A. Model details

A.1. NeSymReS Transformer Details

The model consists of an encoder and a decoder. The encoder takes as input numerical data, $X \in \mathbb{R}^{(d_x+d_y)\times n}$, where d_x is the number of independent variables, d_y the number of dependent variables and n the number of support points. In order to prevent exploding gradient and numerical instabilities, we convert each entry of X into a multi-hot bit representation according to the half precision IEEE-754 standard. This operation yields a new input tensor, $\tilde{X} \in \mathbb{R}^{(d_x+d_y)\times b\times n}$ where $b = 16$ is the dimension of the bit representation. The output of the encoder is a latent vector $z \in \mathbb{R}^{d_z}$, providing a compressed representation of the equation to be modelled. Such latent vector is then used to condition the decoder via a standard Transformer multi-head attention mechanism. During the pre-training phase, the input of the decoder is given by the sequence of tokens representing the ground truth-equation expressed in prefix notation. Such sequence is opportunely masked in order to prevent information leakage in the decoder forward step. The output is then given by a string of symbols, representing the predicted equation, again in prefix notation. During inference, the decoder is only provided with the information from the latent vector z and generates a prediction autoregressively. We use *beam search* to obtain candidate solutions. After removing potentially invalid equations, the remaining equations are modified to include constant placeholders with the procedure described in Appendix B. Using BFGS, we fit these constants. We select the best equation among the so-found candidates, based on the validation loss, with an added regularization penalty of 10^{-14} for each token in the skeleton. Note that BFGS is currently the most time-consuming step of our pipeline. While in all our experiments we run the optimization procedure serially (i.e., one candidate equation at the time), the procedure can be easily parallelized across equations.

Encoder and decoder use the same hidden dimension H and number of heads, h for their multi-head attention modules. In the following, we provide further details about the architectural design choices, hyper-parameters and library of functions used by our model. We trained our model on a single GeForce RTX 2080 GPU for 3 days.

Encoder For the encoder, we opted for the Set Transformer architecture from Lee et al. (2019). Our choice is motivated in light of the better scaling properties of this method when it comes to input lenght n, i.e. $\mathcal{O}(nm)$ compared to the standard transformer encoder, $\mathcal{O}(n^2)$, where m is a set of trainable inducing points. Referring to the notation of the original paper, our encoder is formed by n_e Induced Set Attention Blocks (ISABs) and one final component performing Pooling by Multihead Attention (PMA). ISABs differ from the multi-head self attention blocks present in the original Transformer architecture, since they introduce $m < n$ learnable inducing points that reduce the computation burden associated with the self-attention operation. PMA allows us to aggregate the output of the encoder into d_z trainable abstract features representing a compressed representation of the input equation. Overall, the encoder consists of $11M$ trainable parameters. All the hyper-parameters of the encoder are listed in Table 2.

Decoder The decoder is a standard Transformer decoder. It has n_d layers, hidden dimension H, h attention heads. Input symbols are encoded into the corresponding token embeddings and information about relative and absolute positions of the tokens in the sequence is injected by adding learnable positional encodings to the input embeddings. Two different masks are used to avoid information leakage during the forward step and to make the padding token hidden to the attention modules. In total, our dictionary is formed by s different tokens, including binary and unary operators and independent variables. A list of all the elements of our dictionary is provided in Table 4. Overall, the decoder consists of $15M$ trainable parameters. All the hyper-parameters of the decoder are listed in Table 3.

Table 2. Encoder hyper-parameters.

Parameter name	Symbol	Value
Number of ISABs	n_e	5
Hidden dimension	H	512
Number of heads	h	8
Number of PMA features	d_z	10
Number of inducing points	m	50

Table 3. Decoder hyper-parameters.

Parameter name	Symbol	Value
Number of layers	n_d	5
Hidden dimension	H	512
Number of heads	h	8
Embedding dimension	s	32

Symbol	Integer Id	Symbol	Integer Id	Symbol	Integer Id	Symbol	Integer Id
sos	1	arcsin	9	$\times$	17	-2	25
eos	2	arctan	10	Pow	18	-1	26
x	3	cos	11	sin	19	0	27
y	4	cosh	12	sinh	20	1	28
z	5	coth	13	$\surd$	21	2	29
c	6	$\div$	14	tan	22	3	30
arccos	7	exp	15	tanh	23	4	31
$+$	8	ln	16	-3	24	5	32

Table 4. Symbols available for expression generation and their corresponding integer tokens. The symbols "sos" and "eos" stand for "start of sequence" and "end of sequence" respectively. The padding symbol is not reported in the table and is associated with the token 0. Note that **not all of these symbols** appear in our pre-training dataset, as detailed in Table 6 we only used a small subset for our experiments.

A.2. Baselines

Deep Symbolic Regression (DSR) For DSR, we use the standard hyper-parameters provided in the open-source implementation of the method, with the setting that includes the estimation of numerical constants in the final predicted equation. DSR depends on two main hyper-parameters, namely the entropy coefficient λ_H and the risk factor ϵ. The first is used to weight a bonus proportional to the entropy of the sampled expression which is added to the main objective. The second intervenes in the definition of the final objective with depends on the $(1-\epsilon)$ quantile of the distribution of rewards under the current policy. According with the open-source implementation and the results reported in (Petersen, 2021), we choose $\epsilon = 0.05$ and $\lambda_H = 0.005$. The set of symbols available to the algorithm to form mathematical expressions is given by $\mathcal{L} = \{+, -, \times, \div, \sin, \cos, \exp, \ln, c\}$, where c stands for the constant placeholder.

Genetic Programming For Genetic Programming, we opt for the open-source Python library `gplearn`. Our choices for the hyper-parameters are listed in Table 5 and mostly reflect the default values indicated in the library documentation. The set of symbols available to the algorithm to form mathematical expressions is the default one and is given by $\mathcal{L} = \{+, -, \times, \div, \sqrt{\cdot}, \ln, \exp, \textit{neg}, \textit{inv}, \sin, \cos\}$, where *neg* and *inv* stand for "negation" ($x \mapsto -x$), and inversion ($x \mapsto x^{-1}$), respectively.

Table 5. Genetic Programming hyper-parameters. The parameter *Population size* is varied within the range indicated during the experiments reported in Section 5.

Parameter name	Value
Population size	$\{2^{10}, ..., 2^{15}\}$
Selection type	Tournament
Tournament size (k)	20
Mutation probability	0.01
Crossover probability	0.9
Constants range	$(-4\pi, 4\pi)$

Gaussian Processes This is the only baseline that is *not* a symbolic regression method per se, as it learns a mapping from x to y directly. The appealing property of Gaussian Processes is that they are very accurate in distribution, and are very fast to fit in the regime we considered. We opted for the open-source `sklearn` implementation of Gaussian Process regression with default hyper-parameters. The covariance is given by the product of a constant kernel and an RBF kernel. Diagonal Gaussian noise of variance 10^{-10} is added to ensure positive-definitness of the covariance matrix. L-BGFS-B is used for the optimization of the marginal likelihood with the number of restarts varied as indicated in Table 1.

A note about function sets Unfortunately, not all methods support all primitive functions that appear in a given dataset. For example, NeSymReS *could* support x^6, and x^y — that appear in the Nguyen dataset described in Appendix C — but as we did not include these primitives in the pre-training phase, the version we use in our experiments will not be able to correctly recover these equations. DSR and the implementation of Genetic Programming that we adopted are both lacking $\arcsin$ in their function set.

While missing primitives lowers the upper bound in performance that a method can reach for a given dataset, it also makes it easier to fit the other equations that *do not* contain those primitives, as the function set to search is effectively smaller.

B. Experimental details

B.1. Training

Operator	$+$	$\times$	$-$	$\div$	$\sqrt{}$	Pow	ln	exp	sin	cos	tan	arcsin
Unormalized Prob	10	10	5	5	4	4	4	4	4	4	4	1

Table 6. Operators and their corresponding un-normalized probabilities of being sampled as parent node.

Training Dataset Generation For generating skeletons, we built on top of the method and code proposed in (Lample & Charton, 2019), which samples expression trees. For our experiments, each randomly-generated expression tree has 5 or fewer non-leaf nodes. We sample each non-leaf node following the unnormalized weighted distribution shown in Table 6. Each leaf node has a probability of 0.8 of being an independent variable and 0.2 of being an integer. Trees that contain the independent variable x_2 must also have the independent variable x_1. Those containing the independent variable x_3 must also include the independent variables x_1 and x_2. We then traverse the tree in pre-order and obtain a semantically equivalent string of the expression tree in a prefix notation. We convert the string from prefix to infix notation and simplify the mathematical expression using the Sympy library. The resulting expression is then modified to include constant placeholders as explained in the following paragraph. This expression is what we refer to as a *skeleton*, as the value of constants has not been determined yet.

For our experiments, we repeat the procedure described above to obtain a pre-compiled dataset of 10M equations. To compile the symbolic equation into a function that can be evaluated by the computer on a given set of input points, we relied on the function *lambidfy* from the library Sympy. We store the equations as functions, in order to allow for the support points and values of the constants to be resampled at mini-batch time during pre-training. We opted for a partially pre-generated dataset instead of sampling new equations for every batch in order to speed up the generation of training data for the mini-batches.

Training Details As described in Section, 4.2, during training we sampled mini-batches of size $B = 150$ from the generated dataset. For each equation, we first choose the number of constants, n_c, that differ from one. n_c is randomly sampled within the interval ranging from 0 to $\min(3, N_c)$ where N_c is the maximum number of constants than can be placed in the expression. Then, we sample the constants' values from the uniform distribution $\mathcal{U}(1, 5)$. We randomly select n_c among the available placeholders and replace them with the previously obtained numerical values. The remaining constants are set to one. We then generate up to 500 support points by sampling- independently for each dimension - from uniform distributions with varying extrema as described in Section 4. If the equation does not contain a given independent variable, such variable is set to 0 for all the support points. For convenience, we drop input-output pairs containing NaNs and entries with an absolute value of y above 1000. Finally we take the equation in the mini-batch with the minimum number of points, and drop valid points from the other equations so that the batch tensor has a consistent length across equations. Figure 6 shows the training and validation curves of the main model used for all our experiments.

Training Dataset Distribution The dataset does not consist of unique mathematical expressions. Indeed, some skeletons are repeated, and some skeletons are mathematically equivalent. Overall, within the 10M dataset, we have $\sim$ 1.2M unique skeletons. Since longer expressions tend to be simplified into shorter expressions during the dataset generation, shorter expressions are the most frequent ones. We report in Table 7 the ten most recurrent expressions. Of these 1.2M unique skeletons, at least 96.3% represents unique (numerically distinct) mathematical expressions. The counting procedure is described in the next paragraph. The average length of an expression in infix notation is 8.2 tokens. The minimum and the maximum are 1 and 23 respectively, which corresponds in infix notation to the expressions x and $\frac{x^2 \operatorname{asin}^2(x)}{-x_1^2 \operatorname{asin}^2(x)+1}$.

Addition of Numerical Constants During dataset generation and inference, we introduce constant placeholders by attaching them to the generated or predicted skeletons. This step is carried out by multiplying all unary operators in the expressions by constant placeholders (except for the "pow" operator). The same procedure is repeated with independent

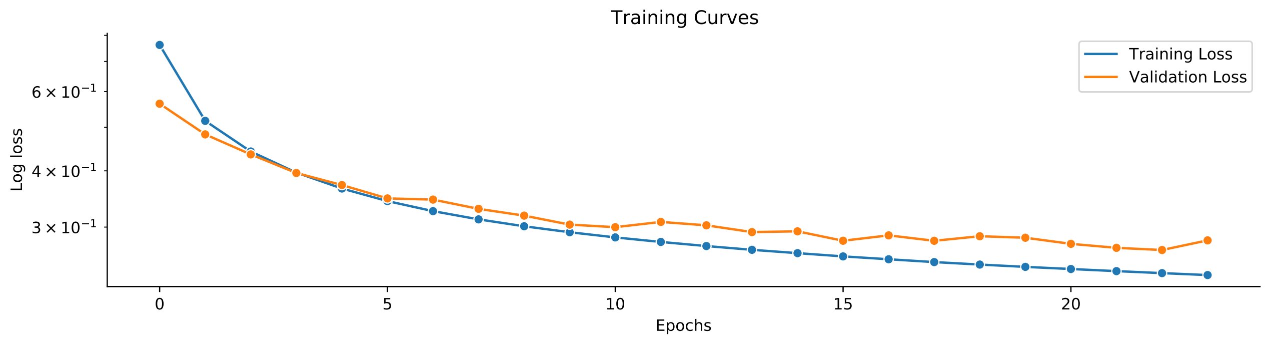

Figure 6. Loss as a function of pre-training for NeSymReS

Expression	x_1	$x_1 + x_2$	x_2^2	$x_1 x_2$	$x_1 + x_2 + x_3$	$x_1 + 1$	$-x_1$	$x_1 - 1$	e^{x_1}	$\cos(x_1)$
Frequency	18649	6716	4789	4707	3781	3727	3698	2917	2594	2157

Table 7. 10 most frequent expressions in the dataset

variables for which also additive constants are introduced. Longer expressions tend to have more placeholders in comparison to shorter ones.

B.2. Evaluation details

All results reported, i.e. for all methods and datasets, are accuracies over all equations in the dataset. Error bars in all plots denote the standard error of the mean estimate.

Metrics Details As detailed in 4.6 we evaluate performances both within the training support (A^{iid}) and outside of the training support (A^{ood}). More specifically, for the latter the support is created as follows: given an equation with in-sample support of (lo, hi), we extend the support of each side by $(hi - lo)$ for every variable present in the equation.

Creating the SOOBE Dataset The SOOBE (stricly out-of-sample equations) dataset contains entirely different skeletons from the pre-training dataset, which do not overlap numerically nor symbolically. To create it, we list all the different expressions in the training dataset and then randomly sample from this set, excluding the sampled expressions from the training set. We first sample a random support of 500 points from the uniform distribution $\mathcal{U}(-10, 10)$, for each independent variable. Two expressions are different if their images, given the support points, are different. Note that this is a conservative criterion, as two expressions may have the same image in the sampled support (relatively to a fixed tolerance), yet being distinct.

Benchmarks As explained in 4.4 we evalutate our trained model on five datasets: AI-Feynman, SOOBE-WC, SOOBE-NC, SOOBE-FC, Nguyen. All the equations of AI-Feynman used in our evaluation are listed in table 8. 50 randomly sampled equations out of 200 from the SOOBE dataset, are listed in table 9.

Expression	Support x_1	Support x_2	Support x_3	Expression	Support x_1	Support x_2	Support x_3
$\frac{\sqrt{2}e^{-\frac{x_1^2}{2}}}{2\sqrt{\pi}}$	(1, 3)	None	None	$\frac{x_1x_3^2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{\sqrt{2}e^{-\frac{x_2^2}{2x_1^2}}}{2\sqrt{\pi}x_1}$	(1, 3)	(1, 3)	None	$\frac{x_1}{4\pi x_2^2}$	(1, 5)	(1, 5)	None
$\frac{\sqrt{2}e^{-\frac{(x_2-x_3)^2}{2x_1^2}}}{2\sqrt{\pi}x_1}$	(1, 3)	(1, 3)	(1, 3)	$\frac{x_1}{4\pi x_2x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)	$\frac{3x_1^2}{20\pi x_2x_3}$	(1, 5)	(1, 5)	(1, 5)
x_1x_2	(1, 5)	(1, 5)	None	$\frac{x_1x_2^2}{2}$	(1, 5)	(1, 5)	None
$\frac{x_1}{4\pi x_2x_3^2}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1}{x_2(x_3+1)}$	(1, 5)	(1, 5)	(1, 5)
x_1x_2	(1, 5)	(1, 5)	None	$\frac{x_1x_2}{-\frac{x_1x_2}{3}+1}+1$	(0, 1)	(0, 1)	None
$x_1x_2x_3$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{x_1^2x_2}{2}$	(1, 5)	(1, 5)	None	$\frac{x_1x_2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)
$\frac{x_1x_2}{\sqrt{-\frac{x_2^2}{x_3^2}+1}}$	(1, 5)	(1, 2)	(3, 10)	$-x_1x_2\cos(x_3)$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_2+x_3}{1+\frac{x_2x_3}{x_1^2}}$	(1, 5)	(1, 5)	(1, 5)	$-x_1x_2\cos(x_3)$	(1, 5)	(1, 5)	(1, 5)
$x_1x_2\sin(x_3)$	(1, 5)	(1, 5)	(0, 5)	$\sqrt{\frac{x_1^2}{x_2^2}-\frac{\pi^2}{x_3^2}}$	(4, 6)	(1, 2)	(2, 4)
$\frac{x_1}{x_2}$	(1, 5)	(1, 5)	None	$x_1x_2x_3^2$	(1, 5)	(1, 5)	(1, 5)
$\operatorname{asin}(x_1\sin(x_2))$	(0, 1)	(1, 5)	None	$x_1x_2^2$	(1, 5)	(1, 5)	None
$\frac{1}{\frac{x_3}{x_2}+\frac{1}{x_1}}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1x_2}{2\pi x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1}{x_2}$	(1, 10)	(1, 10)	None	$\frac{x_1x_2x_3}{2}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1\sin^2\left(\frac{x_2x_3}{2}\right)}{\sin^2\left(\frac{x_2}{2}\right)}$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1x_2}{4\pi x_3}$	(1, 5)	(1, 5)	(1, 5)
$\operatorname{asin}\left(\frac{x_1}{x_2x_3}\right)$	(1, 2)	(2, 5)	(1, 5)	$x_1x_2(x_3+1)$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_3}{1-\frac{x_2}{x_1}}$	(3, 10)	(1, 2)	(1, 5)	$\frac{x_1}{2x_2+2}$	(1, 5)	(1, 5)	None
$\frac{x_3\left(1+\frac{x_2}{x_1}\right)}{\sqrt{1-\frac{x_2^2}{x_1^2}}}$	(3, 10)	(1, 2)	(1, 5)	$\frac{4\pi x_1x_2}{x_3}$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_1x_2}{2\pi}$	(1, 5)	(1, 5)	None	$\sin^2\left(\frac{2\pi x_1x_2}{x_3}\right)$	(1, 2)	(1, 2)	(1, 4)
$x_1+x_2+2\sqrt{x_1x_2}\cos(x_3)$	(1, 5)	(1, 5)	(1, 5)	$\frac{x_1x_2}{2\pi}$	(1, 5)	(1, 5)	None
$\frac{3x_1x_2}{2}$	(1, 5)	(1, 5)	None	$2x_1(1-\cos(x_2x_3))$	(1, 5)	(1, 5)	(1, 5)
$\frac{x_2x_3}{x_1-1}$	(2, 5)	(1, 5)	(1, 5)	$\frac{x_1^2}{8\pi^2x_2x_3^2}$	(1, 5)	(1, 5)	(1, 5)
$x_1x_2x_3$	(1, 5)	(1, 5)	(1, 5)	$\frac{2\pi x_1}{x_2x_3}$	(1, 5)	(1, 5)	(1, 5)
$\sqrt{\frac{x_1x_2}{x_3}}$	(1, 5)	(1, 5)	(1, 5)	$x_1(x_2\cos(x_3)+1)$	(1, 5)	(1, 5)	(1, 5)

Table 8. AI-Feynman equation with less than 4 independent variables and the supports as indicated in (Udrescu & Tegmark, 2020)

Expression	Expression
$4.931x_1 - x_2 + 4.023\tan\left(x_1^2 - 4.027x_3\right)$	$x_1\sqrt{-x_3 + \sin(x_2)}$
$\sin(\cos(3.488x_1\tan(2.798x_1) + 2.938x_1))$	$2.29x_2\cos(x_2) + \cos\left(\frac{1.044x_1}{x_2}\right)$
$\sqrt{-x_1 + \frac{x_2x_3}{x_1}}$	$\frac{x_3 + \frac{3.797\sin(x_1)}{x_2}}{x_3}$
$\sin(4.84x_3(2.3x_1 - 3.494x_2 + 1))$	$x_1 - 4.843x_2x_3 + x_2 + \cos(x_3)$
$\sin(x_3) + \sin\left(\frac{x_3}{x_1 - x_2}\right)$	$\cos\left(x_1 + 1.504x_2 + (x_2 + x_3)^2\right)$
$x_1(2.683x_1 + x_2\cos(x_3)) + 1$	$x_1\left(-4.641x_1 + \cos^2\left(4.959\sqrt{x_2}\right)\right)$
$4.631\sin\left(4.419\sin\left(\frac{x_2x_3}{x_1^2}\right)\right)$	$x_2\left(x_2 - \frac{-x_1 - 1}{x_2}\right)$
$3.874x_3 + 4.12 - \frac{1}{x_1 + 4.322x_2x_3}$	$4.47x_1 + 1.193\cos\left(1 + \frac{1}{x_2}\right)$
$\frac{1.858x_1x_3}{-x_1 + x_2} - 3.661x_3$	$3.63x_1\cos\left(1.427x_2^3 + x_2\right)$
$2.846x_2 + \sin\left(x_1^5 + 2.258x_3\right)$	$-x_1(1.196x_1 + \sin(x_1 + x_2))$
$\frac{x_2 + x_3 + \frac{x_2 - 4.615}{x_2}}{x_1}$	$x_3 + \frac{x_3}{x_1 + 2.318x_2 + x_3}$
$-x_3 + \frac{0.221(-x_1 + x_2)}{\log(x_2)} - 1$	$x_1 - \frac{7.74\sqrt{0.383x_1 + x_2}}{x_2}$
$(1.261x_1 + 3.29\cos(1))\log(4.169x_2)$	$1 + \frac{0.221\tan(3.972x_2)}{x_2(-3.549x_1 + x_2)}$
$\frac{x_2}{x_2 + \cos(x_1x_3)}$	$1 - \sin(x_1(x_1 + \sin(x_1)))$
$2.161x_3\cos^3\left(x_1^2 + x_2\right)$	$\cos(x_1) - \sqrt{\cos(x_2)}$
$3.196\tan(\cos(4.459x_1) - \tan(1)) - 1$	$-2.586x_2 + \frac{0.693\cos(x_2 - 1)}{x_1}$
$\sqrt{x_2^2 - x_2 - e^{2.103x_1}}$	$-8.802x_1 + 3.379\log\left(x_1 + x_2^4\right)$
$-x_2 + \log\left(x_1\left(-x_2 + \frac{1.513}{e}\right)\right)$	$\sin\left(\frac{2.696x_2}{-x_1 + 2.364x_2}\right) + 1.097$
$3.919\log\left(1.731\sqrt{x_1\sin(2.176x_2)}\right)$	$x_1 - x_2\cos\left(x_1^3\right)$
$x_1 + \frac{0.422x_1}{x_2(4.26x_1 + \cos(x_2))}$	$x_1 - 2.636(3.271x_2 + x_3)\sin(x_1) + 2.387$
$\sin\left(\frac{3.553(-0.531x_2 + x_3)^2}{x_1 + 1.244x_2}\right)$	$x_1(x_1 + x_3 + 1) + \sqrt{x_2}$
$\sqrt{x_1\left(\frac{x_2}{x_3} + x_3\right)}$	$x_2 + \log(3.949x_1(x_1 + \sin(x_2)))$
$-x_1\sin(\tan(x_1x_2)) + x_1$	$x_1 + 3.183\sin(3.696\log(x_2(x_2 - 1)))$
$x_2\sin\left(x_1 - 1.47x_3^2\right)$	$x_2^2 + 1.209\sin(x_1x_2)$
$x_1\log(\sin(\tan(x_1)))$	$\frac{x_2(x_1 + 1.756\log(2.756\cos(x_3)))}{x_1}$

Table 9. 50 random equations extracted from the SOOBE dataset (version with constants).

C. Additional Results

C.1. Additional Metrics on all Benchmarks

In this section, we show that the conclusions drawn in Section 5 with the A_2 metric are consistent when the A_1 metric is considered instead.

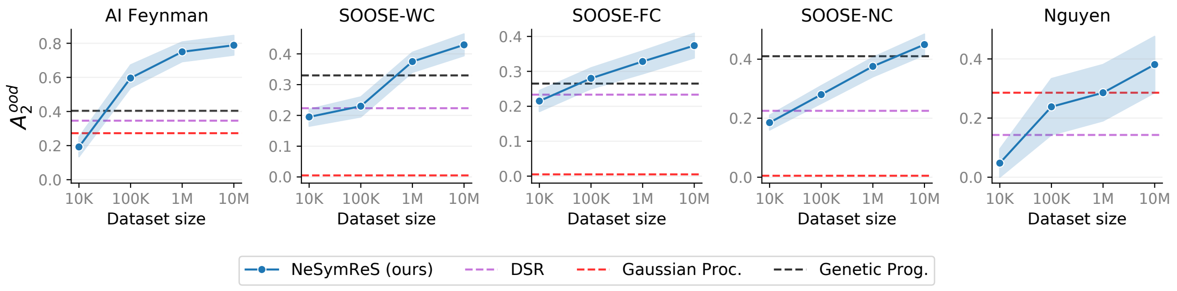

Figure 7. Accuracy as a function of the size of the pre-training dataset, for a fixed computational budget (∼100 s) at test time.

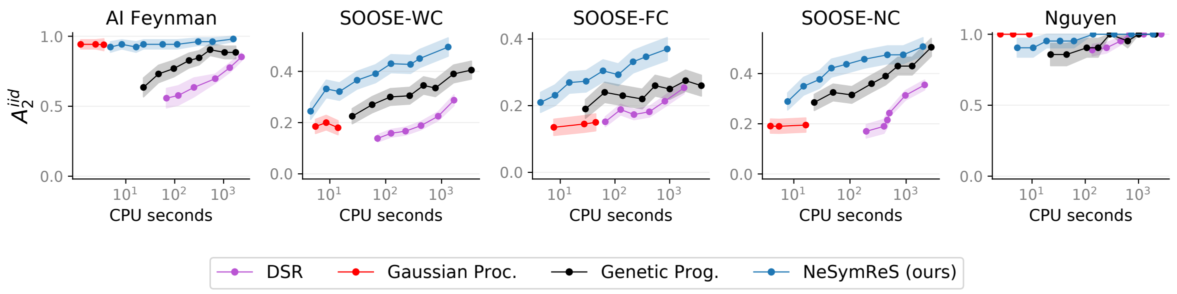

Figure 8. Accuracy in distribution as a function of time for all methods ran on a single CPU per equation.

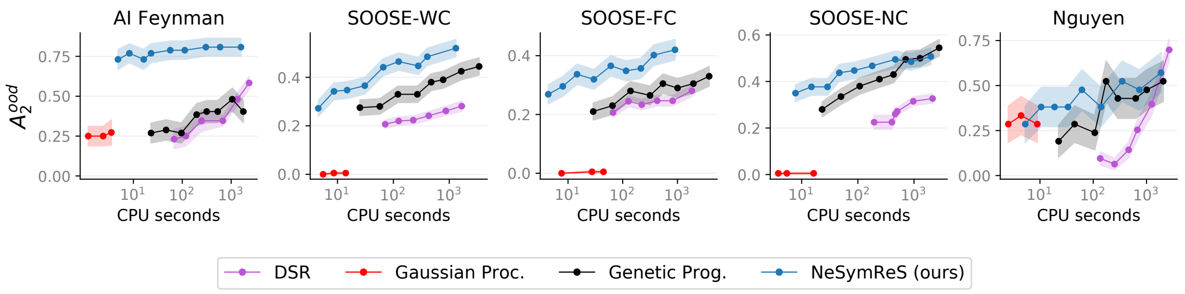

Figure 9. Accuracy out of distribution as a function of time for all methods ran on a single CPU per equation.

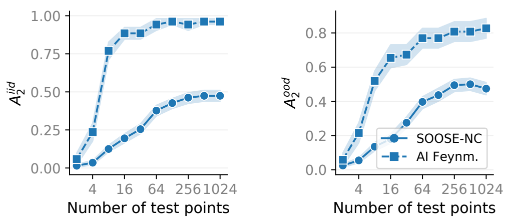

Figure 10. Accuracy as a function of number of input-output pairs observed at test time.

C.2. OOD Extrapolation Examples

Examples of 2D Functions Fig. 11 provides 4 examples of functions learned by our model. All the considered functions depend only on two independent variables, x_1 and x_2. The visualizations show that our method, given a relatively small set of support points (black dots in the figure) is able to extrapolate out of distribution by retrieving the underlying symbolic expression. The last row of Fig. 11 shows that NeSymReS at times finds valid alternative expressions based on trigonometric identities, i.e. $\sin(x) = \cos(x - \pi/2) = -\cos(x + \pi/2)$.

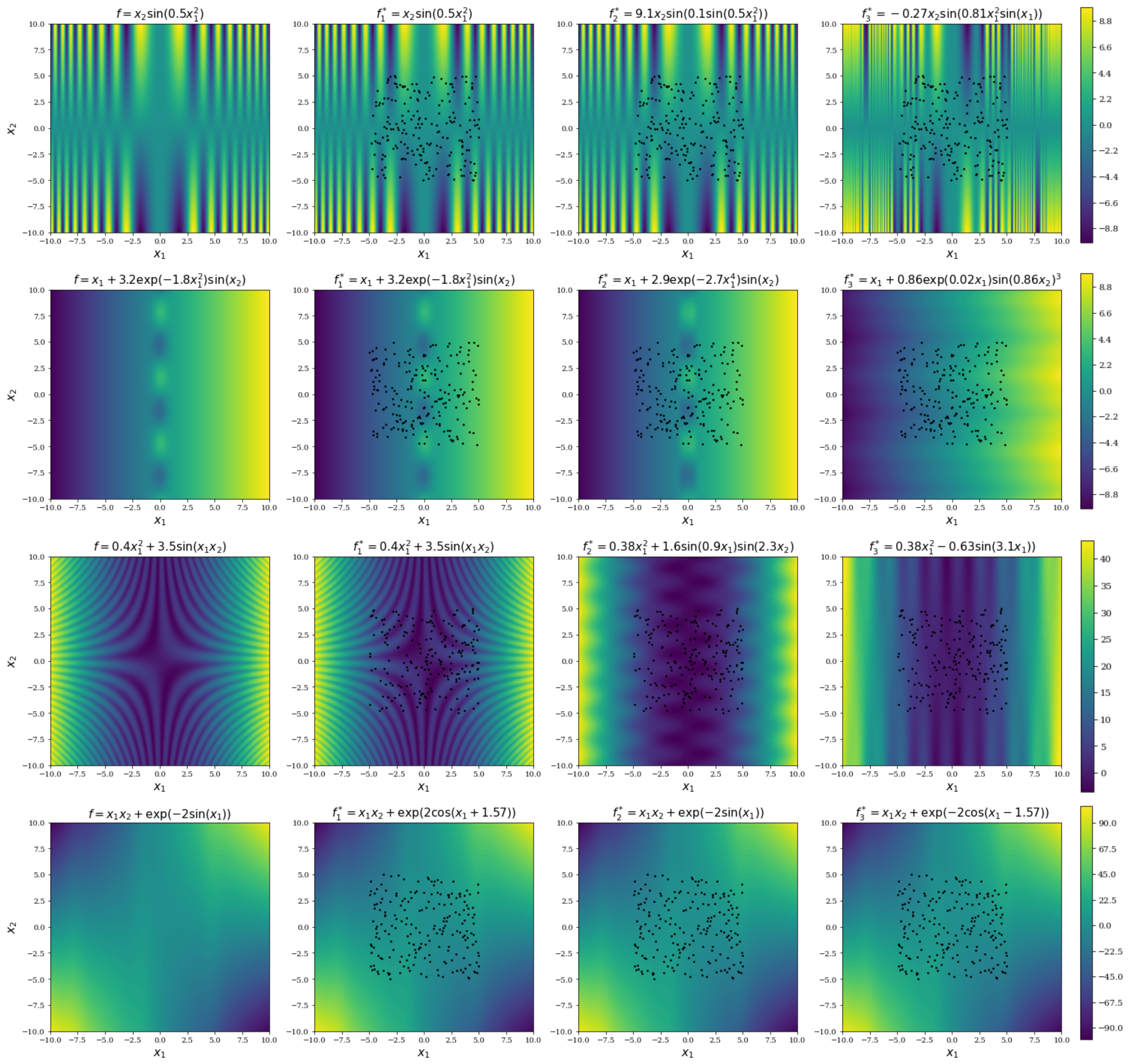

Figure 11. Four examples of functions learned by our model. The first column shows the ground truth equation, whereas the remaining three represent the top predictions of our model sorted by likelihood (from left to right).