Guided Symbolic GPT

Amin Ravanbakhsh

December 2025

Abstract

Symbolic regression aims to discover interpretable mathematical expressions that describe the relationship between input and target variables, a task traditionally performed manually in scientific research. In this work, we introduce *Guided Symbolic GPT*, an extension of Symbolic GPT that incorporates guiding tokens encoding mathematical properties such as symmetry and convergence. These tokens provide structural information that conditions generation and improves robustness and interpretability. **Our contribution is the use of guiding tokens to inject mathematical properties into language-model-based symbolic regression**, and **our results show that Guided Symbolic GPT consistently outperforms the baseline Symbolic GPT on symbolic regression benchmarks while using the same model architecture**.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Symbolic Regression

Symbolic regression is the task of identifying a function that captures the relationship between one or more *input variables* and one or more *target variables*. This involves searching over the space of closed-form mathematical expressions composed of operators and functions such as $\sin, \cos, \log, \exp$, and others. The ideal expression must be sufficiently complex to accurately model the target variables given the input variables, yet simple enough to remain robust in the presence of noise.

The motivation for symbolic regression arises from the automated discovery of physical laws, a process historically carried out manually Koyré [2013]. Physical laws are typically characterized by a limited number of variables and constants, making them well-suited to symbolic regression. Importantly, symbolic regression differs fundamentally from neural network–based approaches: whereas neural networks provide black-box approximations, symbolic regression seeks interpretable, closed-form representations that can often yield mechanistic insights. Furthermore, symbolic regression can incorporate dimensional constraints of variables, enabling physically meaningful formulations. These advantages have already facilitated applications in diverse domains, including the discovery of ecological dynamics Chen et al. [2019], the analysis of material stability He and Zhang [2021], mortality prediction after major surgery Arina et al. [2025], and the identification of inconsistencies in established physical models Kaheman et al. [2019].

Formally, symbolic regression is the problem of finding a mathematical function f defined over the input variables $\mathbf{x} = [x_1, \ldots, x_d]$ given a dataset $D = (\mathbf{x_i}, y_i)_{i=1}^{n}$, such that $f(\mathbf{x}_i) = y_i$, or $f(\mathbf{x}_i) \approx y_i$ in the presence of noise. Moreover, the learned function f should generalize to a test dataset $D' = (\mathbf{x}j, y_j)j = 1^m$ generated by the same underlying process. A further desirable property is parsimony: f should be as simple as possible while still adequately fitting the data.

Unlike conventional machine learning methods, symbolic regression is inherently more

challenging. The task requires exploring an immense, partially discrete, and often hierarchically nested space of candidate functions. As a result, symbolic regression is known to be NP-hard Virgolin and Pissis [2022].

The key contributions of our approach are summarized as follows:

- Introduces **Guided Symbolic GPT**, extending the SymbolicGPT transformer framework with *guiding tokens* that encode mathematical properties such as symmetry and convergence.
- Injects *domain-informed meta-information* directly into the model's generative process, contrasting with scaling-focused approaches.
- Improves model **interpretability** and **robustness** without increasing model capacity.

In this thesis, **Chapter 2** reviews related work in transformer-based and genetic-algorithm–based approaches to symbolic regression. **Chapter 3** presents our methodology, detailing the architecture of Guided Symbolic GPT and the supporting technical components. **Chapter 4** reports experimental results evaluating the performance of Guided Symbolic GPT across a range of tasks. **Chapter 5** offers a discussion of the implications, limitations, and broader impact of our approach. Finally, **Chapter 6** concludes the thesis and highlights promising directions for future research.

Chapter 2

Related works

Symbolic regression (SR) aims to automatically discover interpretable mathematical expressions that describe relationships among variables. Early approaches were largely based on genetic programming (GP) McKay et al. [1995], Augusto and Barbosa [2000], Korns [2013], where candidate expressions evolve through operators such as mutation and crossover. A landmark study demonstrated that evolutionary SR can rediscover fundamental physical laws directly from data, establishing SR as a tool for automated scientific discovery Schmidt and Lipson [2009]. Subsequent GP variants explored probabilistic search strategies and hybrid optimization schemes Gong et al. [2022].

To address scalability and data efficiency, alternative formulations recast SR as a sparse regression problem. Methods such as SINDy identify parsimonious governing equations using compressed-sensing principles Kaiser et al. [2018]. Related techniques have been applied across physics and ecology to recover interpretable dynamical systems Chen et al. [2019], Quade et al. [2016]. In parallel, approaches incorporating physical priors and dimensional constraints, including AI Feynman and the Symbolic Physics Learner, improved accuracy and interpretability in low-data regimes Udrescu and Tegmark [2020], Sun et al. [2022]. Unit-consistency constraints were later integrated into SR frameworks to further guide physics-informed equation discovery Tenachi et al. [2023].

Neural symbolic regression replaced heuristic search with learned inductive priors. Early methods used neural networks to bias symbolic search or parameterize expression trees Li et al. [2019], Mundhenk et al. [2021], Petersen et al. [2019]. Learned guidance was shown to accelerate convergence and improve generalization in neural-guided GP, deep symbolic regression models, and reinforcement-based systems Mundhenk et al. [2021], Xu et al. [2023], Tian et al. [2024]. Transformer architectures, originally introduced for sequence modeling Vaswani [2017], later enabled direct sequence generation of symbolic expressions from data in SR settings.

Large-scale generative transformers marked a turning point for SR. SymbolicGPT demonstrated that pretrained language models can generate compact and interpretable formulas without explicit evolutionary search Valipour et al. [2021]. Later work focused on

scaling behavior, expression validity, and sample efficiency Biggio et al. [2021], Kamienny et al. [2022], Vastl et al. [2024]. Alternative designs explored continuous optimization strategies for symbolic regression Liu et al. [2023], Holt et al. [2023], Scholl et al. [2024], while generative flow networks provided a complementary framework for structured symbolic generation Li et al. [2023]. Sequential decision-making was further incorporated through neural-guided dynamics models and transformer-based planning approaches Li et al. [2024], Shojaee et al. [2023].

Benchmarking and meta-learning efforts have shaped the modern SR landscape. Standardized datasets support reproducible evaluation of generalization Olson et al. [2017], Matsubara et al. [2022], while early benchmarking efforts emphasized interpretability and domain relevance Zhang et al. [2012], with recent extensions providing more systematic and reproducible evaluations de Franca et al. [2024]. Recent surveys highlight that interpretability, physical consistency, and computational scalability remain open challenges Makke and Chawla [2024].

This work builds on transformer-based symbolic regression by introducing **Guided Symbolic GPT**, which incorporates guiding tokens encoding mathematical properties such as symmetry and convergence. Rather than increasing model capacity or focusing solely on scaling, domain-informed meta-information is injected directly into the generative process. This design improves interpretability and robustness and aligns with emerging work on controllable and explainable neural symbolic regression Bendinelli et al. [2023], Li et al. [2022], Virgolin and Pissis [2022]. By combining symbolic reasoning with learned inductive priors, Guided Symbolic GPT advances reliable and interpretable automated scientific discovery.

Chapter 3

Methodology

3.1 Data

The performance of Symbolic GPT is directly influenced by the quality and diversity of its training data. To promote robustness across a wide range of input domains and to mitigate overfitting to specific sets of equations, we designed a specialized data generation pipeline.

3.1.1 Computation Graph

At the core of this pipeline lies the *Computation Graph*, which serves as the foundation for all symbolic representations. The graph is structured as a binary tree, where each node corresponds to either an operator (unary or binary), a variable placeholder, a constant placeholder, or an empty placeholder. Representing expressions as graphs, rather than sequences, enables richer analyses through algorithms such as breadth-first search (BFS) or depth-first search (DFS).

The binary tree structure provides a simple yet powerful and extensible framework, since most mathematical and physical functions can be decomposed into unary and binary operations. Notably, any unary operator can be reformulated as a binary operator by assigning an empty placeholder as its right child. This design choice ensures both generality and consistency in the representation of symbolic expressions.

3.1.2 Metadata Generation

In this component, we introduce *mathematical property tokens*, which are later utilized by Guided Symbolic GPT to produce more robust and reliable outputs. These properties are well established in the mathematical literature and are commonly used by experts to evaluate symbolic expressions. By incorporating such properties, the model gains additional structural information about the underlying datasets, enabling more principled

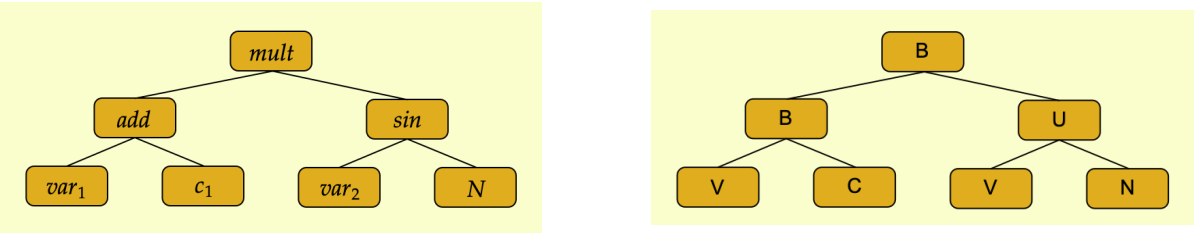


Figure 3.1: Left: Computation Graph of $(var_1 + C_1) \times sin(var_2)$. Right: Structure of the Computation Graph

assessments of the generated outputs.

In a general set up let $f : R^d \to R$ and the input to the function is $\mathbf{x} = (x_1, x_2, ..., x_d)$, and for all except i (x_i) we use $\mathbf{x}_{-i}$.

1. Symmetric

$$f(\mathbf{x}_{-i}, x_i) = f(\mathbf{x}_{-i}, -x_i) \quad \text{for all admissible } (\mathbf{x}_{-i}, x_i).$$

2. Odd Symmetric

$$f(\mathbf{x}_{-i}, x_i) = -f(\mathbf{x}_{-i}, -x_i) \quad \text{for all admissible } (\mathbf{x}_{-i}, x_i).$$

3. Linear Relation

$$f(\mathbf{x}_{-i}, x_i) = a(\mathbf{x}_{-i})\, x_i \quad \text{for some function } a$$

4. Convergence to Zero at Infinity

$$\lim_{\|x_i\| \to \infty} f(x) = 0.$$

5. Monotonic increasing magnitude: There exists $R \geq 0$ such that for $|x_i| \geq R$, $|f|$ is nondecreasing in $|x_i|$ with $\mathbf{x}_{-i}$ fixed.

6. Variable Separability

$$f(\mathbf{x}_{-i}, x_i) = g(\mathbf{x}_{-i})\, h(x_i) \quad \text{for some functions } g \text{ and } h.$$

7. Swapping Variable

$$f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_d) = f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_d)$$

Following algorithms demonstrate how we generate these meta-information based on the equations.

Algorithm 1 Symmetric_Check (multi-variable pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: List of Booleans indicating if f is symmetric in each variable
1: Initialize $diff\_threshold \leftarrow 10^{-3}$, $n\_check \leftarrow 30$, $max\_try \leftarrow 50$
2: **for** each variable var_i **do**
3: Collect up to n_check valid samples by randomly assigning inputs and evaluating f
4: For each sample, negate var_i and evaluate f, then compare outputs within $diff\_threshold$
5: If all samples pass, mark $check\_list[i] \leftarrow True$
6: **end for**
7: **return** $check\_list$

Algorithm 2 Odd_Function_Check (multi-variable pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: List of Booleans indicating if f is odd in each variable
1: Initialize $diff\_threshold \leftarrow 10^{-3}$, $n\_check \leftarrow 30$, $max\_try \leftarrow 50$
2: **for** each variable var_i **do**
3: Collect up to n_check valid samples by randomly assigning inputs and evaluating f
4: For each sample, negate var_i, evaluate f, and check $out \cdot counter\_out \leq 0$ and $|out + counter\_out| < diff\_threshold$
5: If all samples pass, mark $check\_list[i] \leftarrow True$
6: **end for**
7: **return** $check\_list$

Algorithm 3 Linearity_in_Variable_Check (No Intercept)

Require: Function $f(x_1, \ldots, x_n)$, number of variables n_{var}
Ensure: List `isLinear`[1:n_{var}] indicating whether f is linear in each variable (i.e., $f(x_{-i}, x_i) = a(x_{-i})x_i$)

Set $m \leftarrow 30$ {number of random draws of x_{-i}}
Set $k \leftarrow 5$ {number of grid points for x_i}
Set $R2_{\text{threshold}} \leftarrow 0.995$
for $i = 1$ to n_{var} **do**
  Initialize empty list $R2\_values$
  for $j = 1$ to m **do**
    Randomly sample x_{-i} from the domain
    Sample a small grid $\{x_i^{(1)}, \ldots, x_i^{(k)}\}$ from the domain
    for $t = 1$ to k **do**
      $y_t \leftarrow f(x_{-i}, x_i^{(t)})$
    end for
    Fit least-squares slope $\hat{a}$ to $y_t \approx \hat{a}\, x_i^{(t)}$ **(no intercept)**
    Compute and record R^2 for the no-intercept fit; append to $R2\_values$
  end for
  $\tilde{R}2 \leftarrow \text{median}(R2\_values)$
  if $\tilde{R}2 \geq R2_{\text{threshold}}$ **then**
    `isLinear`[i] $\leftarrow$ `True`
  else
    `isLinear`[i] $\leftarrow$ `False`
  end if
end for
return `isLinear`

Algorithm 4 Converge_to_Zero_Check (pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: List of Booleans indicating if f converges to zero for each variable

Initialize parameters: step sizes, limits, $n\_check \leftarrow 30$, $max\_try \leftarrow 50$, threshold
for each variable var_i **do**
  Collect up to n_check valid samples by adding initial large step to var_i and evaluating f
  For each sample, take multiple incremental steps, record $(|var_i|, |f|)$, and check monotonic decrease
  Check a final large step to ensure f is below threshold
  If all samples pass, mark $check\_list[i] \leftarrow True$
end for
return $check\_list$

Algorithm 5 Monotonic_Increasing_Magnitude_Check (pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: List of Booleans indicating if $|f|$ increases monotonically with each variable
1: Initialize parameters: step range $[1, 3]$, $lim\_n\_steps \leftarrow 10$, $n\_check \leftarrow 30$, $max\_try \leftarrow 50$
2: **for** each variable var_i **do**
3: Collect up to n_check valid samples by randomly initializing inputs and evaluating f
4: For each sample, take multiple increasing steps in var_i, record $(|var_i|, |f|)$ pairs
5: Sort values by $|var_i|$ and check that $|f|$ strictly increases, allowing small fluctuations near zero
6: If all samples confirm monotonic increase, mark $check\_list[i] \leftarrow True$
7: **end for**
8: **return** *check_list*

Algorithm 6 Variable_Separability_Check (pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: List of Booleans indicating if each variable is separable
1: Initialize thresholds: $zero\_threshold \leftarrow 10^{-2}$, $d\_sep\_threshold \leftarrow 2 \times 10^{-3}$, $n\_check \leftarrow 30$, $max\_try \leftarrow 50$
2: **for** each variable var_i **do**
3: Collect up to n_check valid samples by generating two random input sets
4: For each sample, swap the values of var_i between the two inputs and evaluate all combinations
5: Compute $d\_sep = |out_1 - out\_comb_1 \cdot out\_comb_2/out_2|/|out_1|$ and compare with $d\_sep\_threshold$
6: If all samples satisfy the threshold, mark $check\_list[i] \leftarrow True$
7: **end for**
8: **return** *check_list*

Algorithm 7 Swap_Variable_Check (pseudocode summary)

Require: Function $f(x_1, x_2, \ldots, x_n)$, number of variables n_var, constants
Ensure: Dictionary indicating if swapping each pair of variables preserves f
1: Initialize $n\_check \leftarrow 30$, $max\_try \leftarrow 50$, $zero\_threshold \leftarrow 10^{-3}$
2: Initialize $swap\_dict[i, j] \leftarrow False$ for all $i < j$
3: **for** each variable pair (var_i, var_j) with $i < j$ **do**
4: Collect up to n_check valid samples by generating random inputs
5: For each sample, swap var_i and var_j and evaluate f before and after swap
6: Compare outputs using $zero\_threshold$; record True if all samples satisfy threshold
7: If all samples pass, set $swap\_dict[i, j] \leftarrow True$
8: **end for**
9: **return** *swap_dict*

3.1.3 Huge Random Formula Set Generation Pipeline

This part outlines the process for generating a large set of random computation graphs.

1. **Structure Generation and Organization**: We define graph structures that serve as templates for computation graphs. Nodes within these structures act as placeholders for binary and unary operators, variable placeholders, constant placeholders, and null placeholders.

2. **Structure Selection**: To ensure a balanced dataset, we constrain sampling based on the number of variable placeholders and the depth of the binary tree, maintaining an upper threshold for each combination.

3. **Skeleton Sampling**: For a given depth and number of variable placeholders, we generate up to 100 skeletons by mapping binary (B) and unary (U) placeholders to corresponding operators.

4. **Variable Assignment**: Each structure is instantiated into a formula of length n by replacing variable (V) placeholders with variables, ensuring all variables appear at least once. Any remaining V placeholders are randomly assigned variables.

5. **Redundant Tree Filtering**: We eliminate redundant computation graphs that contain trivial or simplifiable substructures, such as:

$$\log(\exp(\cdot)), \quad \text{pow}_2(\sqrt{\cdot}), \quad \text{neg}(\text{neg}(\cdot)), \ldots$$

3.2 Model

This model consists of two primary components: GPT and PointNet. The PointNet module transforms the tabular dataset into a point embedding that encapsulates the information contained within the original data. This embedding is then combined with the embedding generated by GPT from the previous tokens. The resulting representation is used by the model to produce the logits for the next token. Finally, by sampling from these logits, the model generates the most probable next token.

As illustrated in Figure 3.3, the tabular dataset is processed such that each row is transformed into an embedding through a neural network. Subsequently, a Global Max layer is applied to capture the maximum values across all embeddings, resulting in a unified representation of the dataset.

Guided Symbolic GPT builds upon Symbolic GPT Valipour et al. [2021], with the key distinction being the introduction of *guiding tokens* alongside equation tokens. These guiding tokens encode additional mathematical information, which enhances the model's ability to generate more accurate and reliable predictions.

As illustrated in Figure 3.4 and 3.2, the architecture remains largely the same, with the guiding tokens being the only major addition. These tokens represent mathematical properties—such as symmetry, convergence to zero at infinity, and others—that serve as inductive biases. By explicitly incorporating such properties into the tokenization process, Guided Symbolic GPT achieves more robust and interpretable predictions compared to its predecessor.

The tokenizer used in this model is trainable and is trained specifically on the dataset employed in this study. In addition, mathematical property tokens are incorporated as part of the token set. Overall, the model's vocabulary comprises approximately 3,434 tokens.

As shown below, each input sequence is constructed with a fixed length of 40 tokens. The sequence starts with a guiding segment consisting of 20 tokens, which includes the guiding tokens encoding mathematical properties as well as padding tokens when necessary. This guiding segment is followed by the ground-truth equation tokens and their corresponding padding, occupying the remaining 20 tokens of the sequence.

Example:

```
[META\_START] [EVEN:x1] [ODD:x2] [SEP:x1] [SWAP:x1\_x3] [LIN:x2]
[META\_END] [Gpad] [Gpad] [EQUATION\_START] [sin] [x1] [+] [cos] [x2] [*] [x3]
[EQUATION\_END] [pad] [pad] [pad]
```

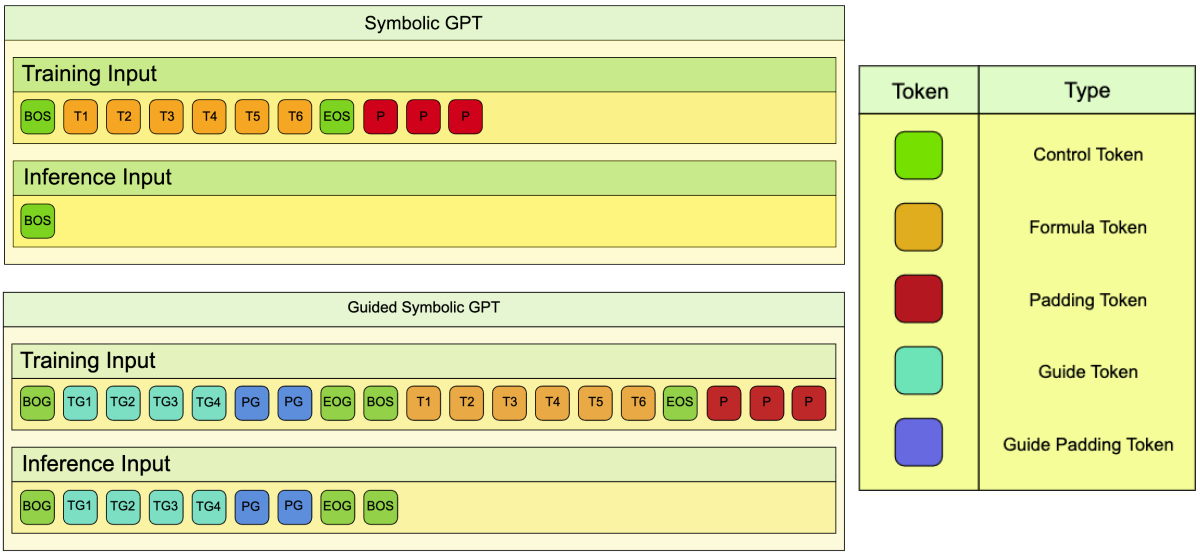
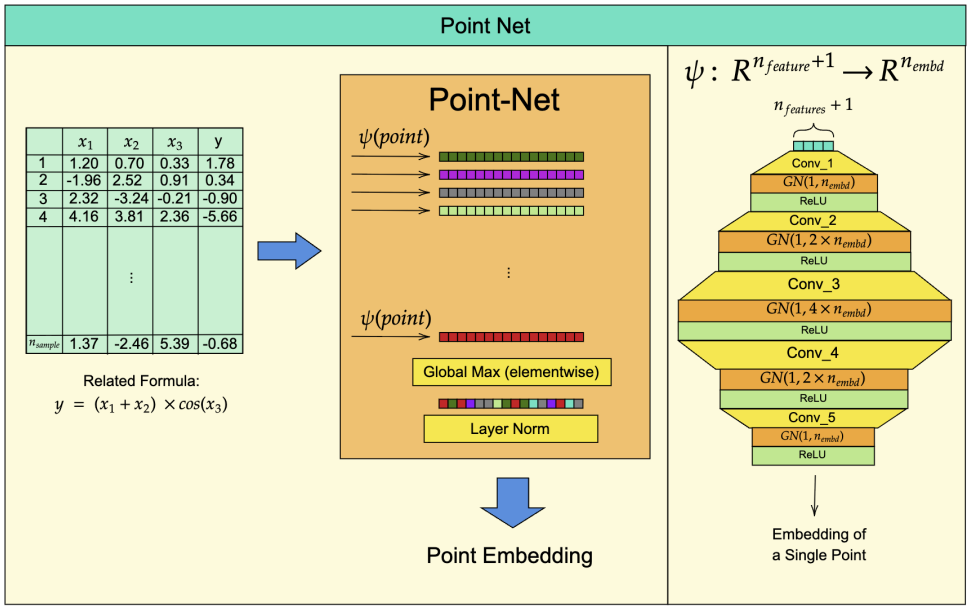


Figure 3.2: Token-level comparison between Symbolic GPT and Guided Symbolic GPT.

3.2.1 Training Details

To train the model, we employed a neural architecture comprising approximately 45 million parameters. The embedding dimension was set 512. The transformer component



	x_1	x_2	x_3	y
1	1.20	0.70	0.33	1.78
2	-1.96	2.52	0.91	0.34
3	2.32	-3.24	-0.21	-0.90
4	4.16	3.81	2.36	-5.66
		⋮		
n_{sample}	1.37	-2.46	5.39	-0.68

Figure 3.3: Point Net

consisted of 8 layers and 8 attention heads, with a dropout rate of 0.2. The attention block size was configured to 40 tokens, divided equally between 20 guiding tokens and 20 equation tokens. The model was trained for 10 epochs using a learning rate of 0.0001. Find the details in table 3.1

Operator set	$\{+, -, \times, \div, \sin, \cos, \exp, \log, gaussian, sqrt, inverse\}$
Variables	$d \in \{1, 2, 3\}$
Tree depth	max depth = 5
Constants	uniform in $[-3, 3]$
Input sampling	each $x_i \sim \mathcal{U}[-5, 5]$
Noise	none / Gaussian $\mathcal{N}(0, \sigma^2)$, $\sigma = 0.01$
#expressions (train/val/test)	70k / 15k / 15k
Seeds	data seed = 42, init seed = 42
Batch & optimizer	batch = 512, Adam (lr = 1×10^{-4})
Epochs	10
Embedding dimension	512
#Layer & #Attention head	8,8
Points per expression	500 points
Token budget	20 meta + 20 equation; truncation policy as above

Table 3.1: Summary of the dataset generation and training configuration used in our experiments.

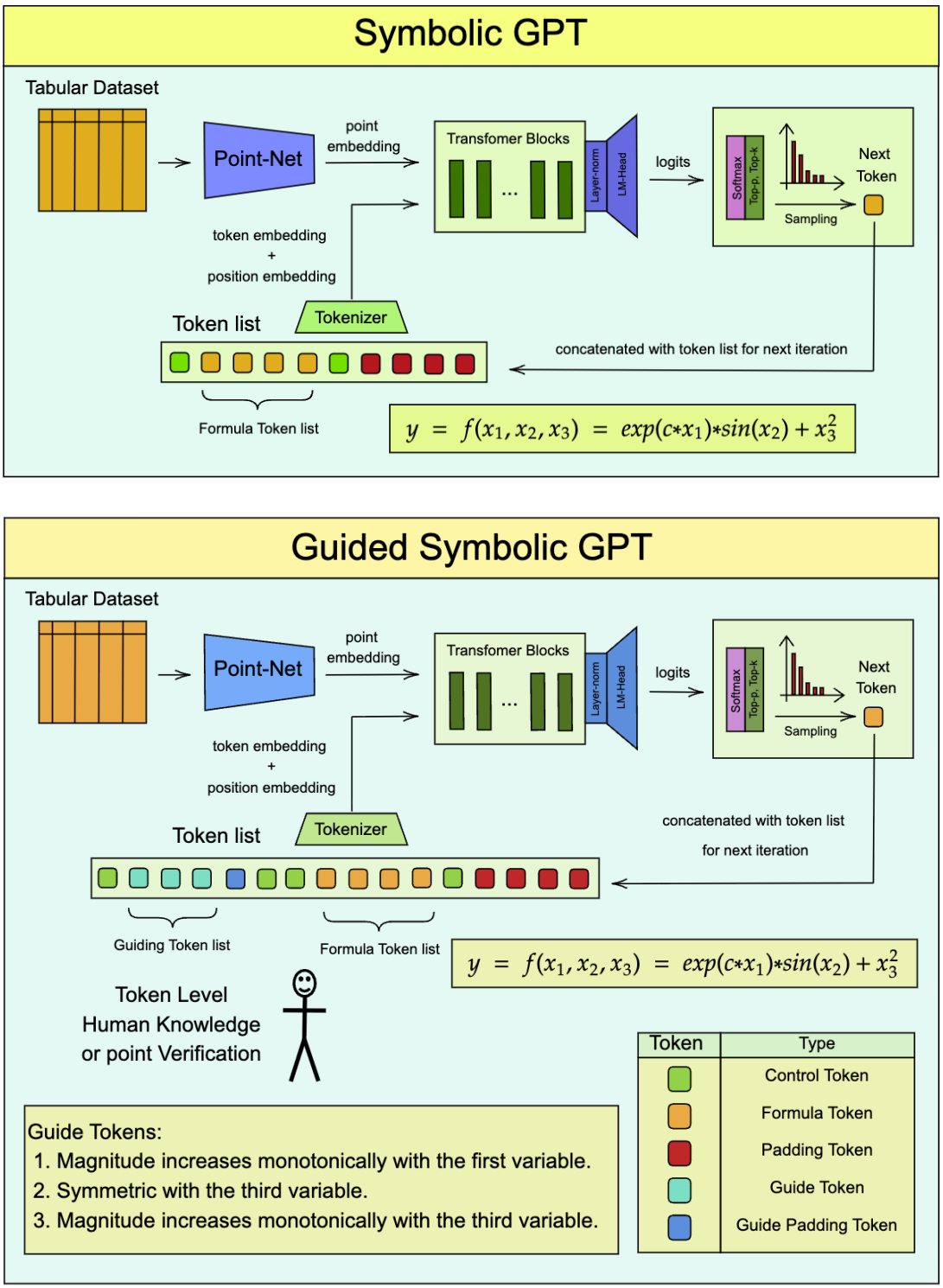


Figure 3.4: Symbolic GPT vs Guided Symbolic GPT

Chapter 4

Results

Our experiments demonstrate that *Guided Symbolic GPT* consistently outperforms *Symbolic GPT* under fair and controlled conditions. Model performance is evaluated using the coefficient of determination (R^2-score), defined as

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

where

- y_i denotes the true values,
- $\hat{y}_i$ represents the predicted values, and
- $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the mean of the true values.

To ensure a fair comparison, both models used in the experiments were constrained to the same scale, each containing approximately 45 million parameters. This design choice guarantees that improvements in predictive accuracy can be attributed to the inclusion of guiding tokens, rather than differences in model capacity or computational power.

Model	$\mathbf{R^2 \geq 0.99}$	$\mathbf{R^2 \geq 0.95}$	$\mathbf{R^2 \geq 0.90}$
Baseline (no tokens)	59 ± 1	61 ± 1	66 ± 2
Guided (all tokens)	77 ± 1	79 ± 2	84 ± 2
Guided (random tokens)	70 ± 2	74 ± 3	80 ± 3
Guided (noisy tokens)	64 ± 2	70 ± 3	74 ± 4

Table 4.1: Hit rates by R^2 threshold (mean±SD over seeds). Random/noisy tokens test robustness.

To illustrate the effectiveness of the proposed approach, we present several representative examples. Table 4 compares the performance of the baseline Symbolic GPT model with that of the proposed **Guided Symbolic GPT**, which incorporates mathematical property tokens as additional inputs. These guiding tokens encode structural information

such as symmetry and convergence, enabling the model to generate expressions that are not only more accurate but also more consistent with underlying mathematical principles. The results demonstrate that incorporating these tokens leads to measurable improvements in predictive accuracy and interpretability.

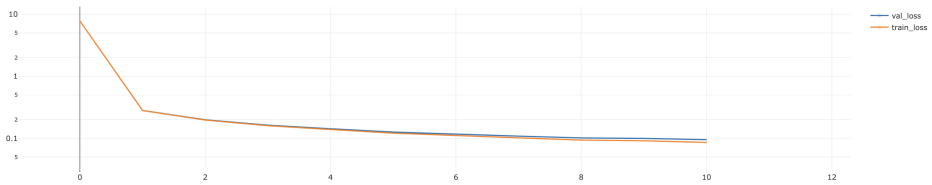


Figure 4.1: Train and validation loss in each step

Target Formula	**Math Properties**	**Symbolic GPT**	**Guided Symbolic GPT**
$x_1x_2x_3$	Linear in x_1, x_2, x_3	$x_1x_2\sin(x_3)$	$x_1x_2x_3$
$\frac{x_1x_2^2}{x_3}$	Convergent to zero in x_3, Linear in x_1, Symmetric in x_2	$x_1\cos(x_2) - x_3$	$\frac{x_1x_2^2}{x_3}$
$x_1x_3(x_1 + x_2)$	Linear in x_3	$x_1x_2 + x_1x_3$	$x_1x_3(x_1 + x_2)$
$\cos(x_1)x_2 + c$	Symmetric in x_1	$\sin(x_1)x_2 + c$	$\cos(x_1)x_2 + c$

Table 4.2: Comparison between Symbolic GPT and Guided Symbolic GPT

Chapter 5

Discussion

Our results demonstrate that incorporating guiding tokens significantly improves symbolic regression performance by enabling the model to retain and exploit meta-information during generation. By encoding mathematical properties directly into the input sequence, guiding tokens introduce an explicit structural bias that improves robustness, consistency, and interpretability of the generated expressions. This added structure is particularly valuable in scientific settings, where interpretability is as critical as predictive accuracy. Since scientists and domain experts often possess prior knowledge about expected model behavior, the proposed approach provides a practical mechanism for injecting such expertise into large language models, leading to more reliable and scientifically meaningful predictions.

Despite these advantages, the current implementation is limited in scope. Only seven guiding tokens are supported, a restriction imposed by the available datasets and the data-generation pipeline used in this work. As a result, the model is evaluated on a relatively narrow set of mathematical properties, and the generality of the approach beyond these constraints remains to be explored.

Several concrete directions for future work emerge from these limitations. First, expanding the set of guiding tokens to represent a broader range of mathematical and physical properties would allow the framework to capture richer forms of prior knowledge. Second, developing automated or data-driven methods for constructing meta-information could reduce reliance on handcrafted guidance and improve scalability. Finally, extending the framework to support post-hoc or iterative refinement of generated expressions based on meta-information would enable adaptive correction and further improve reliability in scientific discovery tasks.

Chapter 6

Conclusion

In this work, we introduced Guided Symbolic GPT, an extension of Symbolic GPT that incorporates explicit mathematical meta-information through guiding tokens. By encoding properties such as symmetry and convergence directly into the input representation, the proposed framework enables language-model-based symbolic regression to generate expressions that are more robust, interpretable, and consistent with known structural constraints. Experimental results on symbolic regression benchmarks demonstrate that this guidance consistently improves performance over the unguided baseline while preserving the original model structure. While the current implementation is limited to a small, predefined set of guiding tokens and does not support iterative refinement based on meta-information, the results establish guided symbolic generation as an effective and extensible approach. Overall, this work highlights the value of integrating expert-informed meta-information into neural symbolic regression and points toward guided generation as a promising direction for reliable and interpretable scientific modeling.

Bibliography

Pietro Arina, Davide Ferrari, Nicholas Tetlow, Amy Dewar, Robert Stephens, Daniel Martin, Ramani Moonesinghe, Vasa Curcin, Mervyn Singer, John Whittle, et al. Mortality prediction after major surgery in a mixed population through machine learning: a multi-objective symbolic regression approach. *Anaesthesia*, 2025.

Douglas Adriano Augusto and Helio JC Barbosa. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian symposium on neural networks*, pages 173–178. IEEE, 2000.

Tommaso Bendinelli, Luca Biggio, and Pierre-Alexandre Kamienny. Controllable neural symbolic regression. In *International Conference on Machine Learning*, pages 2063–2077. PMLR, 2023.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. Pmlr, 2021.

Yize Chen, Marco Tulio Angulo, and Yang-Yu Liu. Revealing complex ecological dynamics via symbolic regression. *BioEssays*, 41(12):1900069, 2019.

Fabricio O de Franca, Marco Virgolin, M Kommenda, MS Majumder, M Cranmer, G Espada, L Ingelse, A Fonseca, M Landajuela, B Petersen, et al. Srbench++: Principled benchmarking of symbolic regression with domain-expert interpretation. *IEEE transactions on evolutionary computation*, 2024.

Chi Gong, Jordan Bryan, Alex Furcoiu, Qichang Su, and Rainer Grobe. Evolutionary symbolic regression from a probabilistic perspective. *SN Computer Science*, 3(3):209, 2022.

Mu He and Lei Zhang. Machine learning and symbolic regression investigation on stability of mxene materials. *Computational Materials Science*, 196:110578, 2021.

Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression. *arXiv preprint arXiv:2401.00282*, 2023.

Kadierdan Kaheman, Eurika Kaiser, Benjamin Strom, J Nathan Kutz, and Steven L Brunton. Learning discrepancy models from experimental data. *arXiv preprint arXiv:1909.08574*, 2019.

Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

Michael F Korns. A baseline symbolic regression algorithm. *Genetic Programming Theory and Practice X*, pages 117–137, 2013.

Alexandre Koyré. *The Astronomical Revolution: Copernicus-Kepler-Borelli*. Routledge, 2013.

Li Li, Minjie Fan, Rishabh Singh, and Patrick Riley. Neural-guided symbolic regression with asymptotic constraints. *arXiv preprint arXiv:1901.07714*, 2019.

Sida Li, Ioana Marinescu, and Sebastian Musslick. Gfn-sr: Symbolic regression with generative flow networks. In *NeurIPS 2023 AI for Science Workshop*, 2023.

Wenqiang Li, Weijun Li, Linjun Sun, Min Wu, Lina Yu, Jingyi Liu, Yanjie Li, and Songsong Tian. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Wenqiang Li, Weijun Li, Lina Yu, Min Wu, Linjun Sun, Jingyi Liu, Yanjie Li, Shu Wei, Yusong Deng, and Meilan Hao. A neural-guided dynamic symbolic network for exploring mathematical expressions from data, 2024. URL `https://arxiv.org/abs/2309.13705`.

Jingyi Liu, Weijun Li, Lina Yu, Min Wu, Linjun Sun, Wenqiang Li, and Yanjie Li. Snr: Symbolic network-based rectifiable learning framework for symbolic regression. *Neural Networks*, 165:1021–1034, 2023.

Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, 57(1):2, 2024.

Yoshitomo Matsubara, Naoya Chiba, Ryo Igarashi, and Yoshitaka Ushiku. Srsd: Rethinking datasets of symbolic regression for scientific discovery. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

Ben McKay, Mark J Willis, and Geoffrey W Barton. Using a tree structured genetic algorithm to perform symbolic regression. In *First international conference on genetic algorithms in engineering systems: innovations and applications*, pages 487–492. IET, 1995.

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

Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017. ISSN 1756-0381. doi: 10.1186/s13040-017-0154-4. URL `https://doi.org/10.1186/s13040-017-0154-4`.

Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

Markus Quade, Markus Abel, Kamran Shafi, Robert K. Niven, and Bernd R. Noack. Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1), July 2016. ISSN 2470-0053. doi: 10.1103/physreve.94.012214. URL `http://dx.doi.org/10.1103/PhysRevE.94.012214`.

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

Philipp Scholl, Katharina Bieker, Hillary Hauger, and Gitta Kutyniok. Parfam – (neural guided) symbolic regression based on continuous global optimization, 2024. URL `https://arxiv.org/abs/2310.05537`.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression, 2023. URL `https://arxiv.org/abs/2303.06833`.

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

Wassim Tenachi, Rodrigo Ibata, and Foivos I Diakogiannis. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *The Astrophysical Journal*, 959(2):99, 2023.

Yuan Tian, Wenqi Zhou, Hao Dong, David S. Kammer, and Olga Fink. Sym-q: Adaptive symbolic regression via sequential decision-making, 2024. URL `https://arxiv.org/abs/2402.05306`.

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

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

Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access*, 2024.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Marco Virgolin and Solon P Pissis. Symbolic regression is np-hard. *arXiv preprint arXiv:2207.01018*, 2022.

Yilong Xu, Yang Liu, and Hao Sun. Rsrm: Reinforcement symbolic regression machine. *arXiv preprint arXiv:2305.14656*, 2023.

Ying Zhang, Pham Minh Duc, Oscar Corcho, and Jean-Paul Calbimonte. Srbench: a streaming rdf/sparql benchmark. In *The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I 11*, pages 641–657. Springer, 2012.