Controllable Neural Symbolic Regression

Tommaso Bendinelli [*1] **Luca Biggio** [*1 2] **Pierre-Alexandre Kamienny** [3 4]

Abstract

In symbolic regression, the goal is to find an analytical expression that accurately fits experimental data with the minimal use of mathematical symbols such as operators, variables, and constants. However, the combinatorial space of possible expressions can make it challenging for traditional evolutionary algorithms to find the correct expression in a reasonable amount of time. To address this issue, Neural Symbolic Regression (NSR) algorithms have been developed that can quickly identify patterns in the data and generate analytical expressions. However, these methods, in their current form, lack the capability to incorporate user-defined prior knowledge, which is often required in natural sciences and engineering fields. To overcome this limitation, we propose a novel neural symbolic regression method, named Neural Symbolic Regression with Hypothesis (NSRwH) that enables the explicit incorporation of assumptions about the expected structure of the ground-truth expression into the prediction process. Our experiments demonstrate that the proposed conditioned deep learning model outperforms its unconditioned counterparts in terms of accuracy while also providing control over the predicted expression structure.

1. Introduction

Symbolic Regression (SR) is a method that searches over the space of analytical expressions $\mathcal{E}$ to find the best fit for experimental data by balancing the need to minimize expression complexity with maximizing accuracy. Unlike over-parametrized methods such as decision trees and neural networks, SR produces human-readable expressions that can provide valuable insights in fields such as material science (Wang et al., 2019; Kabliman et al., 2021; Ma et al., 2022) and fundamental physics (Schmidt and Lipson, 2009; Vaddireddy et al., 2020; Sun et al., 2022; Cranmer et al., 2020; Hernandez et al., 2019; Udrescu and Tegmark, 2020). The goal of SR is to gain a deeper understanding of the underlying mechanisms of physical systems rather than trying to fit the data exactly, which can be affected by measurement errors. SR accomplishes this by converting input numerical data into compact and low-complexity representations in the form of symbolic mathematical expressions.

Researchers in natural sciences frequently rely on prior knowledge and analogies to comprehend novel systems and predict their behavior. When studying specific physical phenomena, scientists might anticipate particular constants or symmetries to appear in the mathematical laws describing the data. For instance, in astrophysics, the gravitational constant has a significant impact on determining the scale of interactions between celestial bodies, while in fluid dynamics, the Reynolds number denotes the relative significance of inertial and viscous forces. Thus, it is crucial to prioritize expressions that contain such constants while employing symbolic regression techniques, as they conform better to the physics laws governing the data. Access to a part of the underlying ground-truth system equation is also a common assumption made in the system identification literature where the physical laws are known up to a few parameters (Brunton et al., 2016; Kaheman et al., 2020). In our work, we will refer to the assumptions made by the SR practitioner about the underlying symbolic expression as *hypotheses*. These hypotheses may be incomplete or partially incorrect and can be used in any form to restrict the search space. If a hypothesis is true, we will name it *privileged information*.

Related work and background

Genetic Programming. Searching for a satisfactory analytical expression is a hard optimization problem, traditionally tackled using genetic programming (GP) algorithms. These methods work by i) defining a class of programs, represented in SR as tree structures where nodes are unary (e.g. $\texttt{cos}, \texttt{exp}$) or binary operators (e.g. $\texttt{add}, \texttt{mul}$) and leaves are variables and constants (e.g. x_1, 3.14) and ii) evolving a population of analytical expressions through selection, mutation, and crossovers. Being a greedy search approach, GP algorithms are prone to falling into local minima, and extensive exploration leads to relatively large run

[*]Equal contribution [1]CSEM SA, Alpnach [2]ETH Zürich [3]Meta AI [4]Sorbonne Université, CNRS, ISIR. Correspondence to: Tommaso Bendinelli <tommaso.bendinelli@csem.ch>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

times. In practice with time constraints, such as the 24 hours-limit in (La Cava et al., 2021), the most accurate GP methods provide expressions with overly large complexity thus preventing the derivation of meaningful physical insights; on the Feynman datasets (Udrescu and Tegmark, 2020), whose expressions have averaged complexity 20 as defined in (La Cava et al., 2021), the current state-of-the-art (Burlacu et al., 2020) predicts expressions with averaged complexity 100. Up to our knowledge, injection of prior information in GP methods can only be accomplished by filtering during selection, e.g. using properties like function positivity or convexity (Kronberger et al., 2022; Haider et al., 2022). This strategy is inherently greedy and can result in the selection of suboptimal expressions due to early convergence to local minima. Other forms of high-level prior information available to the user, e.g. complexity of the expected expression, can hardly be incorporated into GP algorithms. Recently (Mundhenk et al., 2021), a combination of neural networks and genetic programming (GP) has been proposed to improve the performance of symbolic regression. The neural network is used to generate the initial population for a GP algorithm, resulting in a hybrid approach that combines the strengths of both methods. This combination allows for the ability to learn patterns and explore a large solution space, resulting in remarkable performances. However, these systems are not easily controllable, meaning that it can be difficult for the user to constrain the predictions to conform to high-level properties that are known from prior knowledge of the problem.

AI-Feynman. Recent studies (Udrescu and Tegmark, 2020; Udrescu et al., 2020) have investigated the idea of constraining the search to expressions that exhibit particular properties, such as compositionality, additivity, and generalized symmetry. By utilizing these properties, the task of SR becomes significantly less complex as it leverages the modular nature of the resulting expression trees. However, these approaches necessitate fitting a new neural network for every new input dataset and then examining the trained network to identify the desired properties, leading to an inevitably time-consuming process.

Neural Symbolic Regression. Inspired by recent advances in language models, a line of work named *Neural Symbolic Regression* (NSR), tackles SR as a natural language processing task (Biggio et al., 2020; 2021; Valipour et al., 2021; d'Ascoli et al., 2022; Kamienny et al., 2022; Vastl et al., 2022; Li et al., 2022; Becker et al., 2022). NSR consists of two primary steps: firstly, large synthetic datasets are generated by i) sampling expressions from a prior distribution $p_\theta(\mathcal{E})$ where θ is a parametrization induced by an off-the-shelf expression generator (Lample and Charton, 2019), ii) evaluating these expressions on a set of points $\mathbf{x} \in \mathbb{R}^d$ where d is the feature dimension, e.g. sampled from a uniform distribution. Secondly, a generative model $g_\phi(\mathcal{E}|\mathcal{D})$, practically a Transformer (Vaswani et al., 2017) parametrized by weights ϕ, that is conditioned on input points $\mathcal{D} = (\mathbf{x}, \mathbf{y})$, is trained on the task of next-token prediction with target the Polish notation of the expression. NSR predicts expressions that share properties of their implicitly biased synthetic generator $p_\theta(\mathcal{E})$. Control over the shape of the predicted expressions, e.g complexity or sub-expression terms, boils down to a sound design of the generator and the pipeline introduced in (Lample and Charton, 2019) allows only limited degrees of freedom such as operators, variables, constants probability, and tree depth. Similarly to querying a text-to-image generative model (Ramesh et al., 2022; Saharia et al., 2022) with a prompt, the SR practitioner might want to restrict the class of predicted expressions to be in a subclass $h(\mathcal{E}) \subset \mathcal{E}$ by using privileged information. Examples of $h(\mathcal{E})$ can be the class of expressions with low complexity, or that include a specific sub-expression like $e^{-\sqrt{x_1^2+x_2^2}}$. However, a trained NSR model $g_\phi(\mathcal{E}|\mathcal{D})$ can only be adapted to $h(\mathcal{E})$ in one of two ways: i) by using rejection sampling, which is time-inefficient and does not guarantee to find candidate expressions with the expected inductive biases, or ii) by designing a new generator with the desired properties and fine-tuning the model on the new dataset, which is a tedious and time-consuming task.

Contributions

In this work, we propose a new method called *Neural Symbolic Regression with Hypotheses* (NSRwH) to address the aforementioned limitations of NSR algorithms. NSRwH efficiently restricts the class of predicted expressions of NSR models *during inference*, if provided privileged information $\mathcal{D}_{\text{PI}}$, with a simple modification to both the model architecture and the training data generation: with the training set of expressions from $p_\theta(\mathcal{E})$, we produce descriptions $\mathcal{D}_{\text{PI}}$, e.g. appearing operators or complexity, and feed this meta-data into the Transformer model as an extra input, i.e. $g_\phi(S|\mathcal{D}, \mathcal{D}_{\text{PI}})$. During training, we use a masking strategy to avoid our model considering sub-classes of expressions when no privileged information is provided. We show that our model exhibits the following desirable characteristics:

1. In a similar vein to the recent literature on expression derivation and integration (Lample and Charton, 2019) and mathematical understanding capabilities of Transformers (Charton, 2022), our results demonstrate that Transformer models can succeed in capturing complex, high-level symbolic expression properties, such as complexity and symmetry.
2. The proposed model is able to output expressions that closely align with user-determined privileged information and/or hypotheses on the sought-for expression

when it is conditioned on such information. This makes the model effectively *controllable* as its output reflects the user's expectations of specific high-level properties. This stands in contrast to previous work in the NSR and GP literature, where steering symbolic regressors toward specific properties required either retraining from scratch or using inefficient post hoc greedy search routines.

3. The injection of privileged information provides significant improvements in terms of recovery rate. Such an improvement is, as expected, proportional to the amount of conditioning signal provided to the model. This effect is even more apparent in the case where numerical data are corrupted by noise and in the small data regime, where standard NSR approaches witness a more marked performance deterioration.

4. We empirically demonstrate that incorporating conditioning hypotheses not only enhances the controllability of NSRwH but also improves its exploration capabilities, in contrast to standard NSR approaches that rely solely on increasing the beam size. In particular, we show that injecting a large number of hypotheses randomly chosen from a large pool of candidates results in better exploration performance compared to a standard NSR approach operating with a large beam size.

In essence, our approach provides an additional degree of freedom to standard NSR algorithms by allowing the user to quickly steer the prediction of the model in the direction of their prior knowledge at inference time. This is accomplished by leveraging established techniques from language modeling and prompt engineering. The paper is structured as follows: in Section 2, we describe our data generation pipeline and the model architecture; in Section 3 we detail our experimental setup and report our empirical results and in Section 4 we discuss promising future directions and the current limitations of our approach. Code is available at `https://github.com/SymposiumOrganization/ControllableNeuralSymbolicRegression`.

2. Method

2.1. Notation and framework

A symbolic regressor is an algorithm that takes as input a dataset $\mathcal{D}$ of n features-value pairs $(\mathbf{x_i}, y_i) \sim \mathbb{R}^d \times \mathbb{R}$, where d is the feature dimension, and returns a symbolic expression $e \sim \mathcal{E}$ such that $\forall(\mathbf{x}_i, y_i) \in \mathcal{D},\ e(\mathbf{x}_i) = \tilde{y}_i \approx y_i$. NSR is a class of SR algorithms that learns a distribution model $g_\phi(\mathcal{E} \mid \mathcal{D})$, parametrized by a neural network with weights ϕ, over symbolic expressions conditioned on an input dataset $\mathcal{D}$. In this work, we introduce NSRwH, a new subclass of neural symbolic regressors, that allows for conditioning their predictions with user-specified prior knowledge about the output expression. More concretely, given a set of privileged information $\mathcal{D}_{\text{PI}}$, NSRwH approaches are trained to model the conditional distribution $g_\phi(\mathcal{E} \mid \mathcal{D}, \mathcal{D}_{\text{PI}})$. An illustration of the proposed approach is shown in Fig. 1.

2.2. Dataset generation

In our framework, a synthetic training sample is defined as a tuple $(e, \mathcal{D}, \mathcal{D}_{\text{PI}})$ where each element is produced as explained below.

Generating e and $\mathcal{D}$. As in other NSR works (Biggio et al., 2021; Kamienny et al., 2022; Valipour et al., 2021), we sample analytical expressions e from $\mathcal{E}$ using the strategy introduced by Lample and Charton (2019): random unary-binary trees with depth between 1 and 6 are generated, then internal nodes are assigned either unary or binary operators as described in Table 2 in Appendix A.1 according to their arity, and leaves are assigned variables $\{x_d\}_{d \leq 5}$ and constants. In order to generate $\mathcal{D}$, for each expression e, we sample a support of n points $\mathbf{x}_i \in \mathbb{R}^d$. The values for each coordinate are drawn independently from one another using a uniform distribution $\mathcal{U}$, with the bounds randomly selected from the interval $[-10, 10]$. Next, the expression value y_i is obtained via the evaluation of the expression e on the previously sampled support. More details on the generation of numerical data can be found in Appendix A.1

Generating $\mathcal{D}_{\text{PI}}$. Privileged information $\mathcal{D}_{\text{PI}}$ is composed of *hypotheses*. From an expression e, we extract the following properties:

- **Complexity.** We use the definition of complexity provided by (La Cava et al., 2021), i.e. the number of mathematical operators, features, and constants in the output prediction.

- **Symmetry.** We use the definition of generalized symmetry proposed in (Udrescu et al., 2020): f has generalized symmetry if the d components of the vector $\mathbf{x} \in \mathbb{R}^d$ can be split into groups of k and $d-k$ components (which we denote by the vectors $\mathbf{x}' \in \mathbb{R}^k$ and $\mathbf{x}'' \in \mathbb{R}^{d-k}$) such that $f(\mathbf{x}) = f(\mathbf{x}', \mathbf{x}'') = g[h(\mathbf{x}'), \mathbf{x}'']$ for some unknown function g.

- **Appearing branches.** We consider the set of all the branches that appear in the prefix tree of the generating expression. For instance, for $x_1 + \sin x_2$ this set would be $[+, x_1, +x_1, \sin, \sin(x_2), x_2, +\sin, +\sin(x_2)]$. For each expression, in the training set, we sample a subset of this list, ensuring that each element of the subset is sampled with a probability inversely proportional to its length squared and that the full expression tree is never given to the model.

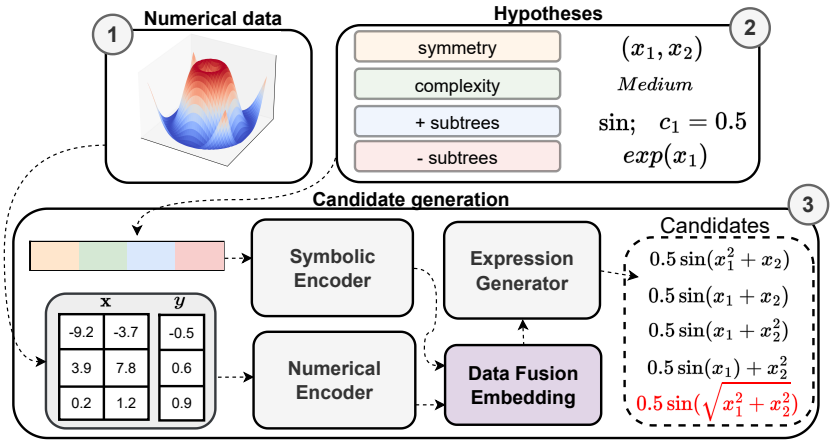

Figure 1. **Neural Symbolic Regression with Hypotheses.** 1) A dataset of numerical observations is obtained; 2) the user formulates a set of hypotheses based on some properties they believe the final expression should possess. After being tokenized independently, the properties tensors are concatenated to form a unique conditioning tensor; 3) numerical data as well as the formulated hypotheses are given as input to two different encoders. Their outputs are then summed and the resulting tensor is processed by a decoder which outputs a set of candidate equations. For NSRwH to be effective and controllable, the candidate expressions should respect the input hypotheses.

- **Appearing constants.** We also enable the inclusion of a-priori-known constants at test time. We implement this conditioning by drawing inspiration from the concept of pointers in computer programming: we give as input to the model the numerical constant and a pointer, and the model has to place the input pointer in the correct location in the output prediction. This approach does not require representing each constant with a different token, hence preventing the explosion of the output vocabulary size.

- **Absent branches.** We condition our model with the information about subtrees not appearing in true expression. The procedure for extracting this property follows the same logic as the extraction of appearing subtrees.

In the rest of the paper, we refer to these properties as `Complexity`, `Symmetry`, `Positive`, `Constants`, and `Negative`. It is important to note that the set of properties used in this work is not exhaustive and can easily be expanded based on the user's prior knowledge. We provide more details on their exact computation along with a practical example of their extraction in Appendix A.2.

2.3. Model

Architecture. We use NeSymReS (Biggio et al., 2021) as our base neural symbolic regressor for its simplicity and in the following, we explain how to incorporate the description $\mathcal{D}_{\text{PI}}$ as an input to the model $g_\phi(e|\mathcal{D}, \mathcal{D}_{\text{PI}})$. Note that the very same conditioning strategy can easily be applied to alternative more advanced NSR architectures, such as those introduced in (Valipour et al., 2021; Kamienny et al., 2022). NSRwH consists of three architectural components: a numerical encoder enc_{num}, a symbolic encoder enc_{sym}, and a decoder dec (see Fig. 1). Numerical data $\mathcal{D}$, represented by a tensor of size (B, n, D), where B is the batch size, n is the number of points and D is the sum of dependent/independent variables ($D = 5 + 1$), is converted into a higher dimensional tensor $\mathcal{D}'$ of size (B, n, H) using a multi-hot bit representation according to the half-precision IEEE-754 standard and an embedding layer, where H is the hidden dimension (512 for our experiments). $\mathcal{D}'$ is then processed by a set-transformer encoder (Lee et al., 2019), a variation of (Vaswani et al., 2017) with better inference time and less memory requirement, to produce a new tensor $\mathbf{z}_{num} = enc_{num}(\mathcal{D}')$ of size (B, S, H), where S (50 for our experiments) is the sequence length after the encoder processing. $\mathcal{D}_{\text{PI}}$, represented by a tensor of size (B, M) where M is the number of tokens composing the conditioning hypotheses string, is converted into a higher dimensional tensor $\mathcal{D}'_{PI}$ of size (B, M, H) via an embedding layer. This new tensor is then input into an additional set-transformer to produce a tensor $\mathbf{z}_{sym} = enc_{sym}(\mathcal{D}'_{PI})$ of size (B, S, H). $\mathbf{z}_{num}$ and $\mathbf{z}_{sym}$ are summed together to produce a new tensor $\mathbf{z}_{fused} = \mathbf{z}_{num} + \mathbf{z}_{sym}$ of size (B, S, H). Finally, $\mathbf{z}_{fused}$ is fed into a standard transformer decoder network, dec, that autoregressively predicts token by token the corresponding expressions using beam search for the best candidates. We resorted to the element-wise summation of $\mathbf{z}_{num}$ and $\mathbf{z}_{sym}$ instead of concatenation in order to reduce memory usage in the decoder, which increases quadratically with the sequence length due to cross attention.

Training and testing. As done in all NSR approaches, we use the cross-entropy loss on next-token prediction using

teacher-forcing (Sutskever et al., 2014), i.e. conditioning $g_\phi(\tilde{e}_{t+1}|e_{1:t}, \mathcal{D}, \mathcal{D}_{\text{PI}})$ on the first t tokens of the ground-truth e. As for NeSymReS, we "skeletonize" target expressions by replacing constants by a constant token $\diamond$ or, in the case the position of the constant is known a priori, a pointer symbol is used. To prevent our model from being dependent on privileged information at test time, we include training examples with partial privileged information. This means we only provide the model with a subset of all the possible conditionings. For example, only `Positive` and `Symmetry` are given, while `Negative`, `Complexity` and `Constants` are masked out. This is a useful feature of our model as, depending on the use case, some information might not be available and we want the model to still be usable in those cases. At test time, as for NeSymReS, we use beam search to produce a set of predicted expressions, then we apply Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) (Fletcher, 1987) to recover the values of the constants by minimizing the squared loss between the original outputs and the output from the predicted expressions. More details on the model and training hyperparameters can be found in Appendix B.

3. Experiments

In this section, we first introduce the datasets and metrics used to evaluate the model and then we present our experiments aimed to assess different properties of NSRwH, including its controllability, and its performance when $\mathcal{D}_{\text{PI}}$ is available, and when it is not. Over the experimental section, we use the standard NeSymReS as a reference baseline, which is referred to as `standard_nesy` in the plots. While our approach could be used with other NSR methods, we have chosen to solely focus on NeSymReS as a baseline model. This allows us to better comprehend the advantages that come from conditioning, instead of assessing various NSR models with distinct numerical input architectures and expression generators. As mentioned in Section 1, GP methods can be hardly conditioned on our set of properties, and as such a comparison with them would be unfair.

3.1. Experimental setup

To generate training data, we follow the pipeline introduced in Section 2.2 resulting in a training set comprising 200 million symbolic expressions with up to 5 variables. The datasets and metrics used to test NSRwH are described below.

Datasets. We use five different databases in our experiments, each characterized by different degrees of complexity: 1) `train_nc`: this dataset comprises 300 symbolic expressions, not including numerical constants. The number of independent variables varies from 1 to 5. The equations are sampled from the same distribution of the training set; 2) `train_wc`: it comprises the same equations of `train_nc` but with numerical constants randomly included in each expression. As such, it represents a more challenging framework than the previous one as the model has the output constant placeholders in the correct positions and BFGS has to find their numerical value; 3) `only_five_variables_nc`: it consists of 300 expressions without constants, strictly selected to have 5 independent variables each. The dataset has been chosen to assess the performance of our algorithm in a higher-dimensional scenario; 4) `AIF`: it comprises all the equations with up to 5 independent variables extracted from the publicly available AIFeynman database (Udrescu and Tegmark, 2020). It includes equations from the *Feynman Lectures on Physics series* and serves to test the performance of NSRwH on mathematical expressions stemming from several physics domains; 5) `black_box`: it is extracted from the ODE-Strogatz (Strogatz, 2018) databases and serves to evaluate NSRwH in the case where no prior information is available. As also noted by Kamienny et al. (2022), these datasets are particularly challenging as they include non-uniformly distributed points and have different input support bounds than those used by our dataset generation pipeline.

Metrics. We use three different metrics to evaluate our models: 1) `is_satisfied`: this metric measures the percentage of output predictions that agree with a certain property. For all the properties this metric is calculated as follows: given a known equation, we calculate the mean over the total number of times the predictions of the model across the beam size matches the property under consideration. The final metric value is given by the average of the above quantity across all the equations in the test set; 2) `is_correct`: given a test equation, for each point (x, y) and prediction $\hat{y}$, we calculate `numpy.is_close(`$y, \hat{y}$`)`. Then, we take the mean over all the support points and obtain a real number. If this number is larger than 0.99, we deem our prediction to match the true one and we assign a score of 1, otherwise 0. The final metric value is obtained by calculating the percentage of correctly predicted equations over the entire test set. Importantly, the support points are chosen to be different from those fed into the model at test time; 3) $R^2_{\texttt{mean}}$: given a test equation, and n points $\{x_i, y_i\}_{i=1}^n$, and the corresponding predictions $\{\hat{y}_i\}_{i=1}^n$, we calculate the coefficient of determination, also known as R^2 score, as defined below:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2} \quad \text{where} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$$

The final metric is calculated by taking the mean of the R^2 scores obtained for each equation in the test set. More details on the test datasets and metrics can be found in Appendix C.

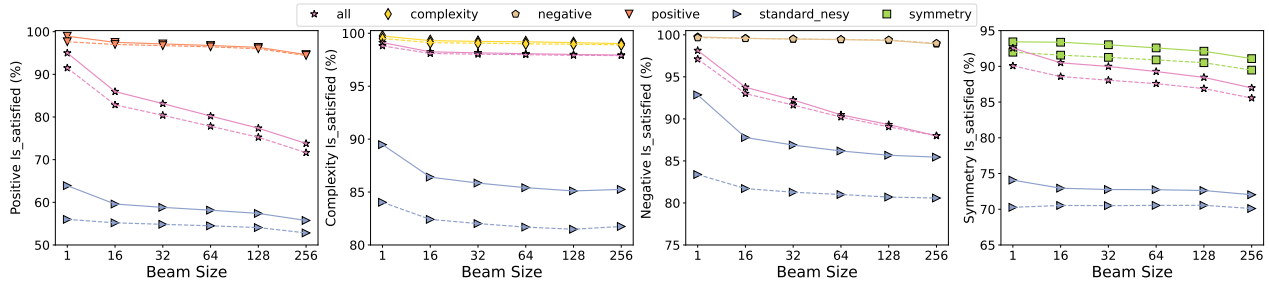

Figure 2. **Controllability and property matching**: The panels show the level of agreement with various types of input conditioning signals – in terms of the `is_satisfied` metric – of our model and the unconditioned baseline (`standard_nesy`), both in the noiseless case (full line) and when noise is injected in the input data (dashed line), as a function of the beam size. The reported results are averaged across all datasets apart from `black_box`.

3.2. Can transformers efficiently restrict the inference space using descriptions?

Arguably, the main challenge in symbolic regression is represented by the extremely large search space over mathematical expressions. Methods based on brute force search techniques are doomed to fail or to fall into spurious local minima. The goal of this section is to show that neural symbolic regression algorithms can be *controlled* in such a way that their output adheres with a set of pre-specified inductive biases – meant to narrow the search space – on the nature of the sought expression.

Each panel in Fig. 2 shows the evolution of the `is_satisfied` metric for various types of conditioning properties as the beam size increases, with and without noise injected in the input data. Noise perturbations are injected in the output of the input data, y, according to the following formula:

$$\tilde{y} = y + \rho\epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, |y|) \;\; and \;\; \rho = 0.01. \tag{1}$$

The goal of the experiment is twofold: first, we want to assess whether NSRwH is able to capture the meaning of the input conditing, and second, we want to verify how consistent such an agreement is as we increase the beam size and inject noise.

From the results in Fig. 2, we can observe that the predictions of NSRwH attain a very high `is_satisfied` score for all the evaluated properties. This is in contrast with the unconditioned model which does not consistently capture the underlying properties. This is particularly evident when noise is added to the data, as our model shows robustness to such perturbations, while the standard NSR method experiences greater variations. This is explained by the fact that the standard method grounds its predictions solely on numerical data. As such, when these are severely corrupted, results deteriorate accordingly. We also note that when all possible conditioning properties are given to the model (see `all`), NSRwH tends to underperform with respect to the case when a single property is provided, in particular as the beam size increases. This is likely due to interference effects between different hypotheses, which causes the model, at large beam sizes, to select the subset of them that is more consistent with the numerical data.

3.3. Can NSRwH leverage privileged information?

In this section, we investigate whether the ability of NSRwH to capture the meaning of the input properties can be leveraged to improve performance.

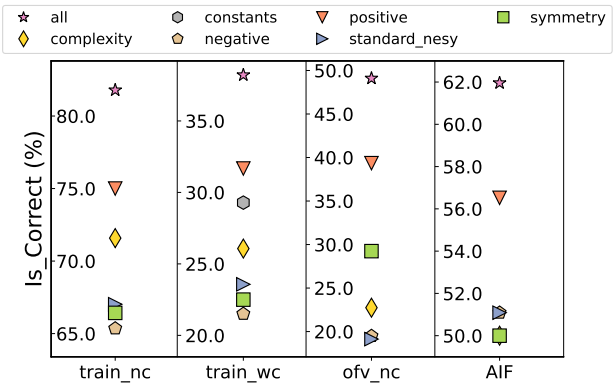

Figure 3. **Conditioning improves performance.** Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric. Each column corresponds to a different test dataset.

To perform these experiments we make use of `is_correct` metric introduced above and we study how performance changes under the effect of noise, number of input data, and amount of conditioning. The beam size for both NSRwH and NeSymReS is set to 5. We start our evaluation from the noiseless case, i.e. no noise is injected in the expressions' evaluation at test time. As such, the mapping between input covariates and the output

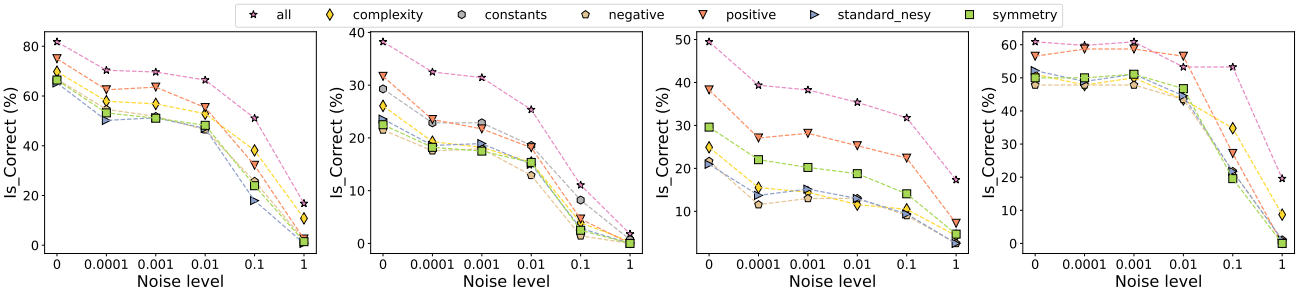

Figure 4. **Dependence on the input noise.** Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric, as a function of the noise level in the input data, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

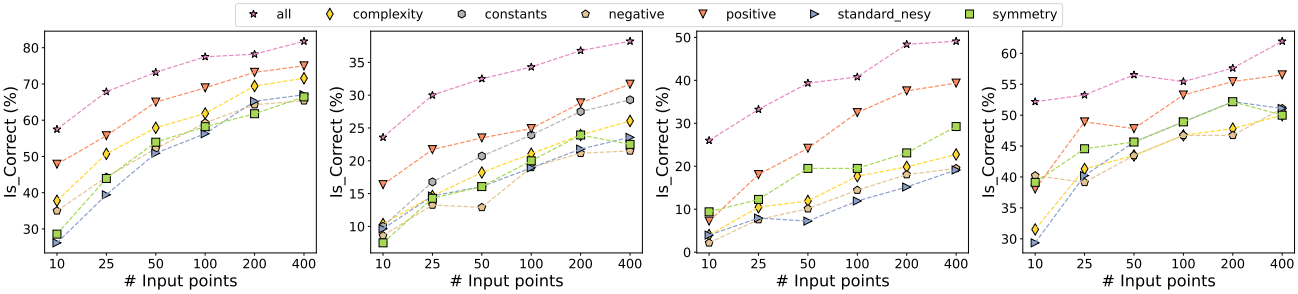

Figure 5. **Dependence on the number of input points.** Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the `is_correct` metric, as a function of the number of input points, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

value is exactly represented by the ground truth symbolic expression. Fig. 3, shows the performance of NSRwH and the unconditioned model in terms of the `is_correct` metric described above and the different properties provided at test time.

Generally, NSRwH efficiently leverages the prompted information to improve its performance. Among the considered individual properties, `Positive` is the most effective one. However, it is interesting to note that `Symmetry` is particularly effective on the `only_five_variables_nc` (ofv_nc) dataset. This is due to the high-dimensional nature of the dataset and the fact the symmetry information is more useful in such cases. Providing information about the ground-truth constants leads to significant performance improvements on the `train_wc` dataset, showcasing the effectiveness of our strategy of providing numerical constants to the model. Finally, `all`, the combination of all the considered properties, is by far the most impacting conditioning. It is noteworthy that, while in some cases the performance of individual properties may not be significantly better than the baseline, their combination (`all`) proves to be highly successful, indicating that the model is able to combine them together effectively.

3.3.1. Case with noise

In this paragraph, we explore the more challenging scenario where noise is injected into the output value y at test time. In particular, we use Eq. 1 with six different noise levels $\rho \in \{0, 0.0001, 0.001, 0.01, 0.1, 1\}$. The beam size for both NSRwH and NeSymReS is set to 5 in this experiment. As shown in Fig. 4, the performance improvements are even more pronounced than in the noiseless case shown in Fig. 3. This illustrates that the incorporation of meaningful inductive biases in our model enables it to effectively manage the impact of noise and, as a result, improves generalization.

3.3.2. Dependence on the number of input points.

In a similar manner as the previous paragraph, this investigation examines whether NSRwH can utilize input conditioning to enhance its performance in the challenging, yet common scenario where small datasets are used as input. As before, the beam size for both NSRwH and NeSymReS is set to 5. As illustrated in Fig. 5, as the number of input points decreases, the performance of both the conditioned and unconditioned models also declines. However, in NSRwH this effect is significantly reduced, keeping relatively high levels of accuracy even when working in the small data regime.

3.3.3. Dependence on the amount of conditioning

In this section, we investigate how the performance changes as we increase the amount of conditioning. We conduct this experiment using both `Positive` and `Constants` as we can easily control the degree of conditioning by adjusting the probability of the number of subtrees and constants that appear, respectively. As before, the beam size for both NSRwH and NeSymReS is set to 5. Fig 6 shows how the value of the `is_correct` metric changes as we vary the amount of `Positive` and `Constants` information. As expected, a monotonic trend can be observed for both properties as the amount of conditioning is increased. The peak in performance is reached when the two properties are provided in the largest amounts, suggesting that the model can combine the two prompts to maximize its prediction accuracy.

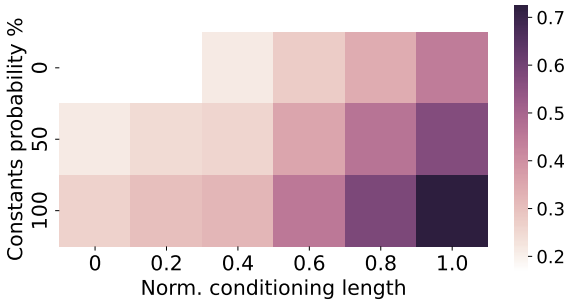

Figure 6. **Dependence on the amount of conditioning.** The heatmap shows how changing the probability of appearing subtrees and constants affects NSRwH's performance on the `train_wc` dataset, measured by the `is_correct` metric. The y-axis shows the probability of constants appearing, with 100% meaning all constants are inputted. The x-axis shows the normalized conditioning length, with 1.0 meaning the model sees positive sub-branches whose length adds up to the prefix ground truth.

3.4. What if no assumptions can be made?

This section investigates the scenario where no prior knowledge is available to condition the model.

Model Type	`is_correct`	R^2_{mean}
Random Positive Conditions	**0.35 ± 0.06**	**0.86 ± 0.05**
Standard Model [5K]	0.23 ± 0.00	0.74 ± 0.05
Standard Model [10]	0.12 ± 0.04	0.32 ± 0.03

Table 1. **No privileged information available.** Comparison between NSRwH with randomly sampled hypotheses, a standard NSR approach (NeSymReS) with beam size 5000, and a standard NSR approach with beam size 10 (same as NSRwH). Results are averaged over 5 runs.

The objective of the experiment is to determine if using NSRwH with randomly sampled hypotheses can outperform a standard NSR model, which can only improve its predictions by increasing the beam size. According to prior work, conventional search techniques of NSR, such as beam search and random sampling, quickly reach a saturation point in exploring the search space, making larger beam sizes ineffective for exploration (see Fig. 16 in (Kamienny et al., 2022)). The experiment is conducted on the `black_box` dataset. The standard model uses a large beam size of $M = 5000$, which is within the saturation regime, and NSRwH uses $N = 500$ diverse, randomly sampled `Positive` conditionings with a beam size of $M/N = 10$ for each. As such, both methods utilize the same computational budget. Table 1 shows that NSRwH outperforms the standard NSR model on the `black_box` dataset. We highlight that the policy used to randomly sample positive operators is very sparse and highly suboptimal. As such, the design of more effective search routines over the space of properties represents an interesting avenue for future research.

4. Discussion

Conclusive remarks. This work presents a novel approach for symbolic regression that enables the explicit incorporation of inductive biases reflecting prior knowledge of the problem at hand. In contrast to previous works, this can be effectively done at test time, drastically reducing the computational overhead. Thanks to this property, our model better lends itself to online and interactive applications of symbolic regression, thus enabling fast hypothesis testing, a highly desirable feature for scientific discovery applications. We demonstrate the value of this approach with a number of examples and ablation studies where numerical data is scarce or affected by noise.

Limitations and future work The main limitation of the proposed approach is realized in the scenario where no prior knowledge is available. In this case, the performance gains obtained in Section 3.3 are not guaranteed. However, in Section 3.4, our final experiment suggests an intriguing opportunity for future research - leveraging NSRwH's extra degree of freedom to explore the equation space more efficiently. In addition, the properties investigated in this work are not exhaustive and it is conceivable to include additional forms of prior knowledge, such as alternative definitions of the complexity of mathematical expressions based on syntax or semantics (Kommenda et al., 2015; Vladislavleva et al., 2009). Finally, we remark that thanks to its simplicity, the same idea at the basis of NSRwH can be applied to more advanced NSR algorithms, like the one recently proposed by (Kamienny et al., 2022), likely resulting in further performance improvements. We intend to investigate the above questions in future work.

References

Sören Becker, Michal Klein, Alexander Neitz, Giambattista Parascandolo, and Niki Kilbertus. Discovering ordinary differential equations that govern time-series. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

Luca Biggio, Tommaso Bendinelli, Aurelien Lucchi, and Giambattista Parascandolo. A seq2seq approach to symbolic regression. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales, 2021.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon c++: An efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, page 1562–1570, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371278. doi: 10.1145/3377929.3398099. URL https://doi.org/10.1145/3377929.3398099.

François Charton. What is my math transformer doing?–three results on interpretability and generalization. *arXiv preprint arXiv:2211.00170*, 2022.

Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33:17429–17442, 2020.

Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.

Christian Haider, Fabrício Olivetti de França, Gabriel Kronberger, and Bogdan Burlacu. Comparing optimistic and pessimistic constraint evaluation in shape-constrained symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 938–945, 2022.

Alberto Hernandez, Adarsh Balasubramanian, Fenglin Yuan, Simon AM Mason, and Tim Mueller. Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *npj Computational Materials*, 5(1): 1–11, 2019.

Evgeniya Kabliman, Ana Helena Kolody, Johannes Kronsteiner, Michael Kommenda, and Gabriel Kronberger. Application of symbolic regression for constitutive modeling of plastic deformation. *Applications in Engineering Science*, 6:100052, 2021.

Kadierdan Kaheman, J Nathan Kutz, and Steven L Brunton. Sindy-pi: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A*, 476(2242):20200279, 2020.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Michael Kommenda, Andreas Beham, Michael Affenzeller, and Gabriel Kronberger. Complexity measures for multi-objective symbolic regression. In *Computer Aided Systems Theory – EUROCAST 2015*, pages 409–416. Springer International Publishing, 2015. doi: 10.1007/978-3-319-27340-2_51. URL https://doi.org/10.1007%2F978-3-319-27340-2_51.

Gabriel Kronberger, Fabricio Olivetti de França, Bogdan Burlacu, Christian Haider, and Michael Kommenda. Shape-constrained symbolic regression—improving extrapolation with prior knowledge. *Evolutionary Computation*, 30(1):75–98, 2022.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance, 2021. URL https://arxiv.org/abs/2107.14351.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

Jiachen Li, Ye Yuan, and Hongze Shen. Symbolic expression transformer: A computer vision approach for symbolic regression. *ArXiv*, abs/2205.11798, 2022.

He Ma, Arunachalam Narayanaswamy, Patrick Riley, and Li Li. Evolving symbolic density functionals. *arXiv preprint arXiv:2203.02540*, 2022.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

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, Dec 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4. URL https://doi.org/10.1186/s13040-017-0154-4.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923): 81–85, 2009.

Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. *arXiv preprint arXiv:2205.13134*, 2022.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

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. 2020. doi: 10.48550/ARXIV.2006.10782. URL https://arxiv.org/abs/2006.10782.

Harsha Vaddireddy, Adil Rasheed, Anne E Staples, and Omer San. Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data. *Physics of Fluids*, 32(1):015113, 2020.

Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

Martin Vastl, Jonás Kulhánek, Jirí Kubalík, Erik Derner, and Robert Babuska. Symformer: End-to-end symbolic regression using transformer-based architecture. *ArXiv*, abs/2205.15764, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ekaterina Vladislavleva, Guido 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*, 13:333–349, 2009.

Yiqun Wang, Nicholas Wagner, and James M Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

A. Dataset generation

A.1. Generating $\mathcal{D}$

We build our training dataset first by generating symbolic expressions skeletons (i.e. mathematical expressions where the values of the constants are replaced by placeholder tokens) using the framework introduced by (Lample and Charton, 2019). Our vocabulary consists of the unary and binary operators shown in Table 2. We considered scalar (output dimension equal to 1) expressions with up to 5 independent variables with a maximum prefix length and depth of 20 and 6 respectively.

To obtain the mathematical expression and corresponding numerical evaluation during training for each equation we adopt the following procedure:

- An equation skeleton is randomly sampled from the pool of symbolic expression skeletons
- The sampled expression is simplified using the `simplify` function from `Sympy` (Meurer et al., 2017) in order to remove any redundant term.
- Constants of the skeleton are sampled from $\mathcal{U}(-10, -10)$ if they are additive, and logarithmically from $\mathcal{U}(0.05, 10)$ if multiplicative.
- The extrema of the support for each independent variable is sampled independently from a uniform distribution $\mathcal{U}(-10, 10)$ with the distance between the left and right extrema of at least 1.
- For each independent variable, n input points are sampled from the previously sampled support, where n is sampled between $\mathcal{U}(1, 1000)$. Support points that lead to absolute values bigger than 65504 or NANs are discarded and re-sampled.
- We evaluate the sampled expression on the previously obtained support points by using the `lambdify` function from `Sympy` (Meurer et al., 2017).

Arity	Operators

Unary	`sqrt, pow2, pow3, pow4`
	`inv, log, exp`
	`sin, cos, asin`

Binary	`add, sub, mul, div`

Table 2. Operators used in our data generation pipeline.

As the input evaluations can lead to large values, we follow (Biggio et al., 2021) and we convert them from float to a multi-hot bit representation according to the half-precision IEEE-754 standard before feeding them into the model.

A.2. Generating $\mathcal{D}_{\text{PI}}$

A.2.1. Complexity

The complexity of a sentence is determined by the sum of the number of nodes and leaves in the expression, as outlined in (La Cava et al., 2021). Each complexity value is represented by a unique token, ranging from 1 (i.e. x_1) to 20.

A.2.2. Symmetry

We use the definition of generalized symmetry proposed by (Udrescu et al., 2020): f has generalized symmetry if the d components of the vector $\mathbf{x} \in \mathbb{R}^d$ can be split into groups of k and $d-k$ components (which we denote by the vectors $\mathbf{x}' \in \mathbb{R}^k$ and $\mathbf{x}'' \in \mathbb{R}^{d-k}$) such that $f(\mathbf{x}) = f(\mathbf{x}', \mathbf{x}'') = g[h(\mathbf{x}'), \mathbf{x}'']$ for some unknown function g. As explained in (Udrescu et al., 2020), in order to check the presence of generalized symmetry in the set of variables $\mathbf{x}'$, it is sufficient to check whether the normalized gradient of f with respect to $\mathbf{x}'$ is independent on $\mathbf{x}''$, i.e. $\frac{\nabla_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x}'')}{|\nabla_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x}'')|}$ is $\mathbf{x}''$-independent. We have created two tokens for each symmetry combination, one to represent the presence of symmetry and one to represent its absence. The total number of tokens is 50, as there are 32 possible symmetry combinations when there are five variables,

but some of them are not informative and are excluded, leaving 25 useful combinations. When the number of variables is less than five, only the tokens related to the actual variables are passed to the model (see example in Section A.2.5).

A.2.3. APPEARING / ABSENT BRANCHES

To sample both appearing and absent branches for an expression, we create two candidate pools: a positive and a negative one. The positive pool is created by using the Depth-First-Search (DFS) algorithm to list all the subtrees within the current prefix expression and then by removing the subtree corresponding to the entire expression and other non-informative subtrees like x_i. The negative pool is created by filtering out the branches that are already present in the current prefix tree from a pre-computed set of branches, obtained from a large pool of expression trees within the training distribution. We sample subtrees from these pools with a probability proportional to the inverse of their length squared, both during training and evaluation. To regulate the total information given to the model, two parameters, p_p for the positive subtrees and p_n for the negative subtrees, are used. The product of p_p (p_n) and the ground truth length determines the total number of tokens provided to the model, denoted as s_p (s_n). Positive (Negative) subtrees are sampled until the aggregate sum of their lengths, $\sum_{i=1}^{N} \texttt{len}(\text{sampled subtrees}_i)$, reaches the s_p (s_n) value. Sub-branches are separated by special separator tokens.

A.2.4. CONSTANTS

Each a-priori-known constant is assigned to a specific symbol, such as `pointer_0` for the first constant, `pointer_1` for the second, and so on. We then give the symbolic encoder the corresponding pointer and a numerical embedding obtained by first converting the known constant in its equivalent 16-bit representation and then passing it through a learnable linear layer that makes its dimension match that of the symbolic embedding. In the target expression, we replace the standard constant placeholder with the `pointer_`i token in the expression. At training and evaluation stages, we regulate the probability of a constant being a-priori-known with a parameter p_c.

A.2.5. EXAMPLE OF EXTRACTION AND PROCESSING OF CONDITIONING INFORMATION

This section provides a concrete example of how different conditionings are extracted and processed to be fed into our model. Consider the expression $x_3 \sin(x_1 + x_2)$.
To determine the `Positive` conditioning, we must first convert it into prefix notation. This is achieved by first rewriting it as ['mul', 'x_3', 'sin', 'add', 'x_1', 'x_2'] and then enumerating all the possible subtrees of the expression, excluding trivial subtrees such as 'x_2' alone. These are: [['add'], ['mul'], ['sin'], ['add', 'x_1'], ['add', 'x_2'], ['mul', 'x_3'], ['add', 'x_1', 'x_2'], ['sin', 'add', 'x_1', 'x_2'], ['mul', 'sin', 'add', 'x_1', 'x_2']]. Positive conditionings are then sampled from this pool with a probability inversely proportional to the length squared of the subtree. So a positive conditioning such as ['mul', 'x_3'] is less likely to be sampled than ['sin'] but more likely than ['mul', 'sin', 'add', 'x_1', 'x_2'].
To obtain the `Negative` conditionings, we generate subtrees at random that are absent from the positive pool. This is achieved by randomly selecting an expression, enumerating the subtrees within it, and then randomly choosing subtrees from the expression that are not present in the positive pool. For example, for the expression above, a negative conditioning could be ['mul', 'x_1'], or ['exp'] since none of these are present in the positive pool. The number of sub-trees supplied to the model is determined by the values of p_p and p_n, and the total length of the expression. For example, if p_p is 0.5, then the total length of the sampled sub-trees will be 3, since the overall length of the ground truth is 6.
`Constant` conditioning would be empty since no constants can be obtained.
`Complexity` conditioning is simply the sum of total nodes and leaves of the prefix expression tree, so in this case, it is equal to 6.
For the `Symmetry` conditioning, we followed the definition given provided by (Udrescu et al., 2020). For our example expression, we will have symmetry between x_1 and x_2 but not between x_1, x_3 or x_2, x_3.
Once computed, the conditionings are wrapped into a string, tokenized, and then fed into the model. The string will have the following form:

[¡Positive¿, 'sin', ¡/Positive¿, ¡Positive¿, 'mul', 'x_3', ¡/Positive¿, ¡Negative¿, 'exp', ¡/Negative¿, ¡Negative¿, 'mul', 'x_1', ¡/Negative¿, 'Complexity=6', 'TrueSymmetryX1X2', 'FalseSymmetryX2X3', 'FalseSymmetryX1X3'].

If some conditionings should be masked, they are simply excluded from the list; for instance, if we only want to provide symmetry conditioning, the string would have the following form:

['TrueSymmetryX1X2', 'FalseSymmetryX2X3', 'FalseSymmetryX1X3']

B. Training and testing details

We trained the model with 200 million equations using three NVIDIA GeForce RTX 3090 for a total of five days with a batch size of 400. As in (Biggio et al., 2021), we used a 5-layer set encoder as our numerical encoder and a five-layer standard Transformer decoder as our expression generator. The conditioning and numerical embedding are summed before the expression generator.
In the training process, the Adam optimizer is employed to optimize the cross-entropy loss, utilizing an initial learning rate of 10^{-4}, which is subsequently adjusted in proportion to the inverse square root of the number of steps taken.
To ensure a fair comparison, the standard model, `standard_nesy`, was trained using the same number of equations and the same numerical encoder and expression generator architecture. In addition, both models have been trained for an equal number of iterations.

B.1. Amount of conditioning during training

During training, we give the model a varying amount of conditioning signals to avoid excessive dependence on them. We adopt the following approach:

- `Positive`: p_p, as defined in the sub-section A.2.3, is 0 with probability 0.7. Otherwise, it is sampled from $\mathcal{U}(0,1)$
- `Negative`: p_n, as defined in the sub-section A.2.3 is 0 with probability 0.7. Otherwise, it is sampled from $\mathcal{U}(0,1)$
- `Complexity`: We provide the complexity token to the network with a probability of 0.3
- `Symmetry`: We provide the symmetry tokens to the network with a probability of 0.2.
- `Constants`: p_c as defined in the sub-section A.2.4 is equal to 0.15.

B.2. Amount of conditioning during testing

We use a variety of conditioning signals, with each combination of signals referred to by a specific term.

- `Positive`: p_p as defined in the sub-section A.2.3 is equal to 0.5. The other conditioning signals are disabled.
- `Negative`: p_n as defined in the sub-section A.2.3 is equal to 0.5. The other conditioning signals are disabled.
- `Complexity`: We provide the complexity token to the network. The other conditioning signals are disabled.
- `Symmetry`: We provide the symmetry token to the network. The other conditioning signals are disabled.
- `Constants`: We provide the value of each constant with a probability of 0.8. The other conditioning signals are disabled.
- `Vanilla`: No conditioning is given (all conditionings are masked). The model sees only the numerical inputs. This is equivalent to the standard model.
- `All`: combines `Positive`, `Negative`, `Complexity`, `Symmetry` and `Constants` conditioning. Each conditioning signal is enabled, with parameters equal to the values mentioned for each individual setting with the sole exception of constants where the probability of providing a constant is set to 0.3.

C. Test datasets and metrics

C.1. Evaluation datasets

We created three datasets, `train_nc`, `train_wc` and `only_five_variables_nc` using the same generator configuration as the training set, but with different initial seeds. For `train_nc` and `train_wc` datasets, we selected 300 equations, removing all constants from the first and selecting random constants for the second. These equations have different levels of complexity. In contrast, for `only_five_variables_nc`, we restricted our dataset to equations with five variables, discarding the others. We also removed any constants from these equations. In addition, we evaluate our model on two open-source datasets, namely `AIF`, consisting of the equation in the AI Feynman database (Udrescu and Tegmark, 2020)

and `black_box` comprising of 14 datasets from the ODE-Strogatz database (Strogatz, 2018). For all experiments except 3.4, the training points were used to both fit constants with BFGS and to select the predicted expression among the beam candidates. Specifically, once the constants were fitted, the expression with the lowest BFGS loss was chosen as the predicted expression. However, since in Section 3.4 a much larger beam size (5000 compared to 5 of the other experiments) was used, we followed a different approach: 60% of the points were used for fitting constants, and the remaining 40% to select the best expression. The expression with the highest R^2 scores on this 40% support was chosen as the predicted expression.

C.2. Evaluation Procedure

For `train_nc`, `train_wc` and `only_five_variables_nc` we sample the support points following the same procedure as in the training pipeline. For AI Feynman equations, we use the support defined in the dataset. For the ODE-Strogatz dataset we followed the approach from (La Cava et al., 2021) and used 75% of the points from the function call `fetch_data` from the PMLB repository for training (Olson et al., 2017) and the remaining for testing.
For the other datasets, to test the quality of our prediction, we sampled 500 points from the OOD support $\mathcal{U}(-25, 25)$. Our criterion for identifying equations as symbolically equivalent to the ground truth was a 99% or higher average output of the `numpy.is_close(`$y, \hat{y}$`)` function across the support points. This threshold accounted for numerical inaccuracies, such as those caused by numerical instability near support points close to zero, so that equations symbolically equivalent were not misclassified due to these errors.

D. Additional results

In this section, we report some additional results obtained by evaluating the model on the R^2 metric. We conclude with a subsection comparing the model obtained by completely masking the symbolic encoder of NSRwH (`vanilla` model) and a standard NSR model without any symbolic encoder (`standard_nesy`).

D.1. R^2 metric

Figs 7, 8 and 9 repeat the analysis performed in the main body but with the R^2 metric instead of the `is_correct` metric. The scores in this section are calculated by extracting the R^2 value for each expression. If such a value is above 0.99, a score of 1 is assigned, otherwise zero. Finally, the so obtained boolean scores are averaged across the entire test set. We refer to this metric as $R^2_{0.99}$.

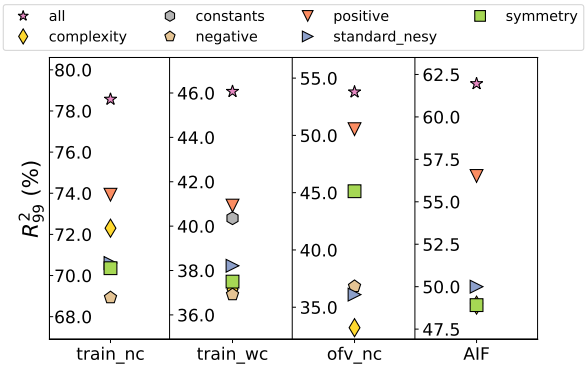

Figure 7. **Conditioning improves performance.** Comparison between NSRwH conditioned with different types of hypotheses and the standard NeSymReS for the different datasets in terms of $R^2_{0.99}$ score

D.2. Comparison between masked NSRwH and standard model

In this section, we compare the fully masked model – referred to as `vanilla` – to the standard NSR method (without a symbolic encoder) – referred to as `standard_nesy`. The goal is to show that their performance is aligned, indicating that NSRwH represents an enhanced version of standard NSR.

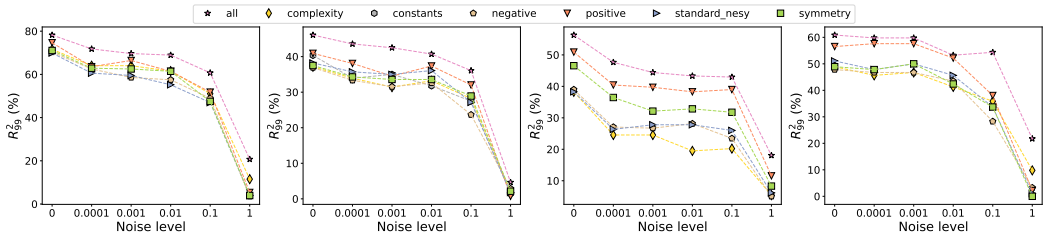

Figure 8. **Dependence on the input noise.** Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the $R^2_{0.99}$ metric, as a function of the noise level in the input data, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

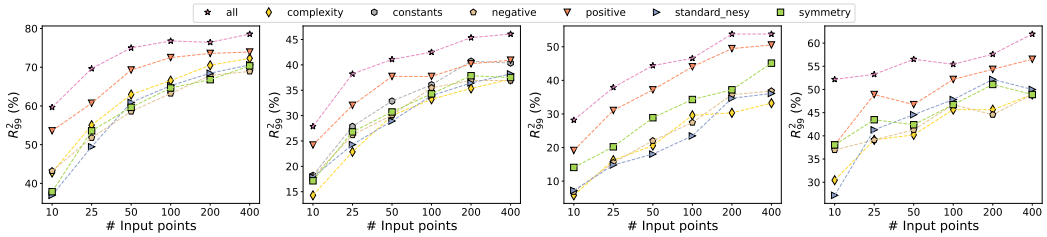

Figure 9. **Dependence on the number of input points.** Comparison between NSRwH conditioned with different types of hypotheses and the unconditioned baseline (`standard_nesy`) in terms of the $R^2_{0.99}$ metric, as a function of the number of input points, for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

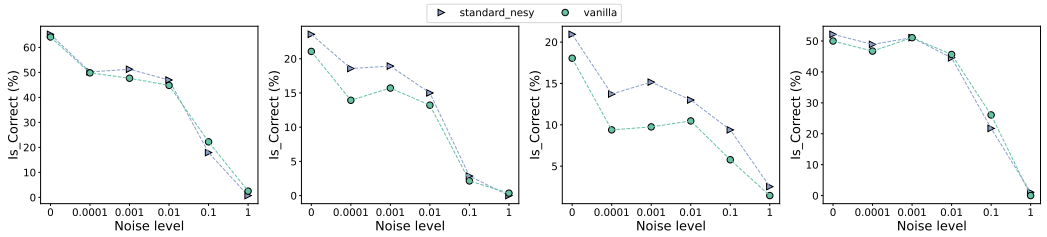

Figure 10. **Masked NSRwH vs. NeSymReS.** Comparison between fully masked NSRwH (`vanilla`) and standard NeSymReS (`standard_nesy`) for different noise levels for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.

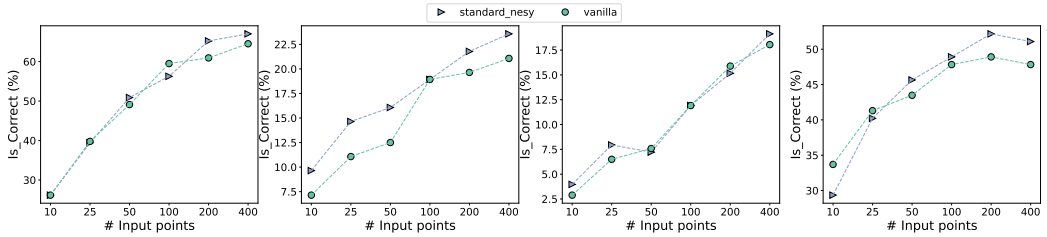

Figure 11. **Masked NSRwH vs. NeSymReS.** Comparison between fully masked NSRwH (`vanilla`) and standard NeSymReS (`standard_nesy`) for a different number of input points for the `train_nc`, `train_wc`, `only_five_variables_nc` and `AIF` datasets from left to right.