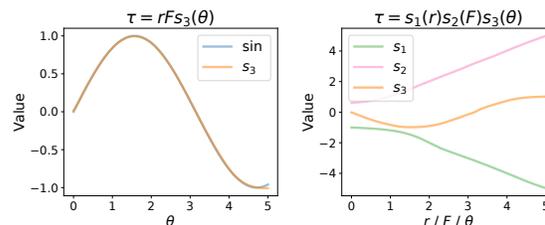
Figure 5: Equations found by fitting SHAREs to a torque equation $\tau = rF\sin(\theta)$. Each row in the table shows the best-found equation with the corresponding number of shape functions (#s). Bottom left panel: shape function from the second row compared to ground truth. Bottom right panel: shape functions from the fourth row.

The equation that is symbolically equivalent to the ground truth is in the second row, $\tau = rFs_3(\theta)$. It achieves a nearly perfect R^2 score. By plotting s_3, we can verify that it matches sin function well (Figure 5, bottom left panel).

Shape Functions of the Longer Equations Consider the expression in row 4 from the table in Figure 5, $\tau = s_1(r)s_2(F)s_3(\theta)$. It might look complicated because it contains three shape functions. But, if we inspect s_1 and s_2 (Figure 5, bottom right panel), we see that they are linear functions. We can fit straight lines to extract their slopes and intercepts and put them into the found SHARE to get a simple expression $-(0.98r + 0.07)(F + 0.02)s_3(\theta)$.

5.3 SHAREs Go Beyond SR and GAMs

We consider the following problem. Given m grams of water (in a liquid or solid form) of temperature t_0 (in $°C$), what would be the temperature of this water (in a solid, liquid, or gaseous form) after heating it

with energy E (in calories). We restrict the initial temperature to be from -100 °C to 0 °C. This is a relatively simple problem with only 3 variables but we will show that both GAMs and SR are not sufficient to properly (and compactly) model this relationship.

GAMs   First, we fit GAMs without interactions using EBM (Lou et al., 2012; Nori et al., 2019). The shape functions of EBM are presented in Figure 6. The R^2 score on the validation set is 0.758. We can also see that the two of the shape functions are very jagged. This makes it difficult to gain insight into the studied phenomenon.
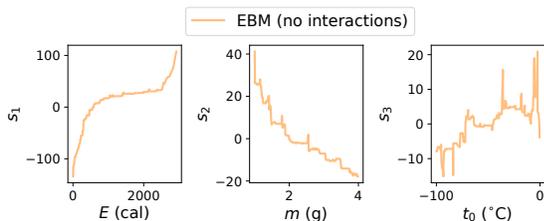
Figure 6: Shape functions from the GAM fitted to the temperature dataset

GA²Ms   Now, we fit GAMs with pairwise interactions (Lou et al., 2013), once again using the EBM algorithm. The shape functions of EBM are presented in Figure 9 in Appendix C.3. Although the R^2 score has been improved to 0.875, EBM's transparency is reduced even further by adding pairwise interactions.

Symbolic Regression   We fit symbolic regression using the PySR library (Cranmer, 2020). We limit the size of the program to 40, and we present the results in Table 3.

Table 3: Equations found by SR when fitted to the temperature data. The last four equations do not fit in the table; they are reproduced in Appendix C.3.

Equation	Size	R^2 score
$y = 13.5 \log(E)$	4	0.384
$y = \frac{0.193E}{m}$	5	0.485
$y = 39.4 \log\left(\frac{E}{m}\right) - 141$	8	0.733
Appendix C.3 *Equation 9*	17	0.768
Appendix C.3 *Equation 10*	23	0.817
Appendix C.3 *Equation 11*	33	0.841
Appendix C.3 *Equation 12*	40	0.867

We can see that shorter equations achieve a relatively low performance, no more than 0.768. This is comparable to a GAM without interaction terms. The first equation to include the term t_0 appears only when the

size is 17. It's already too long to include in the table. It looks like this:

$$y = 74.0 \cos\left(\log\left(\frac{0.739E}{m} + 19.1\right)\right) + 39.1 + \frac{t_0}{E}$$

Only the most complex equations give us a performance comparable to a GAM with interactions: 0.867. The last equation from the table is shown below. We argue that its complexity hinders its transparency.

$$y = \frac{t_0}{\log(E)} - 1.72 e^{e^{\cos\left(\frac{0.0103E}{m}\right)}} + 80.1 + 56.3$$
$$\times \cos\left(\log\left(\frac{0.58E}{m} + 31.7 \cos\left(\frac{0.021E}{m} + 0.93\right)\right)\right)$$

SHAREs   We finally fit SHAREs to the temperature data. The found expressions are shown in Figure 7. We immediately see a very good performance from all models apart from the one not using any shape functions at all. The scores are also much better than the scores achieved by GAMs (with or without interactions) and SR. Let us investigate the equation in the third row; the shape functions are presented in Figure 7 (right panel).

We note that the expression $s_1\left(\left(\frac{E}{m} + s_2(t_0)\right)\right)$ has a better performance than GAMs and SR, a more compact symbolic representation than SR, and simpler shape functions than GAM. This exemplifies how, by combining the advantages of GAMs and SR, we can address their underlying limitations. Let us see how this particular SHARE can aid in understanding the phenomenon it fits.

Analysis   We recognize that s_1 is contingent on the energy-to-mass ratio, which is offset by a function of the initial temperature, t_0. As shown in Figure 7's right panel, s_2 appears linear. Replacing s_2 with an equivalent linear function and adjusting the equation gives us: $t = s_1\left(\frac{E}{m} + 0.49t_0 + 7.39\right)$. Analyzing s_1, we find that without energy input, $\frac{E}{m} + 0.49t_0 + 7.39$ ranges from -41.61 ($t_0 = -100$) to 7.39 ($t_0 = 0$), aligning with the first linear part of the s_1 curve. Increasing energy per mass initially raises the temperature linearly to 0 °C, then plateaus, characteristic of an ice-water mixture. When all ice melts, the temperature rises linearly again to 100 °C, remaining constant until all water evaporates, after which steam temperature again increases linearly.

Quantitative Insights   The shape functions also provide quantitative insights. The slopes of s_1's linear parts approximate the specific heat capacities of ice, water, and steam. The constant parts' widths estimate the heat of fusion and vaporization. We compare

#s	Equation	R^2 score
0	$t = m$	-3.513
1	$t = s_1\left(\frac{E}{m} + t_0\right)$	0.970
2	$t = s_1\left(\left(\frac{E}{m} + s_2(t_0)\right)\right)$	0.999
3	$t = s_1\left(\frac{E + s_2(t_0)}{s_0(m)}\right)$	0.988
4	$t = s_1\left(\frac{E}{s_0(s_3(m)s_2(t_0))}\right)$	0.942
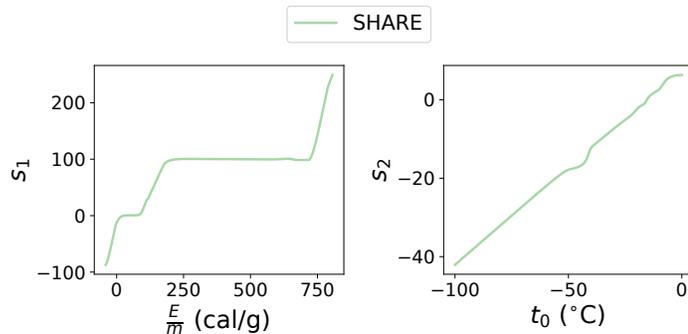
Figure 7: SHAREs found for the temperature dataset. Each row in the table shows the best expression with the corresponding number of shape functions (#s). Right panel: shape functions of the SHARE $s_1\left(\left(\frac{E}{m} + s_2(t_0)\right)\right)$ (third row).

these estimates from s_1 with the physical ground truth in Table 4. We highlight that it is impossible to draw these insights from the fitted GAM (see Figure 6) and from the found closed-form expressions (see Table 3).

Table 4: Properties of water extracted from shape function s_1 compared to the ground truth.

Property	From s_1	Ground truth
Spec. heat cap. of ice $\left(\frac{cal}{g°C}\right)$	0.53	0.50
Spec. heat cap. of water $\left(\frac{cal}{g°C}\right)$	1.01	1.00
Spec. heat cap. of steam $\left(\frac{cal}{g°C}\right)$	0.50	0.48
Heat of fusion $\left(\frac{cal}{g}\right)$	78.85	79.72
Heat of vaporization $\left(\frac{cal}{g}\right)$	540.91	540.00

6 DISCUSSION

Categorical Variables The default way current SR algorithms take care of categorical variables is through one-hot encoding. However, that significantly increases the number of variables and makes the resulting expressions less legible. SHAREs offer a natural way of extending SR to settings with categorical variables by always passing such variables through a shape function first. It assigns a number to each class and can be visualized using, for instance, a bar plot. This can prove useful when we investigate phenomena comprising different objects with some possibly unknown latent properties. In that setting, SHAREs may naturally learn a shape function that differentiates between those objects and their properties. An example of such a property may be a friction coefficient, the moment of inertia, or something that does not relate naturally to commonly used concepts.

Applications SHAREs can be beneficial in settings where transparent models are needed or preferred, such as risk prediction in healthcare and finance. However, we mainly envision its usage in AI applications

for scientific discovery: AI4Science. We believe that we need to add more flexibility to our models for AI4Science to advance beyond the synthetic experiments based on simple physical equations (as is often the case for symbolic regression). Transparent SHAREs add this flexibility without compromising the comprehensibility. AI4Science is also likely to contain multiple scenarios where adequate resources (time and attention) can be spent on analyzing and understanding the models thoroughly from the ground up in a way that is facilitated by the rules we propose. We also hope this new kind of *rule-based transparency* can inspire novel, more systematic, notions of transparency grounded in the way the model is understood.

Limitations of Rule-based Transparency The two rules for building machine learning models in a transparency-preserving way are a step towards more systematic notions of transparency. We show how they are realized in many physics equations and GAMs. They also offer a natural way to constrain the space of models without using crude proxies, such as the number of terms or the depth of the expression tree. However, some shape functions may be difficult to interpret, and the number of compositions may significantly increase the cognitive load required to analyze the expression. Although transparent SHAREs guarantee a certain level of transparency, some additional constraints may need to be enforced in practice. We elaborate on our attempt to encourage "nicer" shape functions in Appendix D.

Limiations of the Implementation The current implementation of SHAREs is mainly for illustrative purposes, and thus, it is time-intensive. While this may not be a concern for certain (not time-sensitive) applications, such as scientific discovery, we are confident that further optimizations will enable wider adoption of this novel approach by enhancing the ability to fit SHAREs to even larger and more complex datasets.

Acknowledgements

References

B. S. Aakash, JohnPatrick Connors, and Michael D. Shields. Stress-strain data for aluminum 6061-T651 from 9 lots at 6 temperatures under uniaxial and plane strain tension. *Data in Brief*, 25:104085, August 2019. ISSN 2352-3409. doi: 10.1016/j.dib.2019.104085.

Ashraf Abdul, Christian Von Der Weth, Mohan Kankanhalli, and Brian Y. Lim. COGAM: Measuring and Moderating Cognitive Load in Machine Learning Model Explanations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Honolulu HI USA, April 2020. ACM. ISBN 978-1-4503-6708-0. doi: 10.1145/3313831.3376615.

Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural Additive Models: Interpretable Machine Learning with Neural Nets. In *Advances in Neural Information Processing Systems*, volume 34, pages 4699–4711. Curran Associates, Inc., 2021.

Ahmed M. Alaa and Mihaela van der Schaar. Demystifying Black-box Models with Symbolic Metamodels. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, June 2020. ISSN 1566-2535. doi: 10.1016/j.inffus.2019.12.012.

L. Biggio, T. Bendinelli*, A. Neitz, A. Lucchi, and G. Parascandolo. Neural Symbolic Regression that Scales. In *38th International Conference on Machine Learning*, July 2021.

J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, June 2007. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.0609476104.

Rich Caruana, Steve Lawrence, and C. Giles. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730, Sydney NSW Australia, August 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788613.

Chun-Hao Chang, Rich Caruana, and Anna Goldenberg. NODE-GAM: Neural Generalized Additive Model for Interpretable Deep Learning, March 2022.

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. Neurosymbolic Programming. *Foundations and Trends® in Programming Languages*, 7(3):158–243, December 2021. ISSN 2325-1107, 2325-1131. doi: 10.1561/2500000049.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, August 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785.

Yize Chen, Marco Tulio Angulo, and Yang-Yu Liu. Revealing Complex Ecological Dynamics via Symbolic Regression. *BioEssays*, 41(12):1900069, 2019. ISSN 1521-1878. doi: 10.1002/bies.201900069.

Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick. Control Regularization for Reduced Variance Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1141–1150. PMLR, May 2019.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), February 2016.

Miles Cranmer. PySR: Fast & parallelized symbolic regression in Python/Julia. Zenodo, September 2020.

Stéphane D'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and Francois Charton. Deep symbolic regression for recurrence prediction. In *Proceedings of the 39th International Conference on Machine Learning*, pages 4520–4536. PMLR, June 2022.

Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, March 2023. ISSN 0269-2821, 1573-7462. doi: 10.1007/s10462-023-10448-w.

Trevor Hastie and Robert Tibshirani. Generalized additive models. *Statistical Science*, 1(3):297–318, 1986.

Trevor Hastie and Robert Tibshirani. Generalized additive models for medical research. *Statistical methods in medical research*, 4(3):187–196, 1995.

Pascal Hitzler and Md Kamruzzaman Sarker, editors. *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, December 2021. ISBN 978-1-64368-244-0 978-1-64368-245-7. doi: 10.3233/FAIA342.

Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep Generative Symbolic Regression. *The Eleventh International Conference on Learning Representations*, 2023.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456. PMLR, June 2015.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and Francois Charton. End-to-end Symbolic Regression with Transformers. In *Advances in Neural Information Processing Systems*, October 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, volume 114, pages 953–956. Russian Academy of Sciences, 1957.

Michael Kommenda, Andreas Beham, Michael Affenzeller, and Gabriel Kronberger. Complexity Measures for Multi-objective Symbolic Regression. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015*, Lecture Notes in Computer Science, pages 409–416, Cham, 2015. Springer International Publishing. ISBN 978-3-319-27340-2. doi: 10.1007/978-3-319-27340-2_51.

JohnR. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2), June 1994. ISSN 0960-3174, 1573-1375. doi: 10.1007/BF00175355.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Kay Chen Tan. A Survey on Evolutionary Neural Architecture Search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(2):550–570, February 2023. ISSN 2162-2388. doi: 10.1109/TNNLS.2021.3100554.

Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 150–158, New York, NY, USA, August 2012. Association for Computing Machinery. ISBN 978-1-4503-1462-6. doi: 10.1145/2339530.2339556.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 623–631, New York, NY, USA, August 2013. Association for Computing Machinery. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487579.

Georg S Martius and Christoph Lampert. Extrapolation and learning equations. In *5th International Conference on Learning Representations, ICLR 2017-Workshop Track Proceedings*, 2017.

David Navon. Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, 9(3):353–383, July 1977. ISSN 0010-0285. doi: 10.1016/0010-0285(77)90012-3.

Allen Newell, J. C. Shaw, and Herbert A. Simon. Elements of a theory of human problem solving. *Psychological Review*, 65(3):151–166, 1958. ISSN 0033-295X. doi: 10.1037/h0048495.

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. InterpretML: A Unified Framework for Machine Learning Interpretability, September 2019.

Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. PMLB: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, December 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4.

Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep Symbolic Regression: Recovering Mathematical Expressions From Data via Risk-seeking Policy Gradients. In *ICLR 2021*, 2021.

Filip Radenovic, Abhimanyu Dubey, and Dhruv Mahajan. Neural Basis Models for Interpretability. In *Advances in Neural Information Processing Systems*, October 2022.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 00219991. doi: 10.1016/j.jcp.2018.10.045.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0048-x.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16 (none):1–85, January 2022. ISSN 1935-7516. doi: 10.1214/21-SS133.

Subham Sahoo, Christoph Lampert, and Georg Martius. Learning Equations for Extrapolation and Control. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4442–4450. PMLR, July 2018.

Michael Schmidt and Hod Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324 (5923):81–85, April 2009. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.1165893.

Daniel Servén and Charlie Brummitt. pyGAM: Generalized additive models in python, March 2018.

Herbert A. Simon. The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962. ISSN 0003-049X.

Herbert A. Simon. *The Sciences of the Artificial*. The MIT Press, 1996. ISBN 978-0-262-35474-5. doi: 10.7551/mitpress/12107.001.0001.

Trevor Stephens. Gplearn: Genetic programming in python, with a scikit-learn inspired and compatible api, 2022.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, April 2020. ISSN 2375-2548. doi: 10.1126/sciadv.aay2631.

Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2021.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. HOUDINI: Lifelong Learning as Program Synthesis. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 877–884, Portland Oregon USA, July 2010. ACM. ISBN 978-1-4503-0072-8. doi: 10.1145/1830483.1830643.

Marco Virgolin and Solon P. Pissis. Symbolic Regression is NP-hard. *Transactions on Machine Learning Research*, July 2022. ISSN 2835-8856.

Ekaterina J. Vladislavleva, Guido F. Smits, and Dick den Hertog. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, April 2009. ISSN 1941-0026. doi: 10.1109/TEVC.2008.926486.

Simon Vollert, Martin Atzmueller, and Andreas Theissler. Interpretable Machine Learning: A brief survey from the predictive maintenance perspective. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 01–08, September 2021. doi: 10.1109/ETFA45728.2021.9613467.

Yiqun Wang, Nicholas Wagner, and James M. Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, September 2019. ISSN 2159-6859, 2159-6867. doi: 10.1557/mrc.2019.85.

Checklist

1. For all models and algorithms presented, check if you include:

   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes, in Section 4, Section 5, Appendix A, and Appendix B.

   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes, in Appendix D.

   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries.

2. For any theoretical claim, check if you include:

   (a) Statements of the full set of assumptions of all theoretical results. Yes, in Section 4.

   (b) Complete proofs of all theoretical results. Yes, in Appendix A.

   (c) Clear explanations of any assumptions. Yes, in Section 3 and Section 4.

3. For all figures and tables that present empirical results, check if you include:

   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). All experimental and implementation details are described in Appendix B and Appendix C. Code for all experiments can be found at `https://github.com/krzysztof-kacprzyk/SHAREs`.

   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes, in Appendix B and Appendix C.

   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes, where applicable.

   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes, in Appendix C.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

   (a) Citations of the creator If your work uses existing assets. Yes.

   (b) The license information of the assets, if applicable. Yes, in Appendix C.

   (c) New assets either in the supplemental material or as a URL, if applicable. Not applicable.

   (d) Information about consent from data providers/curators. Not applicable.

   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not applicable.

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

   (a) The full text of instructions given to participants and screenshots. Not Applicable.

   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable.

   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable.

TABLE OF SUPPLEMENTARY MATERIALS

A   THEORETICAL RESULTS

This section provides proof of the properties listed in Proposition 1.

First, let us define *active variables* and a *subtree of a node.*

Definition 2 (Active variables). Consider a SHARE $f : \mathbb{R}^n \to \mathbb{R}$ and its expression tree. The *set of active variables* of the tree (or of f) is the set of variables present in the tree.

For instance, a function $f(x_1, x_2, x_3) = x_1 + x_2$, represented as a tree with 3 nodes: $+$, x_1, x_2, has the set of active variables $\{x_1, x_2\}$ and a set of not active variables $\{x_3\}$.

Definition 3 (Subtree of a node). Consider an expression tree T. For each node A in T, we define the subtree of A as a subtree of T containing A and all its descendants. We call f_A the function represented by the subtree of A.

For the rest of the section, we assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a transparent SHARE (according to Definition 1) and its expression tree is called T

A.1   Variable Nodes

Claim: Each variable node appears at most once in the expression tree. The maximum number of leaves is n.

Proof. Assume for contradiction there are two nodes, A and B, describing the same variable x. Consider the lowest common ancestor of A and B called C. If C was a shape function, then the child of C would be a lower common ancestor of A and B. Thus, C is a binary function with children C_1 and C_2. Without loss of generality, assume that A is in the subtree of C_1. Then B has to be in the subtree of C_2 (otherwise B would have to be in a subtree of C_1 and C_1 would be a lower common ancestor of A and B). Thus, the functions f_{C_1}, f_{C_2} have a non-empty set of active variables (contains at least x). Thus C is a binary operator applied to two functions with an overlapping set of active variables. Thus, f does not satisfy Rule 2, which contradicts f being transparent.

Thus, each variable node appears only once in the expression tree. As there are n variables, there are at most n variable nodes. This is the same as the number of leaves, as variable nodes are the only kinds of leaves. $\square$

A.2   Useful Lemma

In the next proofs, the following lemma will be helpful.

Lemma 1. *Consider a node A that corresponds to a binary operator. Let us call the children of A, A_1, and A_2. If f_{A_1} has a active variables f_{A_2} has b active variables then f_A has $a + b$ active variables. Also, $a, b < a + b$.*

Proof. The set of active variables of f_A is the union of active variables of f_{A_1} and f_{A_2}. As these functions have disjoint sets of active variables (because f is transparent), the number of active variables of f_A is just a sum of the numbers of active variables of f_{A_1} and f_{A_2}. $\square$

A.3   Number of Binary Operators

Claim: The number of binary operators is $d - 1$ where d is the number of active variables of f.

Proof. Let us prove the following more general statement: Consider the node B. If the number of active variables of f_B is k then the number of binary operators in the subtree of B is $k - 1$.

We prove it by strong induction on k.

When $k = 1$ (we cannot have $k = 0$ as we do not have any constants), the subtree of B is either a variable node or a shape function with a node variable as a child. In both cases, the number of binary operators is 0.

Let us assume the statement is true for all $m < k+1$. Assume that the subtree of B has $k+1$ active variables. B is either a shape function or a binary operator. If B is a binary operator, then it has two children. Let us call them B_1 and B_2. Let us denote the number of active variables of f_{B_1} as a and of f_{B_2} as b. From Lemma 1, $k+1 = a+b$ and $a < k + 1$, and $b < k + 1$. From the induction hypothesis, the subtree of B_1 has $a - 1$ binary operators, and the subtree of B_2 has $b - 1$ binary operators. Thus the subtree of B has $(a - 1) + (b - 1) + 1 = a + b - 1 = k$ binary operators. If B is a shape function, then its child is a binary operator, and the same argument follows.

By induction, if the number of active variables of f_B is k then the number of binary operators in the subtree of B is $k - 1$.

Now f has d active variables, so it has $d - 1$ binary operators. □

A.4 Depth of the Expression Tree

Let d be the number of active variables of f. By the previous result, it has $d - 1$ binary operators. That means that on the path from the root to the variable node, there are at most $d - 1$ binary operators. On this path, every pair of consecutive binary operators can be separated by at most one shape function (the same is true for a binary operator and a variable node). Thus the maximum number of nodes on the path from the root to the variable node is a sum of $d - 1$ (number of binary operators), $d - 2$ (number of shape functions between the operators), 1 (shape function between the operator and the variable node), 1 (shape function as a root), 1 (the variable node itself). This gives a total of $(d - 1) + (d - 2) + 1 + 1 + 1 = 2d$. Thus, the maximum depth of the tree is $2d$. As $d \leq n$, we get that the maximum depth of the tree is $2n$.

A.5 Size of the Expression Tree

Claim: The number of nodes in a tree is at most $4n - 2$.

Proof. Let us prove the following more general statement: Consider a node A. If the number of active variables of f_A is k then the maximum number of nodes in the subtree of A is $4k - 2$ if A is a shape function and $4k - 3$ otherwise.

Let us prove it by strong induction on k.

Consider $k = 1$. The subtree of A is either a variable node or a shape function with a variable node as a child. The number of nodes is either 1 if it is a variable node or 2 if it is a shape function. As $4 \times 1 - 2 = 2$ and $4 \times 1 - 3 = 1$, the base case is satisfied.

Let us assume the statement is true for all $m < k + 1$.

Consider a node A whose subtree has $k + 1$ active variables. If A is a binary operator, then it has two children, A_1 and A_2. Their subtrees have respectively a and b active variables. From Lemma 1, $a + b = k + 1$. By the induction hypothesis, the maximum number of nodes in the subtree of A_1 is $4a - 2$ and $4b - 2$ in the subtree of A_2. Thus, the maximum number of nodes in the subtree of A is $(4a-2)+(4b-2)+1 = 4(a+b)-3 = 4(k+1)-3$. This proves one part of the claim. If A is a shape function, then its child is a binary operator with $k + 1$ active variables. But we have just proved that the subtree of this operator has at most $4(k + 1) - 3$ nodes. That means that the maximum number of nodes in the subtree of A is $4(k + 1) - 3 + 1 = 4(k + 1) - 2$ as required.

The claim is true by induction. Now, we want to show that such a tree always exists. Consider a binary operator node A_1 whose subtree has k active variables $\{x_1, \ldots, x_k\}$. Let its children be two shape functions B_1 and C_1. Let the child of C_1 be a variable node corresponding to x_1. Let the child of B_1 be a binary operator A_2. We repeat the process. In general, binary operator node A_i has two children, B_i and C_i. The child of C_i is a variable node corresponding to x_i and the child of B_i is the binary operator A_{i+1}. We can repeat the process until $i = k - 1$. At this point, the child of B_{k-1} needs to be a variable node corresponding to x_k. Overall, we have

$k-1$ binary operators $A_1, \ldots, A_{k-1}$. $k-1$ shape functions $B_1, \ldots, B_{k-1}$, $k-1$ shape functions $C_1, \ldots, C_{k-1}$, and k variable nodes. Thus the total number of nodes is $3(k-1) + k = 4k - 3$. If the first node is a shape function, then its child is the binary operator node A_1 and the total number of nodes is $4k - 2$.

As the number of active variables in the whole tree is less than n, then the maximum number of nodes is $4n - 2$. $\qquad\square$

B   IMPLEMENTATION

We implement SHAREs using nested optimization. The outer loop is a modified genetic programming algorithm (based on gplearn (Stephens, 2022) that is used for symbolic regression) that finds a symbolic expression with placeholders for the shape functions and the inner loop optimizes the shape functions. We implement the shape functions as neural networks and optimize the model using a gradient descent algorithm.

B.1   Modifications to the Genetic Algorithm

Gplearn is a symbolic regression algorithm that represents equations as expression trees and uses genetic programming to alter the equations (programs) from one generation to the next based on their fitness score. We modify this algorithm so that the found expressions contain placeholders for the shape function. To compute the fitness of an expression, the whole equation is fitted to the data, and the shape functions are optimized using gradient descent. To guarantee that all equations are transparent (i.e., they satisfy Definition 1), we change the way the initial population is created and modify some of the rules by which the equations evolve. We describe the details of the modifications below.

Initial Population We disable the use of constants and allow only binary operations in $\mathcal{B}$ and shape functions. We grow the expression trees at random, starting from the root. The next node is chosen randomly and constrained such that: a) if the parent node is a shape function, then the child cannot be a shape function, and b) no variable can appear twice in the tree. By the Corollary 1, the second condition is equivalent to satisfying Rule 2.

Crossover We call the variables present in a tree *active variables*. During crossover, we select a random subtree from the program to be replaced. We take the union between the active variables in the subtree and the variables that are not active in the whole program. A donor has a subtree selected at random such that its set of active variables is contained in the previous set. This subtree is inserted into the original parent to form an offspring. This guarantees that no variable appears twice in the offspring.

Subtree Mutation We perform the same procedure as in crossover but instead of taking a subtree from a donor, we create a new program with variables from the allowed set.

Point Mutation This procedure selects a node and replaces it for a different one. A binary operation is replaced by a different binary operation. Shape functions are not replaced. All variables that are supposed to be replaced are collected in a set. This set is enlarged by the variables that are not active. For each mutated node the variable is drawn from this set without replacement.

Reproduction and hoist mutation has not been altered. For more details about the genetic programming part of the algorithm, please see the official documentation of gplearn.

Binary Operations $\mathcal{B}$ We choose the set of binary operations to be $\mathcal{B} = \{+, \times, \div\}$ (we remove "$-$" to remove redundancy and reduce the search space).

B.2   Optimization of the Shape Functions

Shape Functions $\mathcal{S}$ We choose the set of shape functions $\mathcal{S}$ to be a family of Neural Networks with 5 hidden layers, each with a width of 10. Each layer, excluding the last one, is followed by an ELU (Clevert et al., 2016) activation function. We apply batch normalization (Ioffe and Szegedy, 2015) before the input layer.

Dataset vs. Batch Normalization We do not perform any normalization on the whole dataset before training. This is driven by the fact that we want to use the form of the equation for analysis, debugging or gaining insights. Dataset normalization makes the feature less interpretable by, de facto, changing the units in which they are measured. Moreover, such normalization might make certain invariances more difficult to detect. Translational or scale invariances are present in many physical systems and, in fact, have been used to discover closed-form expressions from data (Udrescu and Tegmark, 2020). Consider equation $(x_2 - x_1)^2$. The value of the expression does not depend directly on the values of x_1 and x_2 but rather on their difference $x_2 - x_1$. Detecting this relationship is important for both creating equations with interpretable terms and for pruning the search space. As we tend to use a consistent and familiar set of units, we want to capitalize on that as much as we can. However, features on different scales make neural networks (and other machine learning algorithms) notoriously difficult to train. That is why we perform batch normalization before passing the data to a shape function. That allows to perform a series of binary operations in the original units before a shape function is applied. This is what happens in the example in Section 5, where given the energy and the mass of the substance, their ratio (energy per 1 gram) is discovered to be a more meaningful feature.

B.3   Pseudocode and a Diagram

Block Diagram The training procedure for SHAREs implemented with symbolic regression and neural networks is depicted in Figure 8.
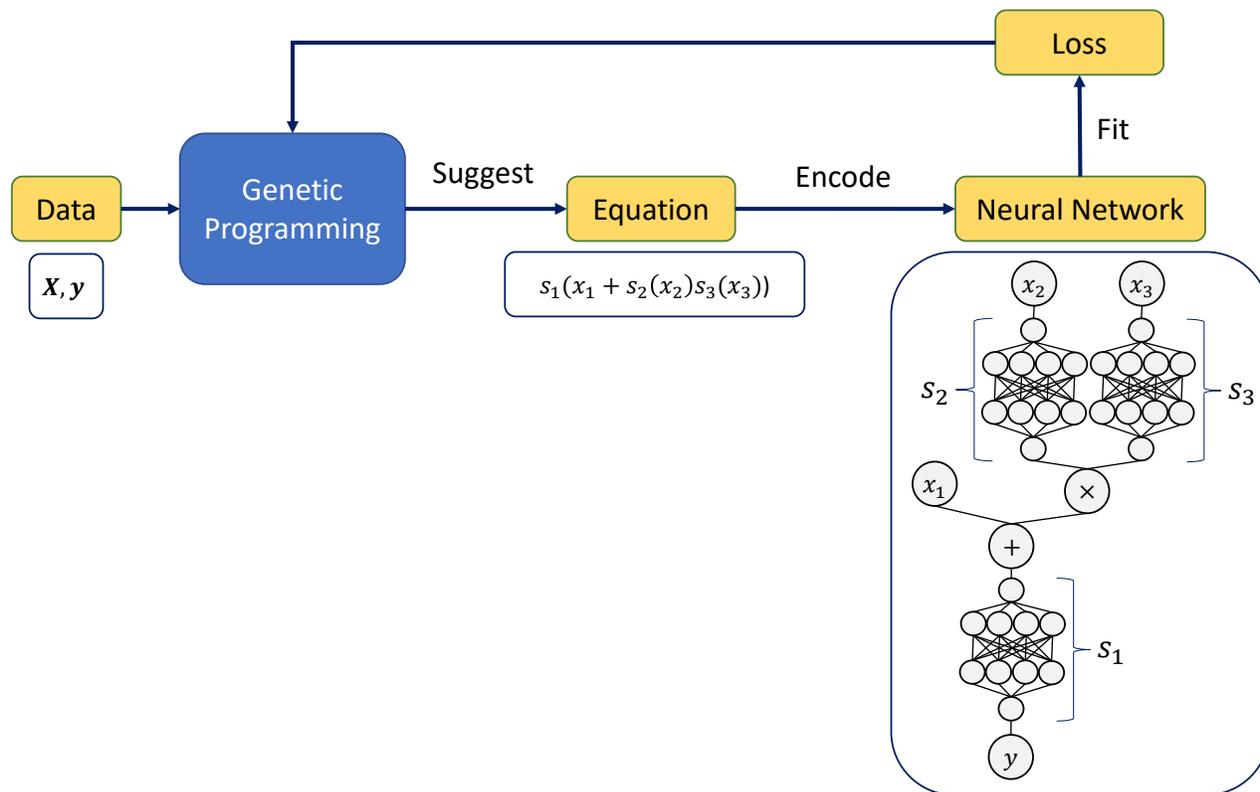
Figure 8: This figure shows a block diagram depicting our implementation of SHAREs

Pseudocode The pseudocode for SHAREs implemented with symbolic regression and neural networks is described in Algorithm 1.

Algorithm 1 SHARE implemented using symbolic regression and neural networks

---

Input: Data $\boldsymbol{X}$, $\boldsymbol{y}$
Input: Symbolic regression optimization algorithm $\mathcal{O}_{\text{symbolic}}$
Input: Gradient-based optimization algorithm $\mathcal{O}_{\text{gradient}}$
Output: SHARE
  procedure Loss(f_e)
    Encode expression f_e as a neural network f
    $f \leftarrow \mathcal{O}_{\text{gradient}} \left(\|\boldsymbol{y} - f(\boldsymbol{X})\|_2^2 \right)$
    return $\|\boldsymbol{y} - f(\boldsymbol{X})\|_2^2$
  end procedure
  $f_e = \mathcal{O}_{\text{symbolic}}(\text{Loss})$
  return f_e

C EXPERIMENTS

C.1 Hyperparameters

gplearn Gplearn hyperparameters used for experiments are presented in Table 5.

Table 5: Gplearn hyperparameters used in the experiments.

Hyperparameter	Value
Population size	500 (Section 5) or 100 (Appendix C.4)
Generations	10
Tournament size	10
Function set	$+, \times, \div, \text{ shape}$
Constant range	None
p_crossover	0.4
p_subtree_mutation	0.2
p_point_mutation	0.2
p_hoist_mutation	0.05
p_point_replace	0.2
Parsimony coefficient	0.0

Optimization of Shape Functions Hyperparameters used for optimizing the shape function are presented in Table 6.

Table 6: Hyperparameters used in shape function optimization.

Hyperparameter	Value
Algorithm	Adam (Kingma and Ba, 2017)
Maximum num. of epochs	1000 (Section 5) or 200 (Appendix C.4)
Learning rate	Tuned automatically for each equation
Weight decay	0.0001

PySR PySR hyperparameters used in the experiments are presented in Table 7.

EBM EBM hyperparameters used in the experiments in Section 2.2 are presented in Table 8. Hyperparameter ranges used for tuning in experiments in Appendix C.4 are shown in Table 9.

XGBoost XGBoost hyperparameter used in the experiments in Section 2.2 are presented in Table 10. Hyperparameter ranges used in Appendix C.4 are presented in Table 11.

Table 7: PySR hyperparameters.

Hyperparameter	Value
Binary operations	$+, -, \times, \div$
Unary operators	$\log, \exp, \cos$
maxsize	25 (Table 1) or 40 (Table 3)
populations	15 (Table 1) or 30 (Table 3)
niterations	400
population_size	33 (Table 1) or 50 (Table 3)

Table 8: EBM hyperparameters (Section 2.2 and Section 5).

Hyperparameter	Value
max_bins	256
max_interaction_bins	32
binning	quantile
interactions	0 or 3
outer_bags	8
inner_bags	0
learning_rate	0.01
validation_size	0.15
early_stopping_rounds	50
early_stopping_tolerance	0.0001
max_rounds	5000
min_samples_leaf	2
max_leaves	3

Table 9: EBM hyperparameter ranges for tuning (Appendix C.4).

Hyperparameter	Value
max_bins	Integer from $[3, 256]$
outer_bags	Integer from $[4, 16]$
inner_bags	Integer from $[0, 8]$
learning_rate	Float (log) from $[1e-3, 1e-1]$
validation_size	Float from $[0.1, 0.3]$
min_samples_leaf	Integer from $[1, 10]$
max_leaves	Integer from $[2, 10]$

PyGAM PyGAM hyperparameter ranges used in experiments in Appendix C.4 are shown in Table 12.

C.2   Data Generation

Risk Scores Dataset   Hastie and Tibshirani (1995) describes a process of applying GAMs to a dataset of patients taking part in a clinical trial for the treatment of node-positive breast cancer. In the paper, three shape functions that resulted from this fitting are presented (Figure 1). To generate the risk scores data we use for experiments in Section 5, we use these plots to create similar-looking functions using BSplines. We sample uniformly each of the covariates from their corresponding ranges, i.e., $x_{nodes} \in (0, 50)$, $x_{age} \in (45, 70)$, and $x_{bmi} \in (17, 45)$. We then apply the created shape functions to these covariates and add their values together. We create 200 samples. Half of them is used for training and the other half for validation. The code for creating this dataset is included in the paper repository.

Table 10: XGBoost hyperparameters (Section 2.2)

Hyperparameter	Value
subsample	1
colsample_bytree	1
colsample_bylevel	1
gamma	0
booster	gbtree
learning rate	0.3
max_depth	6
n_estimators	100
reg_alpha	0
reg_lambda	1
scale_pos_weights	1

Table 11: XGBoost hyperparameter ranges for tuning (Appendix C.4).

Hyperparameter	Value
subsample	Float from $[0.2, 1.0]$
colsample_bytree	Float from $[0.2, 1.0]$
gamma	Float (log) from $[1e-3, 1e1]$
learning rate	Float (log) from $[1e-3, 1.0]$
max_depth	Integer from $[1, 10]$
n_estimators	Integer (log) from $[10, 1000]$
reg_alpha	Float (log) from $[1e-3, 1e1]$
reg_lambda	Float (log) from $[1e-3, 1e1]$

Table 12: PyGAM hyperparameter ranges for tuning (Appendix C.4).

Hyperparameter	Value
lam	Float (log) from $[1e-3, 1e1]$
max_iter	Integer (log) from $[10, 100]$
n_splines	Integer from $[1, 30]$

Temperature Dataset Data used in temperature experiments in Section 5 was generated by simulating the temperature of water based on the laws of physics and constants shown in Table 4. m was uniformly sampled from $(1, 4)$ and t_0 was sampled uniformly from $(-100, 0)$. The energy E was calculated by first sampling energy per mass uniformly from $(1, 800)$ and then multiplying it by the mass m. The temperature (label) range (-100 to 250) was chosen to capture significant non-linear phenomena. Uniform energy sampling was avoided due to its tendency to either under-represent temperatures over 100 or, if the energy range is increased, result in excessively high temperatures (more than few thousands) dominating the training process. This is due to the high heat of vaporization. We draw 2000 samples. Half of them is used for training and the other half for validation. The code for creating this dataset is included in the paper repository.

C.3   Additional Results

GA^{2}M Fitted to the Temperature Dataset We present the shape functions of GA^{2}M fitted to the temperature dataset in Section 5.

Equations From PySR Fitted to the Temperature Dataset. We present the equations found by PySR when fitted to the temperature dataset in Section 5. These equations did not fit into the table with the results.
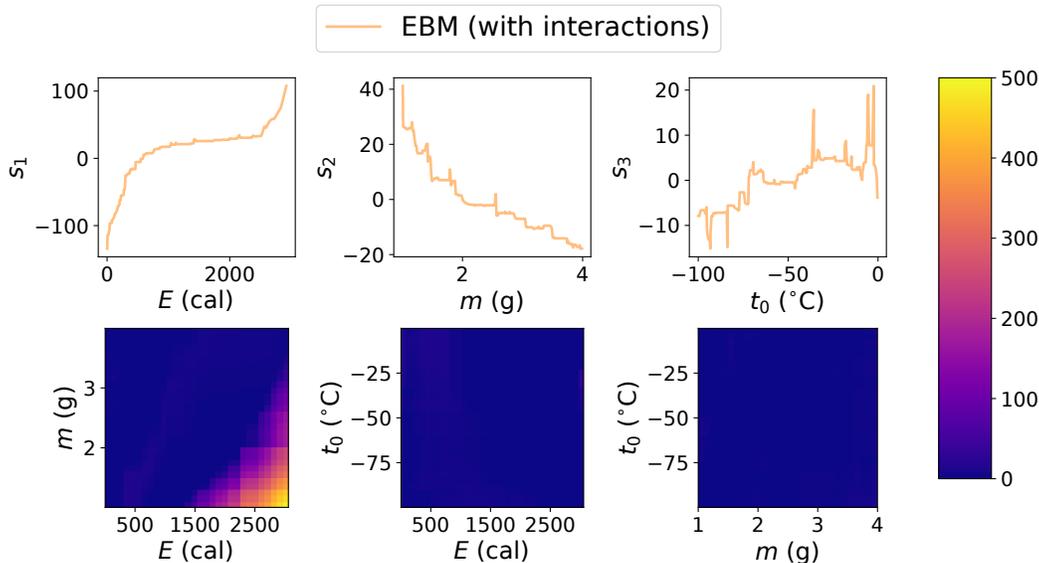
Figure 9: Shape functions from the GA^{2}M fitted to the temperature dataset

$$y = 74.0 \cos \left(\log \left(\frac{0.739E}{m} + 19.1 \right) \right) + 39.1 + \frac{t_0}{E} \tag{9}$$

$$y = \frac{t_0}{\log (E)} + 65.0 \cos \left(\log \left(\frac{E}{m} + 41.5 \cos \left(\frac{0.0275E}{m} \right) \right) \right) + 50.9 \tag{10}$$

$$y = -1.63 e^{e^{\cos \left(\frac{0.0101E}{m} \right)}} + 58.6 \cos \left(\log \left(\frac{0.653E}{m} + 27.1 \cos \left(\frac{0.0261E}{m} \right) \right) \right) + 67.6 \tag{11}$$

$$y = \frac{x_2}{\log (x_0)} - 1.72 e^{e^{\cos \left(\frac{0.0103E}{m} \right)}} + 56.3 \cos \left(\log \left(\frac{0.582E}{m} + 31.7 \cos \left(\frac{0.0214E}{m} + 0.925 \right) \right) \right) + 80.1 \tag{12}$$

C.4 Additional Experiments

We performed additional experiments on a few small classification datasets from the PMLB database (Olson et al., 2017). The results are shown in Table 13

Table 13: AUC scores on small PMLB classification datasets. The higher the score, the better. Linear denotes standard logistic regression. GAM-S is a type of GAM that uses smooth splines as shape functions (as implemented in `pygam` (Servén and Brummitt, 2018)). EBM-1 and EBM-2 are Explainable Boosting Machines (without and with pairwise interactions) as implemented in `InterpretML` package (Nori et al., 2019).

	banana	cancer	breast	diabetes
Linear	0.555 (0.000)	0.595 (0.000)	0.997 (0.000)	0.850 (0.000)
GAM-S	0.804 (0.000)	0.650 (0.000)	0.992 (0.000)	0.862 (0.000)
EBM-1	0.800 (0.001)	0.645 (0.014)	0.995 (0.001)	0.857 (0.004)
EBM-2	0.957 (0.001)	0.651 (0.007)	0.997 (0.001)	0.847 (0.003)
XGBoost	0.800 (0.002)	0.652 (0.028)	0.995 (0.002)	0.847 (0.002)
SHARE	0.915 (0.000)	0.668 (0.032)	0.998 (0.001)	0.846 (0.010)

We performed additional experiments on the temperature dataset, assessing performance under varying noise levels. As illustrated in Figure 10, SHAREs demonstrate comparable robustness to noise as XGBoost (note, label range is (-100, 250)). Furthermore, minimal variance in results across different initialization (indicated by error bars) underscores SHAREs' consistency.
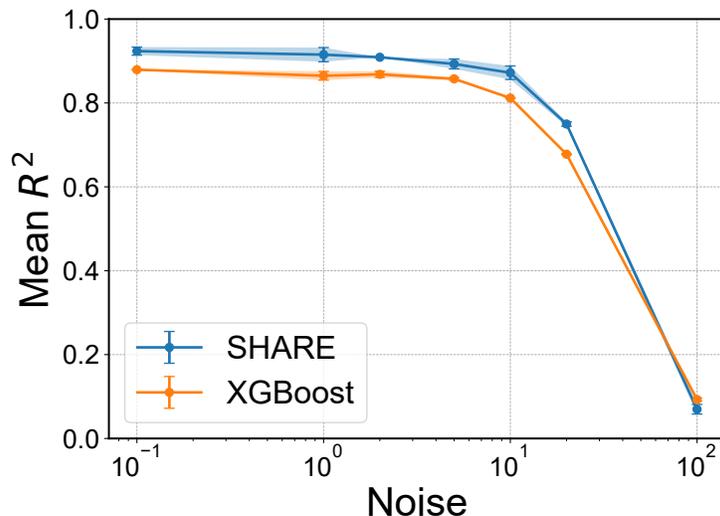
Figure 10: Performance vs. noise on the temperature dataset

C.5 Computation Time

The experiments were performed on 12th Gen Intel(R) Core(TM) i7-12700H with 64 GB of RAM. The average times for experiments in Appendix C.4 are shown in Table 14. The total time for all experiments is around 15 hours.

Table 14: Average computation times (in seconds) for each of the methods and datasets in Appendix C.4.

	banana	cancer	breast	diabetes
Linear	0.01	0.00	0.00	0.01
GAM-S	0.09	0.47	0.17	0.09
EBM-1	0.57	0.04	0.08	0.08
EBM-2	6.66	0.20	0.53	1.17
XGBoost	0.09	0.01	0.02	0.05
SHARE	874.04	2321.60	7535.68	6133.11

C.6 Software Used

We use PySR (Cranmer, 2020) to run symbolic regression experiments.

We use the implementation of EBM (Lou et al., 2012, 2013) available in the InterpretML package (Nori et al., 2019).

We use PMLB (Olson et al., 2017) package to access the Feynman Symbolic Regression Dataset.

C.7 Licenses

The licenses of the software used in this work are presented in Table 15

Table 15: Software used and their licenses

Software	License
gplearn	BSD 3-Clause "New" or "Revised" License
scikit-learn	BSD 3-Clause "New" or "Revised" License
numpy	liberal BSD license
pandas	BSD 3-Clause "New" or "Revised" License
scipy	liberal BSD license
python	Zero-Clause BSD license
PySR	Apache License 2.0
interpret	MIT License
pmlb	MIT License
pytorch	BSD-3
pytorch lightning	Apache License 2.0
tensorboard	Apache License 2.0
py-xgboost	Apache License 2.0
pyGAM	Apache License 2.0

D   DISCUSSION

D.1   Equivalent Solutions

As SHAREs are defined by their symbolic representation, it is possible that there are two different symbolic expressions that describe the same equation (especially as shape functions are flexible). We address this problem in three ways that tackle three types of equivalence relations:

1. In our implementation, we restrict the set of binary operations to $\{+, \times, \div\}$ (Table 5). As subtraction ("$-$") is not included, we do not get the equivalence $s_1(x_1) + s_2(x_2) = s_1(x_1) - s_2'(x_2)$ by $s_2' = -s_2$.
2. A common way two mathematical expressions can be equivalent is through the *distributive property*, i.e., $x_1 \times (x_2 + x_3) = x_1 \times x_2 + x_1 \times x_3$. However, thanks to our definition of transparency, the second expression will never appear in our search space (because the binary operators need to be disjoint).
3. We do not allow constants in transparent SHAREs to prevent equivalence of the type: $s(x) = s'(x \times a)$ for any $a \in \mathbb{R} \setminus \{0\}$ and $s'(x) = s(\frac{x}{a})$

Another type of equivalence relation can arise from the use of exponential and logarithmic functions. For instance, $s_0(s_1(x_1) + s_2(x_2))$ can be represented as $s_0'(s_1'(x_1) \times s_2'(x_2))$ by taking $s_1' = e^{s_1}, s_2' = e^{s_2}, s_0'(x) = s_0(\log(x))$. However this other form of the expression require more shape functions. Thus, as they have the same predictive power, the user can choose the one with a smaller number of shape functions for better understanding.

D.2   Limitations of the Current Implementation

The current implementation of SHAREs is time-intensive and thus does not scale to bigger datasets—thus, it is not a main contribution of our paper. We hope that future work will address the limitations of our implementation and will enhance the ability to fit SHAREs to even larger and more complex datasets.

The main bottleneck comes from nested optimization and the necessity of fitting a separate neural network for every equation. Nevertheless, we want to highlight a few things we have done to make this problem more tractable:

1. The constants are *not* optimized by random mutations but implicitly by fitting the shape functions using gradient descent.
2. By considering only transparent SHAREs, we efficiently reduce the search space of expressions. By Proposition 1, the size of a SHARE is bounded by $4n - 2$, linear in the number of variables.
3. During training, we cache the scores for found expressions so that they can be retrieved if they appear once again during the evolution.

D.3   Raw Variable Combinations and Unit Transformations

Variables in the dataset are often expressed in certain units. These units often provide a lot of information and are frequently used in SR algorithms. Either explicitly (Udrescu and Tegmark, 2020) or implicitly by assuming that certain arithmetic operations make sense. For instance, adding two variables makes little sense if they are not measured in the same units. Of course, units may easily be changed by an affine transformation, but such transformations increase the length of the equation. As many SR algorithms penalize based on the length of the expression, *the change of units changes the score of the expression*. On the other hand, the datasets often combine observations of very different phenomena that might be measured in wildly different units.

So, we want to use the information about units when possible, but we do not want to depend on it. This was one of the motivations behind SHAREs. They allow the use of binary operations on the raw variables, capitalizing on the units in which they were described, but also allow the first pass of a variable through a shape function that can transform the variable. In certain cases (such as an affine transformation), this corresponds to a change of units.

SHAREs, unlike GAMs, can accommodate combinations of raw variables (e.g., $\frac{E}{m}$, $x_1 - x_2$), leading to potential meaningful constructs like energy per unit mass or distance between points. Contrasting with symbolic regression, SHAREs are less unit-sensitive, as variations in units (e.g., °C, °F, K) get absorbed within the shape functions and do not lead to longer expressions.

In the current implementation, these relationships are learned directly from data. Future research direction may include ways of explicitly providing information about units.

D.4   Related works

Complexity Metrics Used in Symbolic Regression   Most of the metrics used in symbolic regression are based on the "size" of the equation. That includes the number of terms (Stephens, 2022), the depth of a tree (Cranmer, 2020; Petersen et al., 2021), and the description length (Udrescu et al., 2021). Pretrained methods often control the complexity of the generated equations by constraining the training set using the above methods (Biggio et al., 2021). Methods that directly represent the equation as a neural network (with modified activation functions) employ sparsity in the network weights (Sahoo et al., 2018). Although these metrics are often correlated with the difficulty of understanding a particular equation, size does not always reflect the equation's complexity as it disregards its semantics. Some approaches try to address this issue. Vladislavleva et al. (2009) introduces a metric based on "order of nonlinearity" with the assumption that nonlinearity measures the complexity of the function. Although simpler models tend to be more linear, and nonlinearity may be important for generalization properties, it is not clear how it aids in model understanding. Similarly, Vanneschi et al. (2010) uses curvature as an inspiration for their metric. Although curvature may be well-suited for characterizing bloat and overfitting, it does not directly relate to how the model is understood. Kommenda et al. (2015) introduces a metric that is supposed to reflect the difficulty in understanding an equation. However, the exact rules chosen to calculate the complexity seem arbitrary. For instance, it is not clear why applying some well-known functions (such as sin, log) increases the complexity in a widely different manner than squaring or taking a square root. We also dispute the motivating example that demonstrates that $e^{\sin \sqrt{x}}$ is nearly 4000 times more complex than $7x^2 + 3x + 5$. It is not clear under what assumptions such a result would be intuitive.

NeuroSymbolic AI   As our implementation of SHAREs contains both symbolic and neural components, it can be classified as an example of Neurosymbolic AI (Hitzler and Sarker, 2021; d'Avila Garcez and Lamb, 2023) that tries to combine symbolic/reasoning elements with neural networks. Neurosymbolic AI encompasses Neurosymbolic Learning Algorithms and Neurosymbolic Representations (Chaudhuri et al., 2021). Examples of the former may include recently proposed algorithms for symbolic regression that utilize large pre-trained transformers (Biggio et al., 2021; D'Ascoli et al., 2022; Kamienny et al., 2022) or where a symbolic loss is used to regularize a deep learning model as in Physic Informed Neural Networks (PINNs) (Raissi et al., 2019). We note, however, that PINNs are inherently a black box model where a physical constraint is used as an inductive bias during training. This significantly differs from our approach, where a human can trace the exact computation steps for the prediction. Thus SHAREs are an example of a neurosymbolic representation, where the model itself is a fusion of symbolic language and neural networks. Examples of such models have also been used in other areas (Valkov et al., 2018; Cheng et al., 2019).

D.5   Meaning of the Word Transparent

In our paper, we have used a widely accepted term: *transparent* (Barredo Arrieta et al., 2020). However, other terms could also be used. That includes: *inherently interpretable* (Rudin, 2019), *intrinsically interpretable* (Vollert et al., 2021), *intelligible* (Lou et al., 2012, 2013), or *white boxes* (Nori et al., 2019). For this paper, we assume all of these terms refer to the same class of models.

D.6   Complexity of Univariate Functions.

Rule 1 in Section 4 is based on the assumption that any univariate function can be understood by plotting it. This assumption is tacitly made in many works on GAMs that introduce algorithms producing sometimes very complicated shape functions (Lou et al., 2012; Caruana et al., 2015; Agarwal et al., 2021). However, in certain scenarios this assumption is too strong and some recent works introduce GAMs that take into account the complexity of shape functions (Abdul et al., 2020). Rule 1 can be modified to include these stronger assumptions. In our implementation, we use neural networks, which are known for their expressiveness, and we took a few design choices that made them simple in practice. We believe their simplicity is a result of

- weight decay to promote smaller weights,
- choosing a smooth ELU activation function instead of ReLU or ExU, (Agarwal et al., 2021) that encourage more jagged functions,
- employing early stopping when training the neural networks to prevent over-fitting,
- optimizing using backprop - models learned that way were shown to be biased toward smooth solutions (Caruana et al., 2000).

Our formulation of SHAREs allows for different kinds of shape functions, and thus, there is a way to enforce simplicity by using splines instead of neural networks. This assumes that our definition of simplicity concerns the smoothness and the number of inflection points. By choosing the number of knots, we can choose the level of simplicity we desire.

D.7   Rules: Practical Example.

We propose a practical example when Rule 1 and Rule 2 are satisfied. Let us assume that our definition of transparency concerns (maybe, among other things) understanding the set of possible values the expression outputs given a particular set of inputs. We characterize the set of inputs by specifying an interval for every feature. That is, we are interested in the range of values of $f(x_1, \ldots, x_n)$ where $x_1 \in [a_1, b_1], \ldots x_n \in [a_n, b_n]$. Let us fix the input intervals. Let us assume that an expression $f(x_1, \ldots, x_n)$ is transparent if we can easily find an interval $[c, d]$ that is an image of this function for the specified inputs. Clearly, x_i is transparent. But so is $f(x_i)$ for any univariate f as long as we are able to characterize the extrema of f at interval $[a_i, b_i]$. Let us assume that a given f is transparent, i.e., we know its image is an interval $[c, d]$. Then, we can compute the image of $s \circ f$ (as long as we are able to characterize the extrema of s at interval $[c, d]$). Thus, we can see how Rule 1 conforms to our notion of transparency. As Rule 2 requires the functions to be transparent and have disjoint sets of arguments, we can easily calculate the range of the whole model by applying the interval arithmetic. The example above demonstrates that our rules have practical application for a broad set of shape functions—the only requirements are being continuous and having easily identifiable extrema at given intervals (which can be found by plotting the function).

D.8   Computational Complexity of the Implementation

Symbolic regression algorithms are known to be computationally intensive, and the problem itself is known to be NP-hard (Virgolin and Pissis, 2022). As our model class also requires discrete and continuous optimization (for the structure and the shape functions, respectively), the task is challenging. We performed several actions (described in Appendix D.2) to speed up the optimization. The algorithm consists of two loops. The outer loop searches through the space of expression trees (where the shape functions are represented as placeholders), and the inner loop optimizes the shape function using gradient descent. This procedure can be phrased as a neural architecture search with the use of genetic programming (Liu et al., 2023). In fact, certain implementations of symbolic regression also perform bi-level optimization. For instance, in the algorithm developed by Cranmer (2020) (PySR), the search is performed over the expression trees, and then the constants are optimized for each

expression. The main overhead in our implementation comes from having more parameters to optimize in the inner loop as we use univariate neural networks. However, that problem can be tackled by choosing flexible shape functions with smaller numbers of parameters.

D.9 Expressivity of Transparent SHAREs

Transparent SHAREs are (by definition) more expressive than GAMs as their search space is bigger. As noted, the Kolmogorov-Arnold theorem (Kolmogorov, 1957) also shows that univariate functions (in theory) should be sufficient to represent any continuous function. In addition, we argue that the assumptions on the structural form of the equation (Rule 2) do not significantly reduce SHARE's ability to model practical complex functions. To demonstrate it, we turn to the current state-of-the-art models we use to model the world - mathematical equations. 86 out of 100 equations from the Feynman Symbolic Regression database (Udrescu and Tegmark, 2020) satisfy Rule 2. We believe that if Transparent SHAREs are expressive enough to fit the majority of equations used in physics, then they are likely to be sufficient to model other real-world phenomena. We also demonstrate it in the additional experiment we performed on real-world datasets (see Appendix C).

D.10 Using Transparency Rules in the Standard Symbolic Regression

By definition (see Remark 1), closed-form expressions can be considered SHAREs if we choose the set to be a finite set of some well-known function. The second rule, thus, can be applied to standard SR. Although the current symbolic regression algorithms are often penalized on the length of the program (Stephens, 2022) or similar length-based criteria (Udrescu et al., 2021), we are not convinced this is the best way to elicit transparency (see Appendix D.4). As we mentioned in Section 4, 86% of equations from the Feynman Symbolic Regression Database (Udrescu et al., 2021) satisfies Rule 2, so the trade-off between performance and interpretability may be worth it.

We note, however, that applying other properties from Definition 1 to SR (shape functions cannot be recursively composed, no numeric constants) is not a good idea because the set of shape functions is not expressive enough (it is fixed and not learned from the data) - it does not satisfy the first criterion.

D.11 Model Selection

When it comes to model selection among interpretable models, the issue is how to balance the trade-off between interpretability and accuracy. Thanks to our definition of transparency, all found models have some guaranteed interpretability (this is not true about the equations found by symbolic regression). However, we recognize that even among transparent SHAREs, some of the models might take less time to understand than others - based on how many shape functions are involved. We do not want to impose a specific strategy on how a user should decide which equation to choose. Instead, as has been done in SR literature (Udrescu et al., 2021), we provide a Pareto-frontier of the found equations. Each of the expressions in the table is Pareto-optimal, i.e., there is no other function that is both simpler (in terms of the number of shape functions) and more accurate.

In certain settings where a more automatic model selection is required, techniques from symbolic regression literature can be used to deterministically choose the "best" equation given the Pareto-frontier. This might involve a simple strategy based on a linear penalty due to the number of shape functions (Stephens, 2022) or more complicated methods based on the accuracy of the most accurate model and the rate of change of the loss as we move along the Pareto-frontier (Cranmer, 2020).

D.12 Scalability of Rule-Based Transparency

Maintaining interpretability as the number of variables increases is one of the main challenges of all intrinsically interpretable methods. As the human brain has a limited capacity for reasoning about more than a few concepts simultaneously, we decided to define transparency so that we can reason about different parts of the model independently. With enough time, we can build intuition and understanding from the ground up, each time combining the knowledge about two smaller models into a bigger one. That is why we believe that our notion of transparency scales well as the number of variables increases. That means SHAREs remain interpretable even in higher-dimensional settings. The traditional size-based notion of transparency employed in symbolic regression does not have this property. As the number of variables increases, the size of the expression has to increase to

accommodate them. Without any constraints similar to Rule 2, we either have to accept that SR cannot be used if the number of dimensions exceeds a certain (quite low) threshold, or we have to accept that equations remain interpretable even as they become very large.