# Narrowing the Search: Classifying Equation Bases to Improve Symbolic Regression

Rayhan Divo Tanudjaja

Under the direction of

Rumen Dangovski
PhD Candidate
Massachussetts Institute of Technology

Owen Dugan
Undergraduate Student
Massachussetts Institute of Technology

Professor Marin Soljačić
Professor of Physics
Massachussetts Institute of Technology

Research Science Institute
August 1, 2023

## Abstract

In this study, we aim to train a transformer model in order to recognize symbols from an equation $y = f(x)$ when given inputs $x_1$, $x_2$, $x_3$ and output $y$. To accomplish this, we utilize the nanoGPT framework, specifically designed for training medium-sized GPTs (Generative Pre-trained Transformers), and leverage the transformer's parallel processing capabilities. Despite achieving a maximum of 20% accuracy for completely correct predictions of all fourteen bases, the transformer performs remarkably in predicting whether the prescence of individual bases in the equation, achieving 80-90% on average.



## Summary

Symbolic regression is the task of discovering mathematical equations given a dataset of input-output pairs. For example, given three inputs 1, 3, and 4 with an output 8 and give it to a symbolic regression model, we would expect it to know that we need to add up the three inputs to find the output. However, past approaches to do this may take a lot of time; in order to allow other models to find the right equation faster, we can train a deep learning model to predict what symbols may be in the equation so that other models can search exclusively through these symbols.

# 1 Introduction

The role that equations play in the natural sciences is crucial, as they provide concise representations of the relationships within natural phenomena, enabling scientists to predict, experiment, and formulate hypotheses about the world around us. [1] Equations are often discovered through observation, much like how Newton was said to have observed an apple falling from a tree, leading to the laws of gravity. In today's age, where the power of computing surpasses the capabilities of the human mind, we can utilize technology to gain deeper insights into the world we live in and create equations from vast amounts of data.

In the domain of machine learning, the challenge of hypothesis generation has been a prominent area of research, which aims to develop methods that can automatically generate meaningful and interpretable hypotheses or equations from data, providing valuable insights into complex relationships and patterns. For instance, in linear regression, machine learning models attempt to fit a linear equation to the data, enabling predictions and inferences.

One approach to this challenge is symbolic regression, a machine-learning regression method that aims to discovers mathematical equations that best describe a given dataset by searching a space of possible mathematical symbols to use in the equations. Up until recently, symbolic regression has been purely approached through the lens of genetic programming, an evolutionary algorithmic technique that can create mathematical expressions through iterative processes of selection and mutation. The more popular approach has been to train a neural network to generate equations using an encoder-decoder architecture, which benefits from their ability to handle large and complex data, but faces challenges related to the risk of overfitting. While all these techniques like these empower researchers to process and model vast data, they all encounter a common challenge: the vastness of the search space.

Depending on how many symbols and coefficients are in the equation, the possibilities are boundless, resulting in infeasible computation. Hence, to optimize the task of symbolic regression, we need to constrain the search space so that symbolic regression models can find the corresponding equation in a shorter amount of time. We can do this by training

a transformer model to predict what symbols are likely to appear when given input-output pairs, so that these models can use the results of these to start their search within these symbols exclusively.

In this paper, we train a transformer to predict which symbols, referred to as "bases" in the context of equations, are most likely to be part of an equation when given a list of input-output pairs to that corresponding equation using a generative pre-trained transformer (GPT), with a data set of 40 thousand equations consisting of bases drawn from a list of fourteen symbols.

## 2 Related Works

### 2.1 Initial Approaches and Genetic Programming

The naive solution towards executing symbolic regression is brute force; by going through every single possible combination. However, this is an inefficient solution and complex equations will take a long time to be found. An approach by Schmidt & Lipson [2] involves the use of genetic programming [3], which uses a tree-based representation to represent potential expressions; the elements of the tree undergo genetic operations such as combination and mutation of symbols and numbers in order to generate equations representative of the data.

The genetic programming approach can find complex mathematical functions without the need for predefined expression structures, allowing it to generate equations suitable for a variety of disciplines. However, genetic programming tends to be extremely sensitive to hyperparameters and does not perform well in a large scale; complex mathematical functions come with a high computing cost as the number of genetic operations required to find the right expression increase as the length of the expression increases, due to the increased possible symbol and number combinations [4].

### 2.2 Neural Networks

The use of neural networks for symbolic regression is a more recent approach; they emulate the structure of the human brain, consisting of layers of neurons that approximate functions to find the numerical relationship between input-output pairs within the context of symbolic regression.

One of the advantages of training a neural network for symbolic regression is its scalability; parallel processing allows neural networks to process large datasets. However, using neural networks may be difficult to interpret, as they are considered black-box models. This makes interpreting the learned mathematical expressions a difficult task.

The first approach to training a neural network for symbolic regression was done by Martius and Lampert [5] at a small scale, however, without Lample & Charton's [6] large equation dataset generator, Biggio et. al's [7] neural network for symbolic regression would not have been able to be trained in a large scale, using 10 million equations in the pre-training stage. More recent approaches towards symbolic regression include OccamNet [8] and Receptance Weighted Key Value (RWKV) [9]. OccamNet emphasizes on finding simpler equations as opposed to high dimensional and complex equations as does the principle of Occam's razor, while Receptance Weighted Key Value (RWKV) combines the work of transformers, which tend to be quicker with finding equations, and recurrent neural networks, which tend to be more accurate in finding equations.

# 3 Methods

## 3.1 Task and Loss Function

For this task, we will be using a multihot classification task. A multihot classification is a type of classification task where the goal is to categorize data into multiple classes. In traditional onehot classification, each data point can only have one active label. However, in multihot classification, a data instance can have multiple "hot" or active labels, while others remain inactive. This representation is often represented as a binary vector, where each element corresponds to a specific class, and a value of 1 indicates the data instance belongs to that class, while 0 means it does not. For example, the equation $y = \log(x_1 \times x_2 + x_3)$ would have a bases vector of

$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ \cdots \\ 1 \end{pmatrix} = \begin{pmatrix} + \\ \times \\ - \\ \div \\ \cdots \\ \log \end{pmatrix}$$

The bases vector will be the target or label while the values of the parameters ($x_1$, $x_2$, $x_3$) will be the input for our model training.

The loss function we are using here is the binary cross entropy loss function. Aside from being one of the more preferred multihot classification task loss functions, it treats classes as independent binary decisions and the probabilities it outputs are easily interpretable. The binary cross entropy loss function is

$$L(\theta) = -\log[P_{\theta(x_i)}]_{y_i}$$

where $P_\theta$ is the probability that symbol $\theta$ appears in the equation for each dataset point

$(x_i, y_i)$. As $P_{\theta(x_i)}$ is maximized, the loss is minimized. We can maximize $P_{\theta(x_i)}$ through gradient descent, shown by the following algorithm below:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

where $\theta$ is the specific parameter corresponding to the symbol in the model, $\alpha$ is the learning rate of the model, and $\nabla_\theta L(\theta)$ is the gradient of the loss function with respect to the model parameters $\theta$.

We are also using an AdamW optimizer, an optimization algorithm that combines elements of Adam and weight decay commonly used in train neural networks. It extends the Adam optimizer by decoupling weight decay from the gradient updates. By doing this, AdamW more effectively prevents overfitting and enhances generalization performance in deep learning models, which makes it a good fit for our model.

### 3.2 Dataset Generation

#### 3.2.1 Implementation

The dataset we used to train the model consists of thirty thousand randomly generated equations, each with one to three parameters ($x_1$, $x_2$, $x_3$). Since we are trying to improve symbolic regression for the purpose of discovery in the natural sciences, we can limit our symbol vocabulary to the ones that appear the most in equations used to describe natural phenomena, like addition, subtraction, multiplication, division, trigonometric functions, and exponential functions.

An equation generator for this purpose was already created by Biggio et. al [7], which also includes a configuration file that allows us to alter the distribution of symbols in the generated dataset as well as a program that converts the raw equation data file format from HDF to CSV, showing the equation and number of parameters.

Table 1 shows a tally of the fourteen symbols used in the a list of 100 natural science equations generated by ChatGPT, providing us with a rough distribution of each symbol

that we plan to use in the dataset, which we can input into the configuration file to create a bias in the model to generate equations more similar to ones in the natural sciences. The list of equations can be found in Appendix A.

| Symbol | Occurrence |
|---|---|
| $+$ | 18 |
| - | 38 |
| $\times$ | 96 |
| $\div$ | 49 |
| $\sqrt{x}$ | 4 |
| $x^2$ | 22 |
| $x^3$ | 3 |
| $x^4$ | 1 |
| sin | 6 |
| cos | 3 |
| tan | 3 |
| arcsin | 3 |
| $e^x$ | 3 |
| log | 2 |

Table 1: Occurrence of symbols in a list of natural science equations

When an equation is obtained from the dataset into the transformer with a data loader, the parameters and corresponding $y$ are calculated. The parameters are randomly generated values between -100 and 100; in the event that any of the randomly generated parameters of an equation return NaN, $-\infty$, or $\infty$, the equation is thrown out and not loaded; this is because such values won't appear in the natural sciences, hence training a model with these values wold not be useful.

### 3.2.2 Limitations

Dealing with equations with domains that do not encompass all real values was difficult; in theory, it should be reasonable to train models with NaNs so that the model can easily identify certain bases that aren't valid for all values of $x_i$ or $y$.

However, due to training with a transformer, the encoder-decoder architecture explained in the section below only accepts numerical inputs; a neural network would not be able to

handle NaNs as an input, hence we had to make the decision to remove all equations that could produce NaNs; this may limit our training data to contain very little equations with bases that do not have a domain encompassing all values of $x$, like log, square roots, and arcsin.

### 3.3 Dataset Model

The model we used for this classification task is a transformer. A transformer model is a powerful neural network that automatically transforms one type of input into another. Transformers use a self-attention mechanisms to detect dependencies between all input tokens, which are the individual elements that the input is comprised of. This mechanism allows parallelization and thus improves efficiency significantly. Commonly used with language models like ChatGPT, transformers have improved accuracy in various tasks like natural language processing as they are able to predict the next word in sentences [10].

nanoGPT is a framework for training medium-sized generative pre-trained transformers (GPT) created by Andrej Karpathy. nanoGPT was built as a language model; however, since we are using numbers instead of letters, we need to modify nanoGPT's encoder-decoder architecture to accept our numerical inputs and give us numerical outputs.

Figure 1 shows how our data from our `DataLoader` is processed into our output. We utilize a custom `DataLoader` to load data in batches, where each batch contains 64 samples ($B = 64$). Each sample consists of a group of 4x100 matrices representing input-output pairs for an equation. Specifically, the first three rows in each matrix represent the inputs of the equation ($x_1$, $x_2$, $x_3$), and the fourth row corresponds to the equation's output $y$. We process these input-output pairs sequentially, which we refer to as timestamps (T=100), making a total of 100 input-output pairs per batch.

The batch is then passed through a linear layer `nn.Linear(4, 384)`, which performs the linear transformation on the batch and maps the 4 input features to 384 output features, creating a new tensor with a shape of B $\times$ 384 $\times$ T, where B represents the batch size

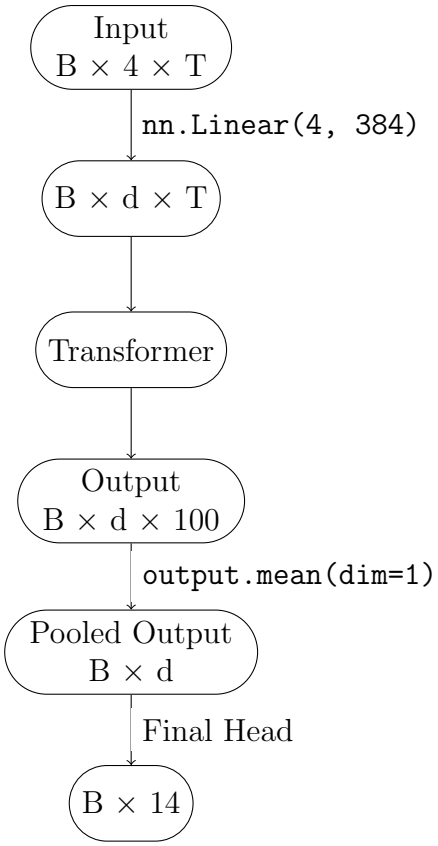


Figure 1: Encoder-decoder architecture of our modified nanoGPT

($B = 64$), d is the dimension of the transformed features ($d = 384$), and T denotes the number of timestamps ($T = 100$). This higher dimensional representation allows the model to capture complex relationships and patterns within the data and hence make more accurate predictions.

Originally, nanoGPT was built with an `nn.Embedding` layer which is optimized for learning grammar and syntax with language inputs, but since our inputs are numerical, a `nn.Linear` layer is better suited for this task.

This tensor is then fed into the transformer, which processes the input using its self-attention mechanism and understanding the intricate dependencies between elements in the sequence. The transformer will output a tensor also in the shape B $\times$ 384 $\times$ T, which represents the model's learned representations of the input.

Now, we pool the output over the timestamp dimension, taking the mean amongst all

100 timestamps for each of the 384 features, giving a tensor of shape B $\times$ 384, where B is the batch size, and each element of the tensor represents the mean value of the corresponding feature across the timestamps.

This tensor is then passed into a final head which condenses the B $\times$ 384 into a B $\times$ 14 tensor, where B is the batch size and 14 is our vocabulary length.

## 4 Experiment and Results

To conduct our experiments, we are finding the best accuracy on the test set on a variety of learning rates on a batch size of 512. The training of the model stops when the accuracy of the model stabilizes or starts decreasing; we tested the model on five learning rates, and the results are shown in table 2.

| Starting Learning Rate | Maximum Accuracy (%) | Epoch |
|---|---|---|
| $1 \times 10^{-3}$ | 19.92 | 18 |
| $6 \times 10^{-4}$ | 19.68 | 15 |
| $1 \times 10^{-4}$ | 20.66 | 25 |
| $6 \times 10^{-5}$ | 19.65 | 37 |
| $1 \times 10^{-5}$ | 18.79 | 74 |

Table 2: Maximum test accuracies to their corresponding learning rate

As we can see, the model is most accurate with a learning rate of $\alpha = 1 \times 10^{-4}$ at epoch 25. Hence, we will analyze our results from this specific model performance.

In figure 2, we observed that the transformer achieved a complete accuracy of only 20% with a loss of 0.275. This complete accuracy indicates how often the transformer correctly identifies the presence of all fourteen symbols in an equation. However, it's important to note that this measure can be quite stringent, as it requires the model to precisely classify the presence of all symbols correctly in order to be counted as completely accurate.

On the other hand, when we evaluate the performance of the transformer based on individual symbols, we find that it performs remarkably well. For most symbols, the accuracy reaches above 90%, indicating that the transformer is adept at identifying individual symbols

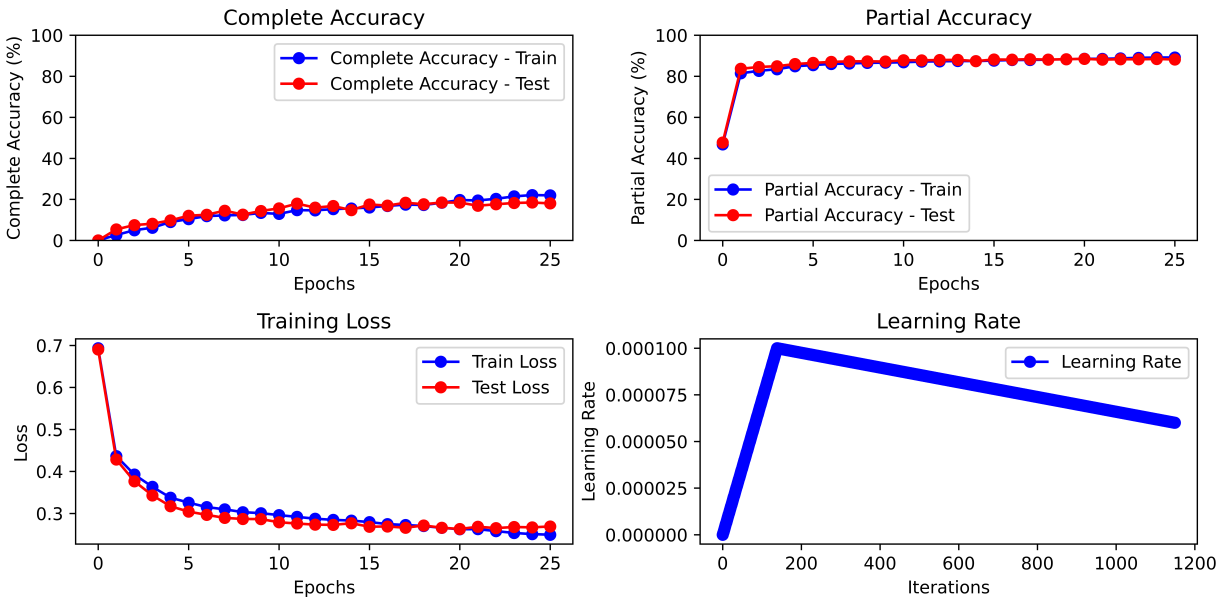


Figure 2: Model performance

accurately.

To better understand the transformer's overall performance, we can consider the average accuracy of each individual symbol. By doing this, we get a more comprehensive and generous view of the model's capabilities. The average accuracy reflects the model's ability to correctly identify symbols on average, taking into account both accurate and slightly inaccurate predictions. In fact, we can use figure 3, figure 4, and figure 5 analyze the individual accuracies of each base to better understand how well this model performs.

These operators had the highest frequency in table 1, meaning that these four symbols appeared the most. The multiplication and divsion bases reach a higher accuracy than the addition and subtraction bases; this could be attributed to the fact that the frequency of the multiplication base is five times that of the frequency of the addition base, meaning that most of the time, the model could possibly be guessing that there would be a multiplication or division symbol all the time and it would mostly be right. Regardless, a 70-80% accuracy on the addition and subtraction bases is still very useful to symbolic regression models; those symbols are still used very frequently throughout scientific equations.

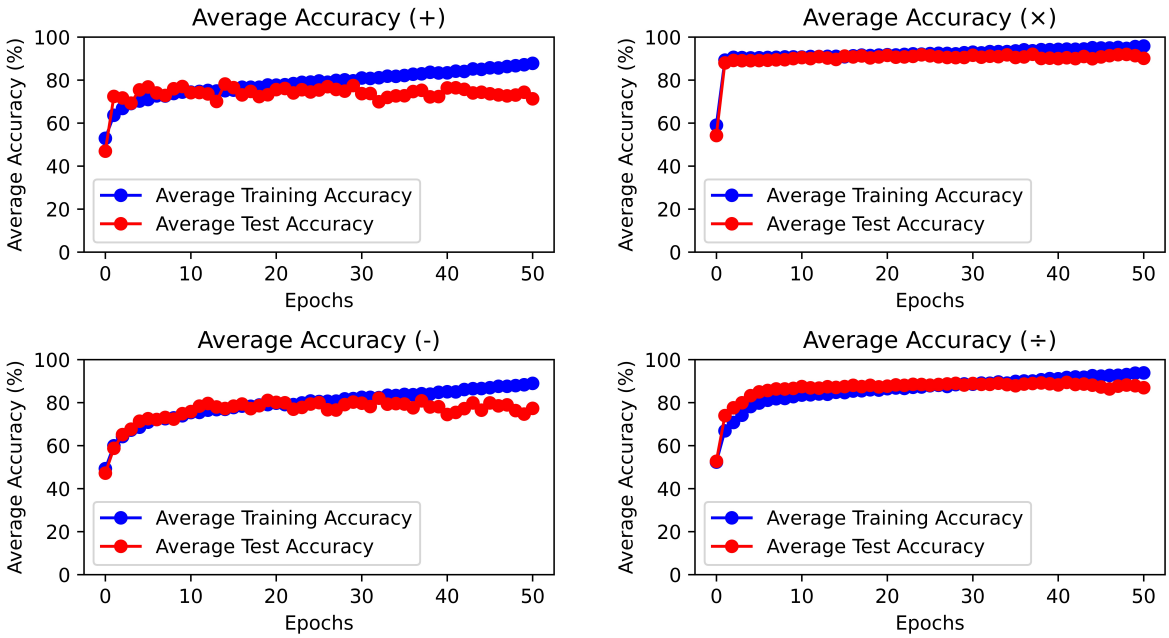


Figure 3: Accuracy per base, operators

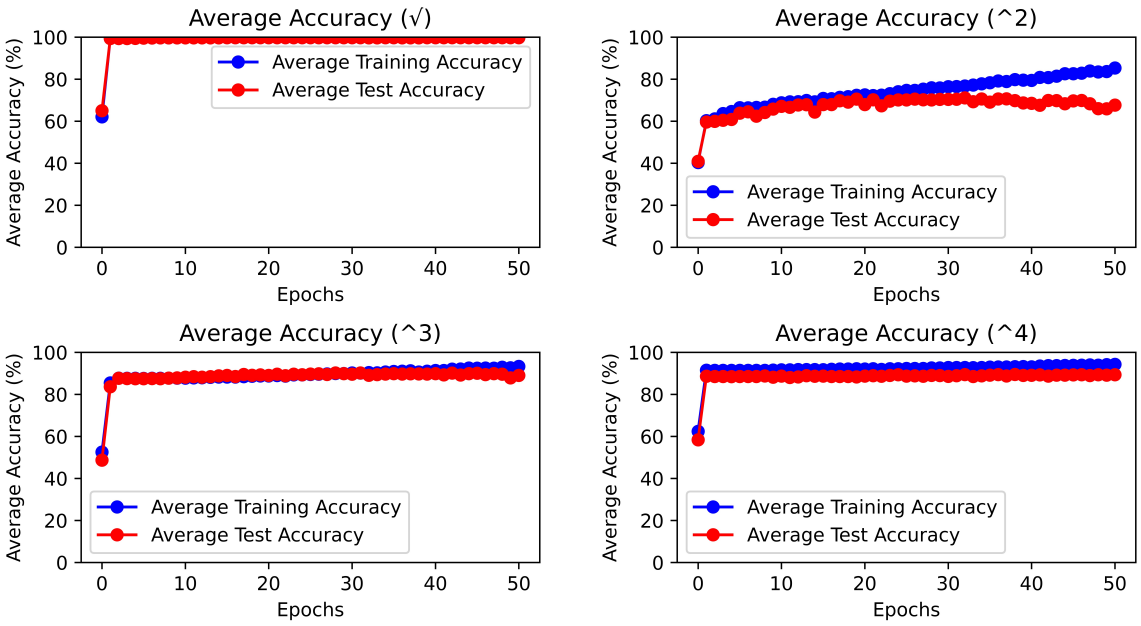


Figure 4: Accuracy per base, power functions

The square root, cubed and fourth power all have near perfect accuracy; this could also be contributed to a similar issue encountered by the multiplication and division symbol, but in the alternate direction. Because these symbols rarely do appear in scientific discovery, the model could also guess that those bases will not be there and it would be correct most of the time. However, the square root is still relatively used in the natural sciences; its perfect accuracy could be due to the fact that we discarded all equations that would throw away NaNs; since square roots do not have a continuous domain throughout all real values of $x_i$.

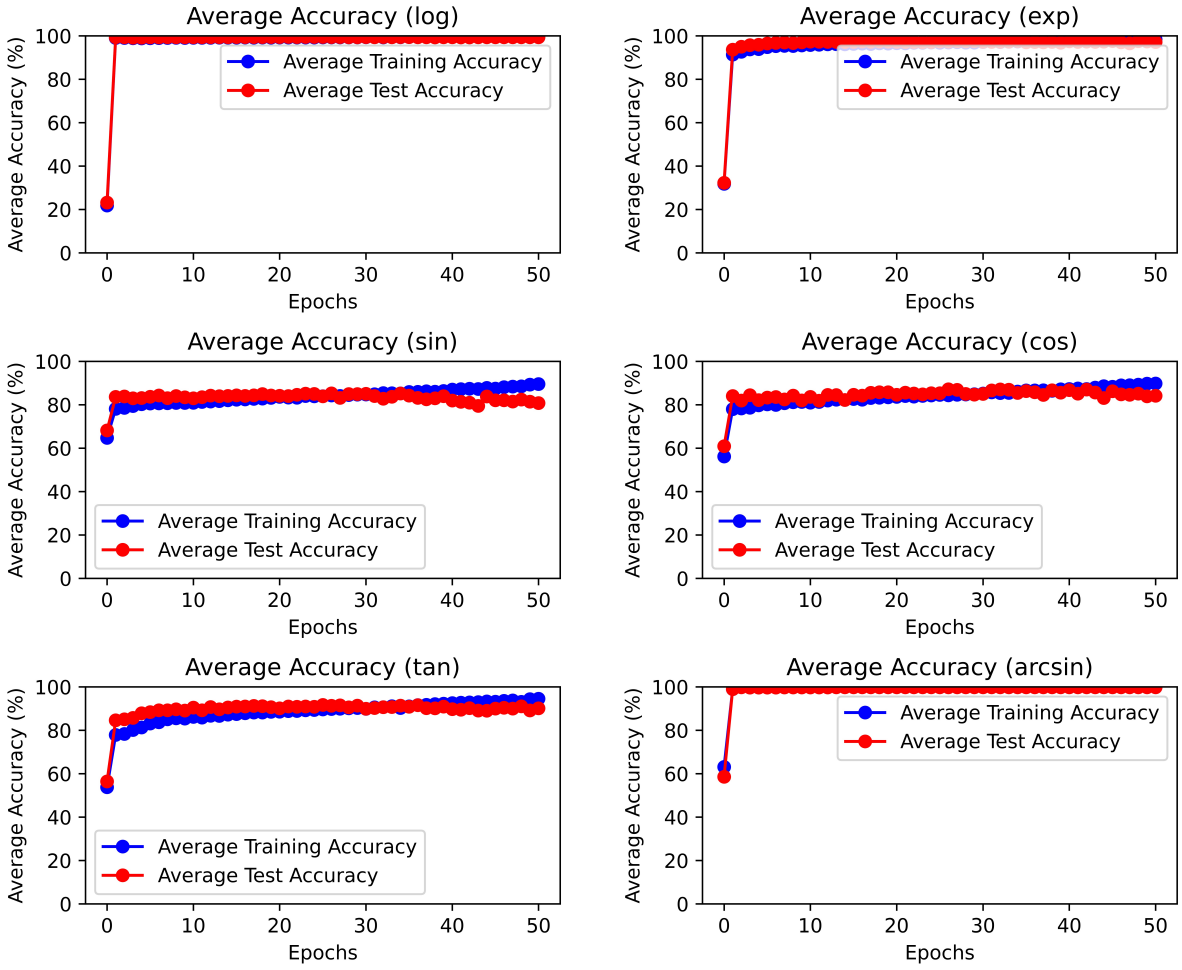


Figure 5: Accuracy per base, exponential and trigonometric functions

The logarithmic and arcsin function also suffer from the same issue due to their non-continuous domain; however, the other trigonometric functions are relatively well detected, presumably because of their limited range; the exponential function also performs quite

well, possibly due to it generating large $y$ values during parameter generation, possibly to the point that it returns inf and becomes discarded, leaving very little functions containing said bases.

Additionally, a quick sampling of predicted values and y-values shows us that in most cases, only one to two bases are misclassified. Sometimes there is a tendency to classify a base as being there even though it is not, which is not too harmful to the task of the model as it just means we constrain the search space to a lesser extent. Sometimes, however, a base that is supposed to be present is classified as not present, and that might make other models take longer to find their equation if the intended solution is not even within their search space.

## 5 Conclusion

In this study, we trained a transformer model to recognize bases from input-output pairs that corresponded to a mathematical function with the purpose to optimize other symbolic regression models by constraining their search space. The model trained relatively well; the model would have a minimum loss of approximately 0.275, and despite having a maximum accuracy of around 20%; this was attributed to the test's definition accurate which was matching all 14 bases to the target vector, whereas in reality all the bases that needed to be detected were in fact detected. If taking the average accuracy of individual base classification, our transformer performs impressively, reaching accuracies of 80% to 90%. However, some base classifications were affected greatly from the bias that we imposed onto the dataset, resulting in near 100% accuracies on bases that would either rarely appear or would nearly always appear. Additionally, some bases that should have been more prevalent in the dataset like the square root were also underrepresented due to their finite domains.

Once we refine the transformer and ensure that it is performing at its best, we plan to incorporate it into OccamNet, a neural network architecture trained for symbolic regression with an emphasis on finding simpler and more interpretable equations as opposed to complex

equations, in order to allow it to find equations faster by reducing its search space, as well as compare our performance to various symbolic regression benchmarks like SRBench.

## 6 Acknowledgments

I would like to thank my mentors Rumen Dangovski and Owen Dugan for their invaluable guidance, Professor Marin Soljačić for allowing my research to happen with his research grant, my tutor Shuvom Sadhuka and teaching assistant Alec Dewulf for their enlightening assistance and comments in the creation of this research paper, Allen Lin, Donald Liveoak, Max Bee-Lindgren, AnAn Desimone, Rich Wang, and Sally Zhu for their assistance in LaTeX, my loving mother and father, as well as the Research Science Institute, the Center for Excellence in Education, and the Massachusetts Institute of Technology for this remarkable program and experience.

## References


[1] E. P. Wigner. The unreasonable effectiveness of mathematics in the natural sciences. richard courant lecture in mathematical sciences delivered at new york university, may 11, 1959. *Communications on Pure and Applied Mathematics*, 13(1):pp. 1–14, 1960. doi: https://doi.org/10.1002/cpa.3160130102. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160130102. https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.3160130102.

[2] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):pp. 81–85, 2009. doi:10.1126/science.1165893. URL https://www.science.org/doi/abs/10.1126/science.1165893. https://www.science.org/doi/pdf/10.1126/science.1165893.

[3] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):pp. 87–112, 1994. doi:doi:10.1007/BF00175355.

[4] B. K. Petersen, M. Landajuela, T. N. Mundhenk, et al. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2021. 1912.04871.

[5] G. Martius and C. H. Lampert. Extrapolation and learning equations, 2016. 1610.02995.

[6] G. Lample and F. Charton. Deep learning for symbolic mathematics, 2019. 1912.01412.

[7] L. Biggio, T. Bendinelli, A. Neitz, et al. Neural symbolic regression that scales, 2021. 2106.06427.

[8] A. Costa, R. Dangovski, O. Dugan, et al. Fast neural models for symbolic regression at scale, 2021. 2007.10784.

[9] B. Peng, E. Alcaide, Q. Anthony, et al. Rwkv: Reinventing rnns for the transformer era, 2023. 2305.13048.

[10] A. Radford, K. Narasimhan, T. Salimans, et al. Improving language understanding by generative pre-training. 2018.

# Appendix

## A Natural Science Equations

**Physics:**

- Newton's Second Law: $F = m \cdot a$
- Einstein's Mass-Energy Equivalence: $E = m \cdot c^2$
- Speed Equation: $v = \frac{d}{t}$
- Ohm's Law: $V = I \cdot R$
- Coulomb's Law: $F = k \cdot \frac{q_1 \cdot q_2}{r^2}$
- Wave Equation: $v = f \cdot \lambda$
- Density Equation: $\rho = \frac{m}{V}$
- Ideal Gas Law: $PV = n \cdot R \cdot T$
- Faraday's Law: $\varepsilon = -\frac{d\Phi}{dt}$
- Gauss's Law: $\oint E \cdot dA = \frac{Q}{\varepsilon_0}$
- Kepler's Third Law: $T^2 = \frac{4\pi^2 \cdot r^3}{G \cdot M}$
- Schrödinger Equation: $\hbar\frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi + V\psi$
- Planck's Relation: $E = h \cdot f$
- Heisenberg's Uncertainty Principle: $\Delta x \cdot \Delta p \geq \frac{\hbar}{2}$
- Einstein's General Theory of Relativity: $G_{\mu\nu} + \Lambda \cdot g_{\mu\nu} = 8\pi G \cdot T_{\mu\nu}$
- Snell's Law: $n_1 \sin\theta_1 = n_2 \sin\theta_2$

- Hooke's Law: $F = -k \cdot x$
- Work-Energy Theorem: $W = \Delta KE$
- Bernoulli's Equation: $P + \frac{1}{2}\rho v^2 + \rho g h = \text{constant}$
- Doppler Effect: $f' = f \cdot \frac{v + v_d}{v - v_s}$
- Ideal Fluid Bernoulli's Equation: $P_1 + \frac{1}{2}\rho v_1^2 + \rho g h_1 = P_2 + \frac{1}{2}\rho v_2^2 + \rho g h_2$
- Gravitational Potential Energy: $U = m \cdot g \cdot h$
- Pascal's Law: $P = \frac{F}{A}$
- Magnetic Field due to a Straight Wire: $B = \frac{\mu_0 \cdot I}{2\pi \cdot r}$
- Angular Velocity: $\omega = \frac{\theta}{t}$
- Photoelectric Effect: $KE = h \cdot f - \Phi$
- Coulomb's Law for Electric Fields: $E = k \cdot \frac{q}{r^2}$

**Chemistry:**

- Ideal Gas Law: $PV = n \cdot R \cdot T$
- Henderson-Hasselbalch Equation: $pH = pKa + \log\left(\frac{[A^-]}{[HA]}\right)$
- Nernst Equation: $E = E^0 - \left(\frac{RT}{nF}\right)\ln(Q)$
- Arrhenius Equation: $k = A \cdot e^{-\frac{E_a}{RT}}$
- Gibbs Free Energy: $\Delta G = \Delta H - T\Delta S$
- Rate Equation for a First-order Reaction: $\text{rate} = k \cdot [A]$
- Rate Equation for a Second-order Reaction: $\text{rate} = k \cdot [A]^2$

- Ideal Gas Law with Density: $PV = R \cdot T \cdot \left(\frac{\rho}{M}\right)$
- Half-Life of a Reaction: $t_{1/2} = \frac{0.693}{k}$
- Rate Law for a Third-order Reaction: $\text{rate} = k \cdot [A]^2 \cdot [B]$
- Avogadro's Law: $\frac{V}{n} = k$
- Law of Mass Action: $Q = \frac{[C]^c \cdot [D]^d}{[A]^a \cdot [B]^b}$
- Raoult's Law: $P_A = X_A \cdot P_A^0$
- Van't Hoff Equation: $\ln\left(\frac{K_2}{K_1}\right) = -\frac{\Delta H^\circ}{R \cdot \left(\frac{1}{T_2} - \frac{1}{T_1}\right)}$
- Clausius-Clapeyron Equation: $\ln\left(\frac{P_2}{P_1}\right) = \frac{\Delta H_{\text{vap}}}{R \cdot \left(\frac{1}{T_1} - \frac{1}{T_2}\right)}$
- Bragg's Law: $n \cdot \lambda = 2d \cdot \sin\theta$
- Law of Definite Proportions (Proust's Law)
- Combined Gas Law: $\frac{P_1 \cdot V_1}{T_1} = \frac{P_2 \cdot V_2}{T_2}$

**Biology:**

- Hardy-Weinberg Equilibrium: $p^2 + 2pq + q^2 = 1$
- Logistic Growth Equation: $\frac{dN}{dt} = r \cdot N \cdot \left(1 - \frac{N}{K}\right)$
- Nernst Equation (Neurobiology): $E = \left(\frac{RT}{zF}\right) \ln\left(\frac{[\text{ion}]_{\text{out}}}{[\text{ion}]_{\text{in}}}\right)$
- Michaelis-Menten Equation: $v = \frac{V_{\max} \cdot [S]}{K_m + [S]}$
- Henderson-Hasselbalch Equation (Biological Buffers): $pH = pKa + \log\left(\frac{[A^-]}{[HA]}\right)$
- Beer-Lambert Law: $A = \varepsilon \cdot l \cdot c$
- Poisson Distribution: $P(x; \lambda) = \frac{e^{-\lambda} \cdot \lambda^x}{x!}$

**Geology:**

- Hubble's Law: $v = H_0 \cdot d$
- Mohr-Coulomb Failure Criterion: $\tau = c + \sigma_n \cdot \tan \phi$
- Darcy's Law: $Q = -K \cdot A \cdot \frac{\Delta h}{L}$
- Law of Superposition: Younger rocks are above older rocks in undisturbed sequences.

**Astronomy:**

- Stefan-Boltzmann Law: $L = 4\pi \cdot R^2 \cdot \sigma \cdot T^4$
- Hubble's Law: $v = H_0 \cdot d$
- Kepler's Third Law (Astronomy Version): $T^2 = \frac{4\pi^2 \cdot a^3}{G \cdot (M_1 + M_2)}$
- Escape Velocity: $v_{\text{escape}} = \sqrt{\frac{2 \cdot G \cdot M}{r}}$

**Environmental Science:**

- LD50 (Median Lethal Dose): Dose that kills 50
- BOD (Biological Oxygen Demand): Measurement of organic pollution in water bodies
- $I = \text{PAT}$: Environmental Impact ($I$) = Population ($P$) $\times$ Affluence ($A$) $\times$ Technology ($T$)

**Mathematics:**

- Pythagorean Theorem: $a^2 + b^2 = c^2$
- Quadratic Formula: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Euler's Formula: $e^{i\pi} + 1 = 0$

- Fundamental Theorem of Calculus: $\int_a^b f(x)\,dx = F(b) - F(a)$

**Astrophysics:**

- Chandrasekhar Limit: Mass limit for white dwarfs' stability
- Schwarzschild Radius: $r_s = \frac{2 \cdot G \cdot M}{c^2}$
- Hubble Parameter: $H = \frac{dV}{dR} \,/\, V$