# Neural Symbolic Regression that Scales

**Luca Biggio** [*1 2] **Tommaso Bendinelli** [*2] **Alexander Neitz** [3] **Aurelien Lucchi** [1] **Giambattista Parascandolo** [1 3]

## Abstract

Symbolic equations are at the core of scientific discovery. The task of discovering the underlying equation from a set of input-output pairs is called symbolic regression. Traditionally, symbolic regression methods use hand-designed strategies that do not improve with experience. In this paper, we introduce the first symbolic regression method that leverages large scale pre-training. We procedurally generate an unbounded set of equations, and simultaneously pre-train a Transformer to predict the symbolic equation from a corresponding set of input-output-pairs. At test time, we query the model on a new set of points and use its output to guide the search for the equation. We show empirically that this approach can re-discover a set of well-known physical equations, and that it improves over time with more data and compute.

## 1. Introduction

Since the early ages of Natural Sciences in the sixteenth century, the process of scientific discovery has rooted in the formalization of novel insights and intuitions about the natural world into compact symbolic representations of such new acquired knowledge, namely, mathematical equations.

Mathematical equations encode both objective descriptions of experimental data and our inductive biases about the regularity we attribute to natural phenomena. When seen under the perspective of modern machine learning, they present a number of appealing properties: (i) They provide *compressed* and *explainable* representations of complex phenomena. (ii) They allow to easily incorporate prior knowledge. (iii) When relevant aspects about the data generating process are captured, they often generalize well beyond the distribution of the observations from which they were derived.

[*]Equal contribution [1]Department of Computer Science, ETH, Zürich, Switzerland [2]CSEM SA, Alpnach, Switzerland [3]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Luca Biggio <luca.biggio@inf.ethz.ch>, Tommaso Bendinelli <tommaso.bendinelli@csem.ch>.

*Proceedings of the $38^{th}$ International Conference on Machine Learning*, PMLR 139, 2021. 

The process of discovering symbolic expressions from experimental data is hard and has traditionally been one of the hallmarks of human intelligence. *Symbolic regression* is a branch of regression analysis that tries to emulate such a process. More formally, given a set of $n$ input-output pairs $\{(x_i, y_i)\}_{i=1}^{n} \sim \mathcal{X} \times \mathcal{Y}$, the goal is to find a symbolic equation $e$ and corresponding function $f_e$ such that $y \approx f_e(x)$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. In other words, the goal of symbolic regression is to infer both model structure and model parameters in a data-driven fashion. Even assuming that the vocabulary of primitives — e.g. $\{\sin, \exp, +, ...\}$ — is sufficient to express the *correct* equation behind the observed data, symbolic regression is a hard problem to tackle. The number of functions associated with a string of symbols grows exponentially with the string length, and the presence of numeric constants further exacerbates its difficulty.

Due to its challenging combinatorial nature, existing approaches to symbolic regression are mainly based on search-techniques whose goal is typically to minimize a pre-specified fitness function measuring the distance between the predicted expression and the available data. The two main drawbacks of such methods are that: (i) *They do not improve with experience.* As every equation is regressed from scratch, the system does not improve if access to more data from different equations is given. (ii) *The inductive bias is opaque.* It is difficult for the user to steer the prior towards a specific class of equations (e.g. polynomials, etc.). In other words, even though most symbolic regression algorithms generate their prediction starting from a fixed set of primitives reflecting the user's prior knowledge, such elementary building blocks can be combined in many arbitrary ways, providing little control over the equation distribution. To overcome both drawbacks, in this paper we take a step back, and let the model *learn the task* of symbolic regression over time, on a user-defined prior over equations.

Building on the recent successes of large models trained on large datasets (Brown et al., 2020; Devlin et al., 2018; Chen et al., 2020a;b), we show that a strong symbolic regressor can be purely learned from data. The key factor behind our approach is that computers can generate unbounded amounts of data with perfect accuracy and at virtually no cost. The distribution over equations used during pre-training strongly influences the prior over equations of the final system. Such a prior thus becomes easy to understand and control.

The main contributions of this paper are the following:

- We introduce a simple, flexible, and powerful framework for symbolic regression, the first approach (to the best of our knowledge) to improve over time with data and compute.
- We demonstrate that *learning the task* of symbolic regression from data is sufficient to significantly outperform state-of-the-art approaches relying on hand-designed strategies.
- We release our code and largest pre-trained model [1]

In Section 2, we detail related work in the literature. In Section 3, we present our algorithm for neural symbolic regression that scales. We evaluate the method in the experiments described in Section 4 and 5 and compare it to state-of-the-art baselines. In Section 6 we discuss results, limitations, and potential for future work.

## 2. Related Work

**Genetic Programming for Symbolic Regression** Traditional approaches to symbolic regression are based on genetic algorithms (Forrest, 1993) and, in particular, genetic programming (GP) (Koza, 1994). GP methods used for symbolic regression iteratively "evolve" a population of candidate mathematical expressions via mutation and recombination. The most popular GP-based technique applied to symbolic regression is undoubtedly the commercial software Eureqa (Dubčáková, 2011) which is based on the approach proposed by Schmidt & Lipson (2009). Despite having shown for the first time the potential of data-driven approaches to the problem of function discovery, GP-based techniques do not scale well to high dimensional problems and are highly sensitive to hyperparameters (Petersen, 2021).

**Neural Networks for Symbolic Regression** A more recent line of research explores the potential of deep neural networks to tackle the combinatorial challenge of symbolic regression. Martius & Lampert (2016) propose a simple fully-connected neural network where standard activation functions are replaced with symbolic building blocks (e.g. "$\sin(\cdot)$", "$\cos(\cdot)$", "$+$", "Identity$(\cdot)$"). Once the model is trained, a symbolic formula can be automatically read off from the network architecture and weights. This method inherits the ability of neural networks to deal with high-dimensional data and scales well with the number of input-output pairs. However, it requires specific extensions (Sahoo et al., 2018) to deal with functions involving divisions between elementary building blocks (e.g. $\frac{sin(x)}{x^2}$) and the inclusion of exponential and logarithmic activations result in exploding gradients and numerical issues.

[1] https://github.com/SymposiumOrganization/NeuralSymbolicRegressionThatScales

Another approach to circumvent the discrete combinatorial search inherent in the symbolic regression framework is proposed in (Kusner et al., 2017). Here, a variational autoencoder (Kingma & Welling, 2013) is first trained to reconstruct symbolic expressions and the search for the best fitting function is then performed over the latent space in a subsequent step. While the idea of moving the search for the best expression from a discrete space to a continuous one is interesting and has been exploited by other approaches (e.g. (Alaa & van der Schaar, 2019)), the method does not prove to be effective in recovering relatively simple symbolic formulas. More recently, Petersen (2021) developed a new technique where a recurrent neural network (RNN) is used to model a probability distribution over the space of mathematical expressions. Output expressions contain symbolic placeholders to indicate the presence of numerical constants. Such constants are then fit in a second stage by an out-of-the-box nonlinear optimizer. The RNN is trained by minimizing a risk-seeking RL objective that assigns a larger reward to the top-epsilon samples from the output distribution. The method represents a significant step forward in the application of deep learning to symbolic regression. While showing promising results, the network has to be retrained from scratch for each new equation and the RNN is never directly conditioned on the data it is trained to model.

Finally, neural networks can also be used in combination with existing techniques or hand-designed rules to perform symbolic regression. Notable examples are (Udrescu & Tegmark, 2020; Udrescu et al., 2020), where neural networks are employed to identify simplifying properties in the data such as additive separability and compositionality. These properties are exploited to recursively simplify the original dataset into less challenging sub-problems that can be tackled by a symbolic regression technique of choice. A similar rationale is followed in (Cranmer et al., 2020), where different components of a trained Graph Neural Network (GNN) are independently fit by a symbolic regression algorithm. By joining the so-found expressions, a final algebraic formula describing the network can be obtained. The aforementioned approaches might provide very good performances when it is known a priori whether the data are characterized by specific structural properties, such as symmetries or invariances. However, when such information is not accessible, more domain-agnostic methods are required.

**Large Scale Pre-training** Our approach builds upon a large body of work emphasizing the benefits of pre-training large models on large datasets (Kaplan et al., 2020; Devlin et al., 2018; Brown et al., 2020; Chen et al., 2020a;b; Belkin et al., 2019). Examples of such models can be found in Computer Vision (Radford et al., 2021; Chen et al., 2020a;b; Kolesnikov et al., 2020; Oord et al., 2018) and Natural Language Processing (Devlin et al., 2018; Brown et al., 2020). There have also been recent applications of Transformers

(Vaswani et al., 2017) to tasks involving symbolic mathematics manipulations (Lample & Charton, 2019; Saxton et al., 2019) and automated theorem proving (Polu & Sutskever, 2020). Our work builds on the results from Lample & Charton (2019), where Transformers are trained to successfully perform challenging mathematical tasks such as symbolic integration and solving differential equations. However, our setting presents the additional challenge of mapping *numerical values* to the corresponding symbolic formula, instead of working exclusively within the symbolic domain.

## 3. Neural Symbolic Regression that Scales

A symbolic regressor $S$ is an algorithm which takes a set of $n$ input-output pairs $\{(x_i, y_i)\}_{i=1}^n \sim \mathcal{X} \times \mathcal{Y}$ as input and returns a symbolic equation $e$ representing a function $f_e$ such that: $y \approx f_e(x), \forall (x, y) \in \mathcal{X} \times \mathcal{Y}$. In this section, we describe our framework to *learn* a parametrized symbolic regressor $S_\theta$ from a large number of training data.

### 3.1. Pre-training

We pre-train a Transformer on hundreds of millions of equations which are procedurally generated for every minibatch. As equations and datapoints can be generated quickly and in any amount using a computer and standard math libraries, we can train the network end-to-end to predict the equations on a dataset that is potentially unbounded. We describe the exact process we use to generate the dataset in Section 4. An illustration of the main steps involved in the pre-training phase is shown in Fig. 1.

**Data** During the pre-training phase, each training example consists of a symbolic equation $e$ which represents a function $f_e : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$, a set of $n$ input points $X = \{x_i\}_{i=1}^n$ and corresponding outputs $Y = \{f_e(x_i)\}_{i=1}^n$. The distribution, $\mathcal{P}_{e,X}$, from which $e$ and the inputs $X$ are sampled will determine the inductive bias of the trained symbolic regressor and should be chosen to resemble the application domain. In particular, $X$ can vary in size (i.e. $n$ is not fixed), and the individual inputs $x_i$ do not have to to be *i.i.d* – neither within $X$ nor across examples or batches. For example, $\mathcal{P}_{e,X}$ could be polynomials of degree up to 6, and input sets of up to 100 points sampled uniformly from the range [0, 1]. In our experiments, an equation $e$ is represented by a sequence of symbols in prefix notation. An equation $e$ can contain numerical constants that are re-sampled at each batch to increase the diversity of the data seen by the model. In Section 4, we describe the details of the data generation process we used in our experiments.

**Pre-training** We train a parametric set-to-sequence model $S_\theta$ to predict the equation $e$ from the set of input-output points $X, Y$. In our implementation, $S_\theta$ consists of an encoder and a decoder. The encoder maps the $(x, y)$ sequence pairs for each equation into a latent space, resulting in a fixed-size latent representation $z$. A decoder generates a sequence $\bar{e}$ given $z$: it produces a probability distribution $P(\bar{e}_{k+1}|\bar{e}_{1:k}, z)$ over each symbol, given the previous symbols and $z$. The alphabet of $\bar{e}$ is identical to the one used for the original equations $e$, with one exception: unlike $e$, $\bar{e}$ does not contain any numerical constants. Instead, it contains a special placeholder symbol '$\diamond$' which denotes the presence of a constant which will be fit at a later stage. For example, if $e = 4.2\sin(0.3x_1) + x_2$, then $\bar{e} = \diamond\sin(\diamond x_1) + x_2$. We refer to the equation where numerical constants are replaced by placeholders as the "skeleton" of the equation, and use the notation $\bar{e}$ to refer to the symbolic equation that replaces numerical constants with '$\diamond$'. The model is trained to reduce the average loss between the predicted $\hat{e}$ and $\text{skeleton}(e)$, i.e. the skeleton of the original equation. Training is performed with mini-batches of $B$ equations each. The overall pre-training algorithm is reported in Algorithm 1.

**Algorithm 1** Neural Symbolic Regression pre-training

**Require:** $S_\theta$, batch size $B$, training distribution $\mathcal{P}_{e,X}$
  **while** not timeout **do**
    $L \leftarrow 0$
    **for** $i$ in $\{1..B\}$ **do**
      $e, X \leftarrow$ sample an equation and input set from $\mathcal{P}_{e,X}$
      $Y \leftarrow \{f_e(x) |\ x \in X\}$
      $\bar{e} \leftarrow \text{skeleton}(e)$
      $L \leftarrow L - \sum_k \log P_{S_\theta}(\bar{e}_{k+1}|\bar{e}_{1:k}, X, Y)$
    **end for**
    Compute the gradient $\nabla_\theta L$ and use it to update $\theta$.
  **end while**

### 3.2. Test time

At test time, given a set of input-output pairs $\{(x_i, y_i)\}_i$ we encode them using the encoder into a latent vector $z$. From $z$ we iteratively sample candidates skeletons of symbolic equations $\hat{\bar{e}}$ from the decoder. Finally, for each candidate, we fit the numerical constants $\diamond$ by treating each occurrence as an independent parameter. This can be achieved using a non-linear optimizer, either gradient-based or black-box, by minimizing a loss between the resulting equation applied to the inputs and the targets $Y$. In our experiments, we used beam-search to sample high-likelihood equation candidates from the decoder, and, like Petersen (2021), BFGS (Fletcher, 1987) on the mean squared error to fit the constants.

## 4. Experimental Set-up

Here, we present the instantiation of the framework described in Section 3 that we evaluate empirically, and detail the baselines and datasets used to test it. For the rest of the paper, we will refer to our implementation as NeSymReS[2].

[2]For **Ne**ural **Sym**bolic **Re**gression that **S**cales

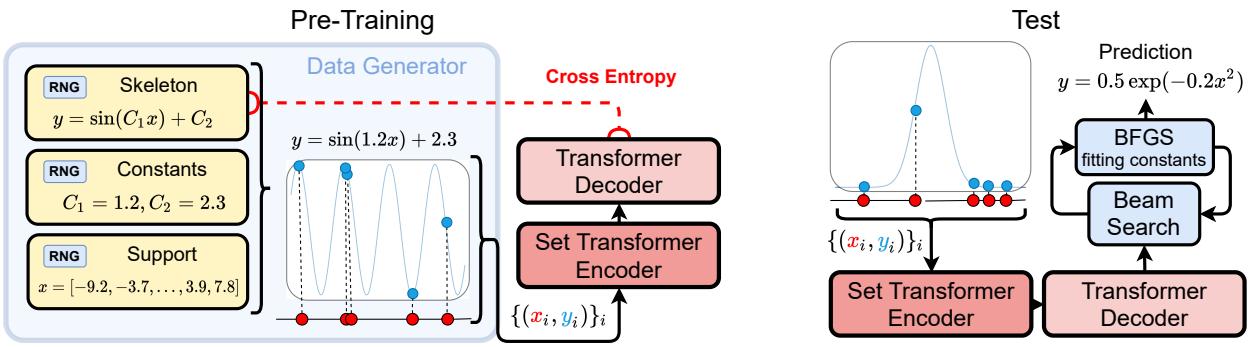


*Figure 1.* (Left) The data generator produces the input for the Transformer and its target expression. It does so by randomly sampling (i) an equation skeleton (including placeholders for the constants), (ii) numerical constants used to replace the placeholders and (iii) a set of support points $\{x_i\}_i$ to evaluate the previously generated equation and get the corresponding $\{y_i\}_i$. The $\{(x_i, y_i)\}_i$ pairs are fed into the Transformer, which is trained to minimize the cross-entropy loss with the ground-truth skeleton without numerical constants. Both the model output and the targets are expressed in prefix notation. (Right) At test time, given new input data, we sample candidate symbolic skeletons from the model using beam-search. The final candidate equations are obtained by fitting the constants with BFGS.

### 4.1. The Model $S_\theta$

For the encoder we opted for the Set Transformer architecture from Lee et al. (2019), using the original publicly available implementation.[3] We preferred this to the standard Transformer encoder, as the number $n$ of input-output pairs can grow to large values, and the computation in Set Transformers scales as $\mathcal{O}(nm)$ instead of $\mathcal{O}(n^2)$, where $m \ll n$ is a set of learnable inducing points (Snelson & Ghahramani; Titsias, 2009) we keep constant at $m = 50$. For the decoder we opted for a regular Transformer decoder (Vaswani et al., 2017), using the default PyTorch implementation. Encoder and decoder have 11 and 13 million parameters respectively. The hyperparameters chosen for both networks — detailed in Section A — *were not* fine-tuned for maximum performance.

### 4.2. Pre-training Data Generator

We sample expressions following the framework introduced in (Lample & Charton, 2019). A mathematical expression is regarded as a unary-binary tree where nodes are operators and leaves are independent variables or constants. Once an expression is sampled, it is simplified using the rules built in the symbolic manipulation library SymPy (Meurer et al., 2017). This sampling method allows us to precisely constrain the search space by controlling the depth of the trees and the set of admissible operators, along with their prior probability of occurring in the generated expression. We opted for scalar functions of up to three independent input variables (i.e. $d_x = 3$ and $d_y = 1$). For convenience, we pre-sampled 10 million skeletons of equations with up to three numerical constants each. At training time, we sample mini-batches of size $B = 150$ of the following elements:

*Equation skeletons* with constant placeholders placed randomly inside the expressions.

*Constants values* $C_1, C_2, C_3$, each independently sampled from a uniform distribution $\mathcal{U}(1, 5)$.

*Support extrema* $S_{1,j}, S_{2,j}$, with $S_{1,j} < S_{2,j}$ uniformly sampled from $\mathcal{U}(-10, 10)$ independently for each dimension $j = 1, \ldots, d_x$.

*Input points* for each input dimension $j = 1, \ldots, d_x$. A set of $n$ input points, $X_j = \{x_{i,j}\}_{i=1}^n$, is uniformly sampled from $\mathcal{U}(S_{1,j}, S_{2,j}, n)$ .

We then evaluate the equations on the input points $X = \{x_i\}_{i=1}^n$ to obtain the corresponding outputs $Y$.

As $Y$ can take very large or very small values, this can result in numerical instabilities and exploding or vanishing gradients during training. Therefore, we convert every $x_i$ and $y_i$ from float to a multi-hot bit representation according to the half-precision IEEE-754 standard. Furthermore, in order to avoid invalid operations (i.e dividing by zero, or taking the logarithm of negative values), we drop out input-output pairs containing NaNs.

We train the encoder and decoder jointly to minimize the cross-entropy loss between the ground truth skeleton and the skeleton predicted by the decoder as a regular language model. We use Adam with a learning rate of $10^{-4}$, no schedules, and train for $1.5M$ steps. Overall, this results in about $225M$ distinct equations seen during pre-training. See Appendix B for more details about training and resulting training curves.

[3] https://github.com/juho-lee/set_transformer

### 4.3. Symbolic Regression at Test Time

Given a set of input-output pairs from an unknown equation $e$, we feed the points into the encoder and use beam-search to sample candidate skeletons from the decoder. We then use BFGS to recover the values of the constants, by minimizing the squared loss between the original outputs and the output from the predicted equations. Our default parameters at test time are beam-size 32, with 4 restarts of BFGS per equation. We select the best equation from the set of resulting candidates based on the in-sample loss with a small penalty of 1e-14 per token of the skeleton.[4]

### 4.4. Evaluation

We evaluate our trained model on five datasets. Unless otherwise specified, for all equations we sample 128 points at test time.

**AI-Feynman (AIF)** First, we consider all the equations with up to 3 independent variables from the AI-Feynman (AIF) database (Udrescu & Tegmark, 2020) [5]. The resulting dataset consists of 52 equations extracted from the popular *Feynman Lectures on Physics* series. We checked our pre-training dataset, and amongst the 10 million equation skeletons, all equations from AIF appear. However, as mentioned in the previous subsection, the support on which they are evaluated, along with the constants and number of points per equation, is continuously sampled at every training iteration, making it impossible to exactly see any of the test data at training time.

**Unseen Skeletons (SOOSE)** This dataset of 200 equations is specifically constructed to have zero overlap with the pre-training set, meaning that its equations are all symbolically and numerically different from those included in the pre-training set. We call it SOOSE, for strictly out-of-sample equations. Compared to AIF, these equations are on average significantly longer and more complex (see Table 9). The sampling distribution for the skeletons is the same as the pre-training distribution, but we instantiate three different versions: with up to three constants (same as pre-training distribution, SOOSE-WC); no constants (SOOSE-NC); constants everywhere (SOOSE-FC, for full constants), i.e. one constant term for each term in the equation. The latter is extremely challenging, and since NeSymReS was only pre-trained with up to three constants, it is far from its pre-training distribution.

**Nguyen Dataset** This dataset consists of 12 simple equations *without* constants beyond the scalars 1 and 2, each with up to 2 independent variables. Nguyen was the main benchmark used in (Petersen, 2021). There are terms that appear in three ground truth equations that are *not* included in the set of equations that our model can fit, specifically $x^6$, and $x^y$, which therefore caps the maximum accuracy that can be reached by our model on this dataset.

### 4.5. Baselines

We compare the performance of our method with the following baselines:

**Deep Symbolic Regression (DSR)** (Petersen, 2021) Recently proposed RNN-based reinforcement learning search strategy for symbolic regression. We use the open-source implementation provided by the authors[6], with the setting that includes the estimation of numerical constants in the final predicted equation.

**Genetic Programming** (Koza, 1994) Standard GP-based symbolic regression based on the open-source Python library `gplearn` [7].

**Gaussian Processes** (Rasmussen, 2003) Standard Gaussian Process regression with RBF and constant kernel. We use the open source `sklearn` implementation[8].

All details about baselines are reported in Appendix A.

Two notable exclusions are AIF (Udrescu & Tegmark, 2020) and EQL (Martius & Lampert, 2016). As also noted by Petersen (2021), in cases where real numerical constants are present or the equations are not separable, the former still requires a complementary symbolic regression method to cope with the discrete search. The latter lacks too many basis functions that appear in the datasets we consider, preventing it from recovering most of the equations. Moreover, its average runtime and number of points required to solve the equations indicated in (Martius & Lampert, 2016; Sahoo et al., 2018) are three orders of magnitudes higher than the standards reported by the aforementioned baselines.

### 4.6. Metrics

Evaluating whether two equations are equivalent is a challenging task in the presence of real valued constants.

We distinguish between accuracy *within* the training support ($A^{\text{iid}}$), and outside of the training support ($A^{\text{ood}}$). $A^{\text{iid}}$ is computed with 10k points sampled uniformly in the training support. $A^{\text{ood}}$ is computed with 10k points in an extended support as detailed in Appendix B, and it will be the main metric of interest.

[4] While we found this strategy to work well in practice, a validation set for model selection might offer better performances with noisy data.

[5] https://space.mit.edu/home/tegmark/aifeynman.html

[6] https://github.com/brendenpetersen/deep-symbolic-regression

[7] https://gplearn.readthedocs.io/en/stable/

[8] https://scikit-learn.org/stable/modules/gaussian_process.html

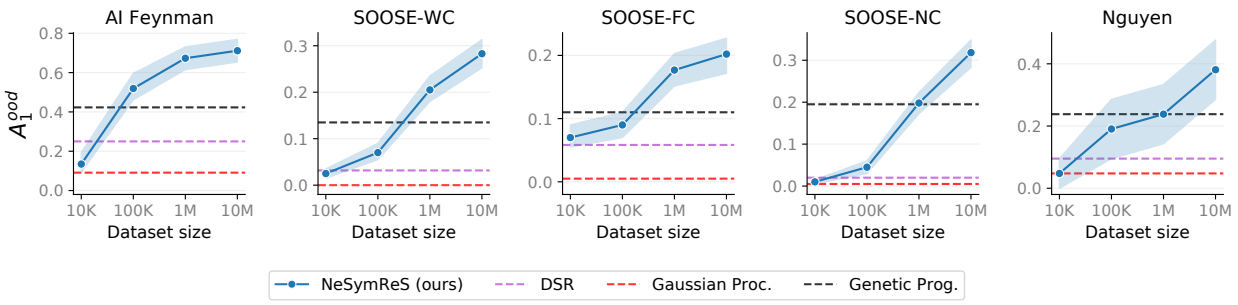


*Figure 2.* Accuracy as a function of the size of the pre-training dataset, for a fixed computational budget (∼100 s) at test time. We report reference values for the baselines to emphasize that these approaches do not improve with experience over time.

We further distinguish between two metrics, accuracy $A_1$ and accuracy $A_2$, each of which can be either computed *iid* or *ood*. Accuracy $A_1$ is computed as follows: for every point $(x, y)$ and prediction $f_{\hat{e}}(x) = \hat{y}$, the point is correctly classified if `numpy.isclose(`$y$`,` $\hat{y}$`)` returns True.[9] Then, an equation is correctly predicted if $> 95\%$ of points are correctly classified. For this metric we can keep *all* outputs, including NaNs and $\pm\infty$, which are still representative of whether the symbolic equation was identified correctly. Accuracy $A_2$ is computed by measuring the coefficient of determination $R^2$ between $y$ and $\hat{y}$, excluding NaNs and $\pm\infty$. An equation is correctly identified according to $A_2$ if the $R^2 > 0.95$. We found the two metrics to correlate significantly, and in the interest of clarity we will use only $A_1$ in the main text, and show results with $A_2$ in the Appendix C.

## 5. Results

We test three different aspects of the proposed approach: (i) To what extent does performance improve as we increase the size of the pre-training data? (ii) How does our approach compare to state-of-the-art methods in symbolic regression? (iii) What is the impact of the number of input-output pairs available at *test time*?

**(i) Accuracy as a Function of Pre-training Data**

In order to test the effect of pre-training data on test performance, we trained our NeSymReS model on increasingly larger datasets. More specifically, we consider datasets consisting of 10K, 100K, 1M and 10M equation skeletons. Every aspect of training is the same as described in Section 4. We train all models for the same number of iterations, but use early stopping on a held-out validation set to prevent overfitting.

In Figure 7 we report the accuracy on the 5 test sets using a beam size of 32 for NeSymReS, and for all baselines whatever hyperparameter configuration that used comparable (but strictly no less) amount of computing time. In all datasets, increasing the size of the pre-training data results in higher accuracy for NeSymReS. Note that the baselines do not make use of the available pre-training data, and as such it does not have any effect on the performance at test time. From here onwards, we will always use the model pre-trained on 10M equation skeletons.

*Conclusion:* The performance of NeSymReS steadily improves as the size of the pre-training dataset increases, exploiting the feature that symbolic equations can be generated and evaluated extremely quickly and reliably with computers. The trend observed appears to continue for even larger datasets, in accordance to (Kaplan et al., 2020), which leaves open interesting avenues for extremely large scale experiments.

**(ii) Accuracy as a Function of Test-time Compute.**

For every method (including baselines), we vary the corresponding hyper-parameter that increases how much time and compute is invested at test time to recover an equation from observing a fixed set of input-output pairs. We report the hyper-parameters and ranges in Table 1.

Making a fair comparison of run-times between different methods is another challenging task. To make the comparison as fair as possible, we decided to run every method on a single CPU at the time. Note that this is clearly a sub-optimal hardware setting for our 26-million parameters Transformer, which would be highly parallelizable on GPU.

The results on all five datasets are shown in Figure 3 and Figure 4. On all datasets, our method outperforms all baselines both in time and accuracy by a large margin on most budgets of compute. On AIF our NeSymRes is more than three orders of magnitudes faster at reaching the same maximum accuracy as the second-best method, i.e. Genetic Programming, despite running on CPU only. We attribute the low accuracy achieved by (Petersen, 2021) to the presence of constants, to the fact that their model does not directly ob-

[9] With parameters atol 1e-3 and rtol 0.05.

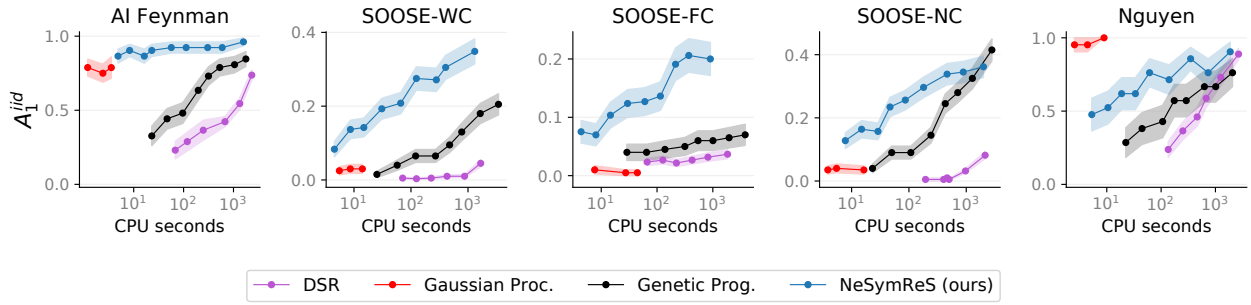


*Figure 3.* Accuracy in distribution as a function of time for all methods ran on a single CPU per equation.

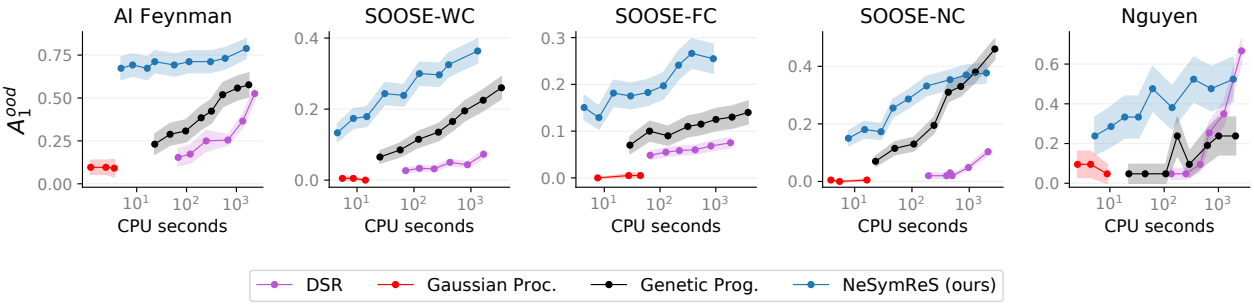


*Figure 4.* Accuracy out of distribution as a function of time for all methods ran on a single CPU per equation.

serve the input-output pairs, and the use of REINFORCE (Williams, 2004). The Gaussian Process baseline performs extremely well in distribution, reaching high accuracy in a very short amount of time, but poorly out of distribution. This is expected as it does not try to regress the symbolic equation. On Nguyen, NeSymReS achieves relatively high scores more rapidly than the other baselines. For large computation times ($\approx 10^3$ seconds) NeSymReS performs comparably with DSR despite the latter being fine-tuned on two equations of the benchmark (Nguyen-7 and Nguyen-10). The relatively lower performance of NeSymReS on SOOSE-NC can be explained by the fact that both datasets do not have any constants in the equations, while NeSymReS is trained with a large prior on the presence of constants.

*Conclusion:* By amortizing the computation performed at pre-training time, NeSymReS is extremely accurate and efficient at test time, even running on CPU.

*Table 1.* Hyper-parameters that vary to increase the amount of compute invested by every method.

| Method | Hyper-param | Range |
|---|---|---|
| G. Proc. (Rasmussen, 2003) | Opt. restarts | $\{8, 16, 32\}$ |
| Genetic Prog. (Koza, 1994) | Pop. size | $\{2^{10}, ..., 2^{17}\}$ |
| DSR (Petersen, 2021) | Epochs | $\{2^2, ..., 2^7\}$ |
| NeSymReS (ours) | Beam size | $\{2^0, ..., 2^8\}$ |

**(iii) Performance Improves with more Points $p$**

In practice, depending on the context, a variable number of input-output pairs might be available at test time. In Figure 5, we report the accuracy achieved for a number of input-output points that varies in the range from 1 to 1024. Even though NeSymReS was pre-trained with no more than 500 points, it still performs reliably with fewer points.

*Conclusion:* NeSymReS is a flexible method and its performance is robust to different numbers of test data, even when such numbers differ significantly from those usually seen during pre-training. Furthermore, its accuracy levels grow with the number of points observed at test time.

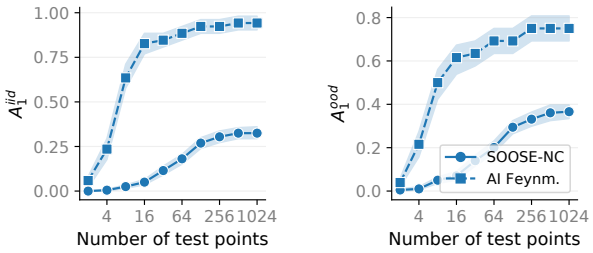


*Figure 5.* Accuracy as a function of number of input-output pairs observed at test time.

## 6. Discussion

Building on the recent successes of large scale pre-training, we have proposed the first method that *learns the task* of symbolic regression. This approach deviates from the majority of existing techniques in the literature which need to be retrained from scratch on each new equation and does not improve over time with access to data and compute (Sutton, 2019). We showed empirically that by pre-training on a large distribution of millions of equations, this simple approach outperforms several strong baselines, and that its performance can be improved by merely increasing the size of the dataset. The key feature that enables this approach is that — unlike for computer vision and natural language — high-quality training data can be generated efficiently and indefinitely using any standard math library and a computer.

In pre-training, the data generation plays a crucial role within our framework. By changing this distribution over equations (including support, constants, number of terms and their interactions), it is possible for the user to finely tune the inductive bias of the model, adapting it to specific applications. In light of its favourable scaling properties and its powerful prior over symbolic expression, we believe that our model could find applications in several domains in the Natural Sciences and engineering, control, and model-based Reinforcement Learning. The scale of our experiments is still relatively small compared to the largest large-scale experiments run to date (Brown et al., 2020; Devlin et al., 2018; Chen et al., 2020b), both in terms of dataset and model sizes. Nonetheless, the results we showed already seem to indicate that NeSymReS could improve significantly with access to extremely large scale compute.

**Time and Space Complexities** The approach we presented scales favorably over several dimensions: computation scales linearly in the number of input-output points due to the Set Transformer (Lee et al., 2019), and linearly in the number of input dimensions. For future work, it would be interesting to train even larger models on larger datasets with more than three independent variables.

**Limitations** Even though our approach can scale to an arbitrary number of input and output dimensions, there are limitations that should be considered. Fitting the constants using a non-linear optimizers like BFGS can prove to be hard if the function to be optimized has several local minima. In this case, other optimization strategies that can deal with non-convex loss surfaces might be beneficial, such as CMA-ES (Hansen, 2016). One more limitation of our approach is that the pre-trained model as presented cannot be used at test time if the number of input variables is larger than the maximum number of variables seen during pre-training. Finally, one more limitation of the neural network we adopt is that it does not directly interact with the function evaluator available in the math libraries of most computers. If, for example, the first candidate sampled from the network is completely wrong, our current approach cannot adjust its posterior over equations based on this new evidence, but simply sample again.

**Conclusions** What are the desirable properties of a strong symbolic regressor? It should:

- *scale* favourably with the number of datapoints observed at test time and with the number of input variables;
- *improve over time* with experience;
- *be targetable* to specific distributions of symbolic equations;
- *be flexible* to accommodate very large or very small values.

In this paper, we showed that all of these properties can be obtained, and provided a simple algorithm to achieve them in the context of symbolic regression. Our largest pre-trained model can be accessed on our repository.

## Acknowledgements

We thank Guillame Lample for the helpful discussion, and Riccardo Barbano for proofreading the manuscript. LB acknowledges the CSEM Data Program Fund for funding. GP acknowledges the Max Planck ETH Center for Learning Systems for funding. AN acknowledges the International Max Planck Research School for Intelligent Systems for funding. This work was also supported by the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B), and the ML cluster funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC number 2064/1 - Project number 390727645.

## References

Alaa, A. and van der Schaar, M. Demystifying black-box models with symbolic metamodels. 2019.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020a.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. Big self-supervised models are strong semi-supervised learners, 2020b.

Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dubčáková, R. Eureqa: software review, 2011.

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

Forrest, S. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

Hansen, N. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. Big transfer (bit): General visual representation learning, 2020.

Koza, J. R. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *International Conference on Machine Learning*, pp. 1945–1954. PMLR, 2017.

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

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/lee19d.html.

Martius, G. and Lampert, C. H. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

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. 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.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Petersen, B. K. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *International Conference on Learning Representations*, 2021.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving, 2020.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer school on machine learning*, pp. 63–71. Springer, 2003.

Sahoo, S. S., Lampert, C. H., and Martius, G. Learning equations for extrapolation and control. In *Proc. 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, 2018*, volume 80, pp. 4442–4450. PMLR, 2018. URL http://proceedings.mlr.press/v80/sahoo18a.html.

Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

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

Snelson, E. and Ghahramani, Z. Sparse gaussian processes using pseudo-inputs.

Sutton, R. The bitter lesson. 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Titsias, M. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pp. 567–574. PMLR, 2009.

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

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *arXiv preprint arXiv:2006.10782*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.