Privileged Deep Symbolic Regression

Tommaso Bendinelli*
CSEM SA, Alpnach
tommaso.bendinelli@csem.ch

Luca Biggio*
ETH Zürich
luca.biggio@inf.ethz.ch

Pierre-Alexandre Kamienny
Meta AI & Sorbonne Université, CNRS, ISIR
pakamienny@meta.com

Abstract

Symbolic regression is the process of finding an analytical expression that fits experimental data with the least amount of operators, variables and constants symbols. Given the huge combinatorial space of possible expressions, evolutionary algorithms struggle to find expressions that meet these criteria in a reasonable amount of time. To efficiently reduce the search space, neural symbolic regression algorithms have recently been proposed for their ability to identify patterns in the data and output analytical expressions in a single forward-pass. However, these new approaches to symbolic regression do not allow for the direct encoding of user-defined prior knowledge, a common scenario in natural sciences and engineering. In this work, we propose the first neural symbolic regression method that allows users to *explicitly* bias predictions towards expressions that satisfy a set of assumptions on the expected structure of the ground-truth expression. Our experiments show that our conditioned deep learning model outperforms its unconditioned counterpart in terms of accuracy while achieving control over the predicted expression structure.

1 Introduction

1.1 Symbolic regression and problem-specific prior knowledge

Symbolic Regression (SR) is a technique that searches over the space of analytical expressions $\mathcal{E}$ to fit experimental data while trading off between minimizing expression complexity and maximizing accuracy. As opposed to over-parametrized approaches, e.g. decision trees and neural networks, SR can produce human-readable expressions that are particularly useful in Natural Sciences, e.g. materials sciences (Wang et al., 2019; Kabliman et al., 2021; Ma et al., 2022), 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), to interpret physical phenomena. Rather than perfectly fitting the data, which are often noisy because of inevitable measurement errors, SR aims at acquiring meaningful insights on the inner mechanisms governing physical systems via compact, or equivalently low-complexity, expressions. La Cava et al. (2021) recent benchmarking effort has shown that symbolic regressors can additionally outperform their over-parametrized counterparts on a set of black-box and synthetic datasets.

In Natural Sciences, researchers generally have some insights on the datasets they study, using analogies to problems where a solution is known. For instance, when studying a system with different elements interacting with each other, a good assumption is that the laws of the system involve terms similar to Gravitational/Coulomb forces, e.g. $\propto \frac{1}{r^2}$ with $r = \sqrt{x^2 + y^2}$. Access to a part of the underlying ground-truth equation is also a common assumption done in the system identification

NeurIPS 2022 AI for Science Workshop.

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 name *hypotheses* the assumptions formulated by the SR practitioner, potentially incomplete or partially incorrect, about the underlying ground-truth expression that can be leveraged in any form to restrict the search space. If an hypothesis is true, we will name it *privileged information*.

1.2 Related work

Searching for a satisfactory analytical expression is a difficult optimization problem, mostly tackled using genetic programming (GP) algorithms. They work by i) defining a class of programs, represented in SR as tree structures where nodes are unary, e.g. `cos`, `exp` or binary operators `add`, `div` and leaves are variables and constants and ii) evolving a population of analytical expressions by means of selection, mutation and crossovers. Being greedy search approaches, GP algorithms are prone to fall in local minima, and extensive exploration leads to relatively large run-times. In practice with time limits, e.g. 24 hours in (La Cava et al., 2021), the most accurate GP methods provide expressions with overly large complexity, thus preventing deriving physical insights; on the Feynman datasets (Udrescu and Tegmark, 2020), whose functions have averaged complexity 20 as defined in (La Cava et al., 2021), the current state-of-the-art (Burlacu et al., 2020) predicts functions with averaged complexity 100. Up to our knowledge, injection of prior information in GPs can only be done by filtering during selection, e.g. using properties like function positivity or convexity (Kronberger et al., 2022; Haider et al., 2022). This strategy is greedy in essence, therefore leading to premature convergence to local minima expressions. Other forms of high-level prior information available to the user, e.g. complexity of the expected expression, can hardly be incorporated in GP. GP recently adopted neural networks for their ability to detect qualitative patterns in the data to reduce search space (Petersen et al., 2019; Mundhenk et al., 2021).

Our contribution draws inspiration from the works of Udrescu and Tegmark (2020) and Udrescu et al. (2020) where the search is restricted to expressions characterized by some specific properties, like compositionality, additivity and generalized symmetry. Making use of these properties effectively simplifies the task of SR by exploiting the modularity of the resulting expression trees. Despite being similar in spirit to our work, their method requires fitting a new neural network on each new input dataset and then probing the trained network in search of the aforementioned properties, a process resulting in an inevitably slow approach.

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). First, large synthetic datasets are generated by i) sampling expressions from a prior distribution $p_\theta(\mathcal{E})$ where θ is a parametrization induced by Lample and Charton (2019) generator, 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 functions, e.g complexity or sub-expression terms, boils down to a sound design of the generator however the pipeline introduced in (Lample and Charton, 2019) allows only limited degrees of freedom: operators, variables or 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 subclass $h(\mathcal{E}) \subset \mathcal{E}$ using privileged information, e.g. $h(\mathcal{E})$ can be the class of expressions with complexity between 10 and 20, or that have a specific sub-expression like $e^{-\sqrt{x_1^2+x_2^2}}$ in them. However, a trained NSR model $g_\phi(\mathcal{E}|\mathcal{D})$, can only be adapted to $h(\mathcal{E})$ by i) relying on rejection sampling, i.e. sampling from $g_\phi(\mathcal{E}|\mathcal{D})$ and discarding all expressions that do not satisfy its criterion, a time-inefficient technique that assumes that the model was trained on a very large class of analytical expressions, e.g. diverse set of operators/complexity, and ii) designing a new generator with desired properties, a tedious task, and fine-tuning the model on the produced datasets.

1.3 Contributions

In this work, we propose a solution to the above limitations of NSR algorithms, named *Neural Symbolic Regression with Hypotheses* (NSRwH). NSRwH efficiently restricts the class of predicted expressions of NSR models during inference, if provided privileged information $\mathcal{D}_{\mathrm{PI}}$ as described in 1.1, 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 $\mathcal{D}_{\mathrm{PI}}$, e.g. appearing operators or complexity, and feed this meta-data into the Transformer model as an extra input, i.e. $g_\phi(S|\mathcal{D}, \mathcal{D}_{\mathrm{PI}})$. During training, we use a masking strategy to avoid our model considering sub-classes of functions when no privileged information is provided.
We consider expressions with real constants with feature space $\mathbb{R}^5$, already a challenging setting as the induced search space is already extremely large (Lample and Charton, 2019).
To the best of our knowledge, the present is the first work that explores the injection of such prior knowledge into SR. We insist on the fact that naive adaptation of GP algorithms via filtering in the selection step is prone to premature convergence to local minina as mutations and crossovers break satisfaction of the properties we consider.

We show that our model manifests the following desirable characteristics:

1. Similarly to expression derivation and integration (Lample and Charton, 2019), we show that Transformers are able to capture non-trivial high-level symbolic expression properties such as complexity, an arguably harder task.
2. When conditioned on specific user-determined privileged information on the sought for expression, the model outputs expressions that closely adhere to the provided prior knowledge. This feature highlights that the proposed model is effectively *controllable* and its output reflects the expectations of the user in terms of the specified high-level properties. This is in stark contrast with prior work both in the NSR and GP literature, where steering symbolic regressors towards specific properties required either training from scratch or resorting to inefficient post-hoc greedy search routines.
3. Our conditioned model outperforms its unconditioned counterpart both in the presence or not of output noise, showing that prior knowledge about the sought for expression helps in alleviating the effect of noise, making SR feasible also in this unfavourable and previously unaddressed setting.

Overall, the proposed framework is general and provides a simple and general recipe to easily condition neural symbolic regressors on any sort of prior information the user deems relevant to restrict the search space over mathematical expressions.

2 Method

2.1 Notation and Background

A symbolic regressor is an algorithm that takes as input a dataset $\mathcal{D}$ of n features-value pairs $(\mathbf{x_i}, y_i) \sim \mathbb{R}^d \times \mathbb{R}$, with d is the feature dimension, and returns a symbolic expression $e \sim \mathcal{E}$ such that $\forall (\mathbf{x}_i, y_i) \in \mathcal{D},\ e(\mathbf{x}_i) = \tilde{y_i} \approx y_i$. SR literature usually considers the R^2 score, defined as below, as the metric to quantify satisfactory fitting levels:

$$R^2(y, \tilde{y}) = 1 - \frac{\sum_i^n (y_i - \tilde{y_i})^2}{\mathrm{Var}(y)} \tag{1}$$

Neural symbolic regression 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 $\mathcal{D}_{\mathrm{PI}}$, NSRwH approaches are trained to model the conditional distribution $g_\phi(\mathcal{E} \mid \mathcal{D}, \mathcal{D}_{\mathrm{PI}})$.

2.2 Data generation

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

Generating expressions. As in other NSR works (Biggio et al., 2021; Kamienny et al., 2022), we sample analytical expressions e using the strategy introduced by Lample and Charton (2019): random unary-binary trees with depth between 1 and 5 are generated, then internal nodes are assigned either unary or binary operators described in Table 3 in the Appendix B.1 according to their arity, and leaves are assigned variables $\{x_d\}_{d\leq 5}$.

Generating $\mathcal{D}$. For each expression from e, we sample a support of n points $\mathbf{x}_i \in \mathbb{R}^d$ where each coordinate is sampled independently from others within uniform distribution $\mathcal{U}$ whose bounds are sampled in $[-10, 10]$ as in (Biggio et al., 2021). Next, the expression value y_i is obtained via the evaluation of the expression e on the previously sampled support. Expressions with more than 10% of NaNs are discarded from the training set. We train our model with output noise drawn from centered Gaussian distribution with deviation

$$\sigma = \gamma \sqrt{\frac{1}{n} \sum_{i}^{n} y_i^2} \tag{2}$$

with $\gamma \in \{0, 1e^{-3}, 1e^{-2}, 1e^{-1}, 1\}$ as prescribed in (La Cava et al., 2021)

Generating $\mathcal{D}_{\mathrm{PI}}$. Privileged information $\mathcal{D}_{\mathrm{PI}}$ is composed of *properties*. From an expression e, we consider the following properties[1]:

1. *complexity bins*: We use the definition of complexity provided by Kommenda et al. (2015) that assigns a score that reflects semantic information of the expression and better aligns with the human intuition of what a complex expression looks like, contrarily to more standard metrics measuring complexity based on the length of the expression tree. Since the SR practioner generally has only a vague idea of the desired function, we bin the complexity into intervals, i.e. $[0, 5], \dots, [1000000, \infty]$, resulting in a total of 6 complexity levels. Note that the bins delimiters where chosen to have approximately the same counts on our training set of expressions.
2. *symmetries*: 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. Udrescu et al. (2020) provides an efficient recipe based on ∇f to assess if f is characterized by a generalized symmetry in every subset of its independent variables. We feed the network with a tensor of $2^d = 32$ components, equal to the number of all possible subset of variables, each indicating whether that subset possesses a generalized symmetry or not.
3. *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 $[+, x_1, +x_1, \sin, \sin x_2, x_2, +\sin, +\sin x_2]$. We then sample between 0 and 4 items from this set, with a weighted probability inversely proportional to their length.
4. *Absent branches:* We sample from 0 up to 4 branches which do not appear in the generating expression, drawn from the space of all branches that appear in expressions in our dataset. As for the previous property, we sample them with a probability inversely proportional to their length.

In our experiments, we explore whether our NSRwH model is able to capture the meaning of such properties and generate expressions that adhere to them. We give more details on the exact computation of each property in the Appendix A.

[1]This set of properties can be straightforwardly extended according to the user's prior knowledge.

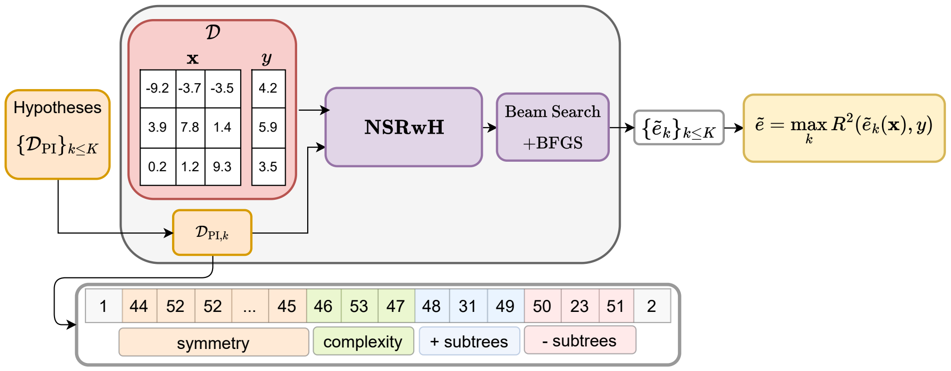

Figure 1: **Structure of our framework and example of privileged information.** The illustration shows an example of how NSRwH can be used for symbolic regression. First, a set of K hypotheses, $\{\mathcal{D}_{PI,k}\}_{k\leq K}$ is formulated. Then, each of them is given independently to the model, along with numerical data $\mathcal{D}$, and the output of the model is evaluated and compared to the ground-truth. The box below shows how privileged information is encoded in the form of a tensor of integers.

2.3 Model

In this section, we report the details of the architecture of our model as well as its training procedure and adaptation at test time.

Architecture. We use Nesymres (Biggio et al., 2021) as our base neural symbolic regressor for its simplicity and explain how we 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 other more advanced NSR architectures, such as the one introduced in (Valipour et al., 2021; Kamienny et al., 2022). Float numbers are tokenized using a multi-hot bit representation according to the half-precision IEEE-754 standard and expressions are tokenized using their polish notation.

The input dataset $\mathcal{D}$ of size n is encoded in $\mathbf{z}_\mathcal{D} \in \mathbb{R}^{n\times d_{\text{emb}}}$ by i) tokenizing floats and ii) passing these tokens through a set encoder (Lee et al., 2019), a variation of (Vaswani et al., 2017) with better inference time and less memory requirement. Positional embeddings are not included to obtain set permutation-invariant representations. We use an additional standard self-attention encoder (Vaswani et al., 2017), which we call *privileged encoder*, to embed privileged information $\mathcal{D}_{\text{PI}}$ into a sequence $\mathbf{z}_{\mathcal{D}_{\text{PI}}} \in \mathbb{R}^{m\times d_{\text{emb}}}$, where $m \neq n$. All properties are first independently tokenized using property tokens, e.g. complexity bin $[0, 5]$, or polish notation for subtree expressions. Then, we concatenate them using property separators, e.g. $< symmetries >$ or $< complexity >$, into a large sequence. The decoder uses self-attention as well as cross-attention on the tensor $\mathbf{z} \in \mathbb{R}^{(m+n)\times d_{\text{emb}}}$ obtained by the concatenation of $z_\mathcal{D}$ and $z_{\mathcal{D}_{\text{PI}}}$ and predicts the expression $\tilde{e}$ in an auto-regressive fashion following $g_\phi(\tilde{e}_{t+1}|\tilde{e}_{1:t}, \mathcal{D}, \mathcal{D}_{\text{PI}})$.

Training. 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 ground-truth e first tokens. As for Nesymres, we “skeletonize” target expressions by replacing constants by a constant token $\diamond$.
To avoid our model being dependent on privileged information at test time, we include training examples with partial privileged information $\mathcal{D}_{\text{PI}}$. Practically, we mask property tokens by using dropout (Srivastava et al., 2014): with probability p, a property changes its value to the $< PAD >$ token (also used for padding sequences of different lengths for batching). More details on the training hyperparameters can be found in Appendix B.

Testing. 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) to recover the values of the constants by minimizing the squared loss between the original outputs and the output from the predicted expressions. The test-time procedure is summarized on Fig. 1

3 Experiments

In this section, we evaluate NSRwH and compare to (Biggio et al., 2021), which we name NSR for the sake of clarity. First, we analyze whether transformers can capture the properties introduced in Sec. 2. In addition, we describe a number of simple examples to illustrate how NSRwH can be practically used to discover expressions given a set of prior assumptions by enhancing user interaction and facilitating hypothesis testing. Finally, we assess the extent to which extra conditioning helps to improve the test performance.

Following the procedure described in 2.2, we generated a set of $20M$ training expressions without duplicates using the Sympy (Meurer et al., 2017) simplification function. We trained our model on the training set using the procedure described in 2.3 on 8 P100 GPU for 2 days.

3.1 Can transformers efficiently restrict the inference space using descriptions?

We investigate the ability of transformers to "understand" the properties of analytical expressions introduced in 2. We give as an input to the NSRwH model a single hypothesis (i.e. $K = 1$ on Fig. 1) which is the privileged one. Our test set consists of 100 randomly-generated mathematical expressions.

3.1.1 Quantitative results

To validate that transformers can restrict the class of predicted expressions according to a particular input property, we introduce a metric $is\_satisfied(g_\phi, \mathcal{D}, property\_type)$ that measures the percentage of properties satisfied in the predicted expression for a given property type (averaged over 10 samples from g_ϕ). For instance, for the property type of appearing subtrees x_1, `sin`, $x_1^2 + x_2^2$, x_1/x_2 hypotheses, in predicted expression $f(x_1, x_2) = x_1 - \sin(x_1^2 + x_2^2)$, only 75% properties are respected since x_1/x_2 does not appear in the ground truth expression.

From the test set $\{(e, \mathcal{D}, \mathcal{D}_{\text{PI}})\}_{j \leq 100}$, we derive a dataset of descriptions and measure $is\_satisfied$ averaged over all 100 examples for each property type. To make the task more challenging, we corrupt the numerical data with zero-mean gaussian noise with standard deviation as in 2, with $\gamma = 10^{-2}$. This test is meant to assess the reliance of our conditioned model on the input description regardless of the noise in the input numerical data. We compare our model with NSR, i.e. a model receiving as input only numerical information, $\mathcal{D}$.

Model g_ϕ	$property\_type$		
	Complexity bin	Symmetries	Appearing subtrees
NSR Biggio et al. (2021) (with $\mathcal{D}$)	29.7	64.2	50.5
NSRwH (with $\mathcal{D}_{\text{PI}}$ and $\mathcal{D}$)	**82.1**	**95.1**	**95.4**

Table 1: $is\_satisfied(g_\phi, \mathcal{D}, property\_type)$ with $\mathcal{D}$ being a test set of 100 held-out expressions. Privileged information is captured by the model in the output expressions.

Overall, as shown in Table. 1, the proposed NSRwH efficiently reduces the inference space when conditioning on a set of hypotheses and outputs predictions that are well-aligned with expectations. We conclude that not only does the model "understands" the meaning of the input conditioning, but it does so also in the presence of a significant amount of noise in the input numerical data, hence manifesting a high-level of controllability.

3.1.2 Qualitative results

In this section, we qualitatively investigate the behaviour of our conditioned model NSRwH via several examples where it is provided with different information about the ground-truth expression.

Role of Generalized Symmetry. Consider the following expression of five independent variables:

$$y = \cos(x_1 + x_2 x_3 + x_4 x_5) \tag{3}$$

When all the privileged information is masked, the model is not able to infer the ground truth expression up to a beam-size of 50, despite capturing its sinusoidal nature in all the output predictions.

When five variables are considered the search space is inevitably very large and any sort of prior knowledge can be precious to restrict it to a smaller subset. It is in this high-dimensional regime that information about generalized symmetry is important. We note that 3 possesses non-trivial generalized symmetries in the variable tuples (x_2, x_3) and (x_4, x_5), (x_1, x_2, x_3) and (x_1, x_4, x_5). Given *only* this high-level information as input, the model is able to recover the true expression among the first 20 output expressions.

Role of complexity. Let's consider the following expression of three independent variables:

$$y = \sin(\cos(x_1 + x_2)) + x_3 \tag{4}$$

Given numerical inputs only, NSRwH does not recover the true expression up to a beam-size of 50, despite capturing its sinusoidal nature and the linear dependence on x_3. What makes the task hard is the complex compositional nature of the first term involving both $\sin$ and $\cos$ operators. In order to bias the model towards more complex formulas, we condition it on a relatively high complexity bin, namely the fourth one. By giving the model this additional information, the model's predictions strictly adhere with the conditioning, giving the correct formula at beam 26.

Role of appearing subtrees. Let's consider the following expression of three independent variables:

$$y = x_1 x_2 x_3 + 10^{-3} \tan(x_3) \tag{5}$$

The second term in the expression above is very difficult to detect precisely, due to the small multiplying factor making its contribution weak. The fully-masked model does not recover the true expression, yet catching the first term consistently. By providing the model with the additional information of the presence of the unary operator $\tan$, NSRwH finds the true expression at beam 20.

3.2 Does extra-conditioning help improve test accuracy?

Model g_ϕ	Mask probability p	Noise level γ			
		$\gamma = 0$	$\gamma = 1e^{-3}$	$\gamma = 1e^{-2}$	$\gamma = 1$
NSRwH	1.0 (NSR)	0.403510	0.39850	0.36341	0.13033
	0.75	0.43610	0.41103	0.40602	0.15790
	0.5	0.44110	0.42356	0.38095	0.17293
	0.25	**0.45865**	0.44361	**0.43860**	0.19298
	0.0	0.448622	**0.44862**	0.43609	**0.21303**

Table 2: Accuracy ($R^2 > 0.99$) of NSR and NSRwH models on a set of 100 test expressions under different noise level γ and masking probabilities p. Results averaged over 5 different seeds.

We investigate if extra-conditioning in the form of privileged information results in improved accuracy and robustness to noise. To this purpose, we utilize a set of 100 randomly-generated expressions, we calculate the R^2 score, as defined in 1, on a set of support points sampled outside the distribution of the input ones, i.e. in the extrapolation regime. An equation is correctly classified if $R^2 > 0.99$.
As shown in Table 3.2 the model's performance improves when leveraging privileged information. In particular, we make the following observations: 1) The smaller the masking probability, the larger the accuracy[2] 2) Conditioning is especially beneficial in the cases where noise is applied, leading to substantial improvements with respect to the fully-masked model.

4 Discussion

Conclusive remarks. This work presents a novel approach for symbolic regression that enables the explicit incorporation of inductive biases reflecting prior knowledge on the problem at-hand. In stark 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, an highly desirable feature for scientific discovery applications. We demonstrate the value of this approach with a number of examples and preliminary experiments where numerical data is sparse or affected by noise.

[2]This is occasionally violated since, depending on the random seed, the specific appearing sub-trees change, and some of them can be more useful than other for the model.

Future work and outlook. We plan to extend the present research by further evaluating NSRwH on more datasets and benchmarks. We emphasize that the current version of our model can be improved from multiple angles. First and foremost, due to limited computational resources, NSRwH is not comparable with modern language models in terms of size. Scaling NSRwH to such regimes represents an exciting avenue for future research and for fully unveiling the real capabilities of NSR. Second, additional properties can possibly be included in the privileged information conditioning according to the user's prior knowledge of the data under consideration. Third, our framework can be easily incorporated into more advanced NSR architectures, like Kamienny et al. (2022), likely resulting in further performance improvements. In conclusion, we believe that the method presented in this work will enable a wider application of symbolic regression, resulting in more effective and user-centric methods.

Impact statement

Symbolic Regression is a growing field and its combination with neural networks has witnessed significant interest in the last few years. We believe that this new direction represents a significant step towards the adoption of symbolic regression methods as useful tools to support the scientific discovery process. We do not foresee any ethical concerns with this work.

References

Biggio, L., Bendinelli, T., Lucchi, A., and Parascandolo, G. (2020). A seq2seq approach to symbolic regression. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. (2021). Neural symbolic regression that scales.

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

Burlacu, B., Kronberger, G., and Kommenda, M. (2020). 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. Association for Computing Machinery.

Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. (2020). Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33:17429–17442.

d'Ascoli, S., Kamienny, P.-A., Lample, G., and Charton, F. (2022). Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*.

Haider, C., de França, F. O., Kronberger, G., and Burlacu, B. (2022). Comparing optimistic and pessimistic constraint evaluation in shape-constrained symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 938–945.

Hernandez, A., Balasubramanian, A., Yuan, F., Mason, S. A., and Mueller, T. (2019). Fast, accurate, and transferable many-body interatomic potentials by symbolic regression. *npj Computational Materials*, 5(1):1–11.

Kabliman, E., Kolody, A. H., Kronsteiner, J., Kommenda, M., and Kronberger, G. (2021). Application of symbolic regression for constitutive modeling of plastic deformation. *Applications in Engineering Science*, 6:100052.

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

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. (2022). End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*.

Kommenda, M., Beham, A., Affenzeller, M., and Kronberger, G. (2015). Complexity measures for multi-objective symbolic regression. In *Computer Aided Systems Theory – EUROCAST 2015*, pages 409–416. Springer International Publishing.

Kronberger, G., de França, F. O., Burlacu, B., Haider, C., and Kommenda, M. (2022). Shape-constrained symbolic regression—improving extrapolation with prior knowledge. *Evolutionary Computation*, 30(1):75–98.

La Cava, W., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. (2021). Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*.

Lample, G. and Charton, F. (2019). Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR.

Li, J., Yuan, Y., and Shen, H. (2022). Symbolic expression transformer: A computer vision approach for symbolic regression. *ArXiv*, abs/2205.11798.

Ma, H., Narayanaswamy, A., Riley, P., and Li, L. (2022). Evolving symbolic density functionals. *arXiv preprint arXiv:2203.02540*.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. (2017). Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., and Petersen, B. K. (2021). Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. (2019). Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*.

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

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sun, F., Liu, Y., Wang, J.-X., and Sun, H. (2022). Symbolic physics learner: Discovering governing equations via monte carlo tree search. *arXiv preprint arXiv:2205.13134*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. (2020). Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity.

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

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

Valipour, M., You, B., Panju, M., and Ghodsi, A. (2021). Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*.

Vastl, M., Kulhánek, J., Kubalík, J., Derner, E., and Babuška, R. (2022). Symformer: End-to-end symbolic regression using transformer-based architecture. *ArXiv*, abs/2205.15764.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

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

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We intend to make the final version open source. The current version is available upon reasonable request
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Section B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Details on properties

A.1 Generalized 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. 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. Thanks to this property, given a symbolic expression, we can rapidly extract its generalized symmetries on every subset of variables. Since in this work we limit ourselves to 5 independent variables, for each expression we have a 32 dimensional tensor encoding whether a symmetry is present for every subset of variables. Ultimately, we exclude from this vector some trivial and non-informative components, i.e. subset of only one-variable, the full-variable subset and the empty set, resulting in a 25-dimensional tensor.

A.2 Appearing / absent branches

In order to sample branches from a prefix tree, we adopt the following procedure: first, we enumerate all possible branches of the tree using a Depth-First-Search (DFS) algorithm. We pre-compute the set of all possible branches for absent trees offline from a subset of the training data. In particular, for the experiments, we use a subset of 10,000 trees. Then we compute the disjoint set between this set and the current prefix tree and sample from there. For both appearing and absent subtrees, branch elements are sampled with a probability inverse to their length during both training and experimental evaluation. Once sampled, the elements are sorted alphanumerically so that their order is consistent during training and evaluation.

A.3 Complexity

We follow the definition of complexity introduced in Kommenda et al. (2015), with only some slight modifications. Starting at the root node, the complexity of an expression tree can be calculated by applying the equation below recursively:

$$\text{Complexity }(n) = \begin{cases} 1 & \text{if } n \equiv \text{ constant} \\ 2 & \text{if } n \equiv \text{ variable} \\ \sum \text{ Complexity }(c) & \text{if } n \in (+,-) \\ \prod \text{ Complexity }(c) + 1 & \text{if } n \in (*,/) \\ \text{Complexity }(c)^2 & \text{if } n \equiv \text{ square} \\ \text{Complexity }(c)^2 & \text{if } n \equiv \text{ squareroot} \\ 2^{\text{Complexity }(c)} & \text{if } n \in (\sin, \cos, \tan, \exp, \log) \end{cases}$$

After obtaining the complexity score associated with an expression, we assign it a bin level according with the following rule:

$$\text{Complexity\_level }(n) = \begin{cases} 0 & \text{if Complexity }(n) \in [0,5] \\ 1 & \text{if Complexity }(n) \in (5,100] \\ 2 & \text{if Complexity }(n) \in (100,1000] \\ 3 & \text{if Complexity }(n) \in (1000,10000] \\ 4 & \text{if Complexity }(n) \in (10000,1000000] \\ 5 & \text{if Complexity }(n) > 1000000 \end{cases}$$

B Training details

B.1 Dataset generation

We sample analytical expressions e using the strategy introduced by Lample and Charton (2019): random unary-binary trees with depth between 1 and 5 are generated, then internal nodes are assigned either unary or binary operators described in Table 3 according to their arity, and leaves are assigned variables $\{x_d\}_{d\leq 5}$

Arity	Operators
Unary	`sqrt, pow2, pow3, pow4` `inv, log, exp,` `sin, cos, tan, atan`
Binary	`add, sub, mul`

Table 3: Operators used in our generators.

B.2 Training hyper-parameters

Set encoder. We embed input datasets with set encoder Lee et al. (2019) using 5 Induced Set Attention Blocks (ISAB) with 512 hidden dimensions, 8 heads, 10 Pooling by Multihead Attention (PMA) features and 50 inducing points.

Privileged encoder. We use the classic transformer encoder Vaswani et al. (2017) using 5 layers with 512 hidden dimensions, 8 heads and embedding dimensions 32.

Decoder. We use the classic transformer decoder using 5 layers with 512 hidden dimensions, 8 heads and embedding dimensions 32.

NN Optimizer. We use Adam with batch size 32, learning rate 10^{-4} and weight decay 10^{-3}.

Numerical constant optimizer. In all quantitative experiments, we use the BFGS optimizer with batch size of 32 and 2 restarts.