# README: Rapid Equation Discovery with Multimodal Encoders

**Gregory Kang Ruey Lau**[*]
National University of Singapore
CNRS@CREATE
greglau@comp.nus.edu.sg

**Yue Ran Kang**[*]
National University of Singapore
kyueran@u.nus.edu

**Zi-Yu Khoo**
National University of Singapore
ziyukhoo@nus.edu.sg

**Apivich Hemachandra**
National University of Singapore
apivich@nus.edu.sg

**Ruth Wan Theng Chew**
National University of Singapore
rwch@nus.edu.sg

**Bryan Kian Hsiang Low**
National University of Singapore
CNRS@CREATE
lowkh@comp.nus.edu.sg

## Abstract

Discovering scientific laws or interpretable equations from data rapidly is important in many setting, such as decision-making in time-sensitive high-stake scenarios or applications involving interactive or iterative experimentation such as in scientific or machine learning workflows. However, existing methods typically require long computational time to achieve good performance and have to run from scratch for each dataset. Recent methods that use pre-trained foundation models for faster inference also suffer from performance limitations and require large training datasets. In this work, we propose README, a framework for rapid equation discovery that can generate performant, interpretable equations from limited, noisy data in just a few seconds, and requires significantly less training data compared to past foundation model approaches. We achieve this by being the first to (1) work with image representations of datasets to efficiently capture their key properties, (2) combine the capabilities of open-sourced pre-trained text and image encoders to produce an informative SR embedding space, and (3) develop a novel Grey Wolf Optimizer with Bayesian Optimization (GWOBO) algorithm to rapidly optimize for the best symbolic expression within seconds. We empirically show that README outperforms benchmarks on a wide range of realistic datasets, including real experimental data from various domains and noisy video-extracted dynamics.

## 1 Introduction

In many scientific and industrial settings, obtaining interpretable symbolic expressions that describe systems accurately is a critical objective. For example, symbolic representation of physical phenomena in areas such as climate science [16, 2], material science [50, 51], and robotics [52, 33] are important in building scientific understanding, and interpretable symbolic expressions describing industrial processes and systems can help in high-stakes decision-making scenarios [39] and applica-

[*]Equal contribution.

tions in aerospace engineering [7], electrical systems [4] and healthcare [49, 12], where verifiability and human oversight are often required.

Symbolic regression (SR) methods aim to achieve automated discovery of the symbolic expressions that best approximate a given dataset, which is a challenging problem given the large search space of possible expressions [48]. However, while existing methods such as genetic programming-based algorithms [29] can generate good approximations, they are typically computationally intensive and slow to converge [5], and suffer from high sensitivity to hyperparameters and the basis function choices [35]. To address these issues, some works have proposed pre-training transformer-based SR models on large corpora of data, so as to amortize computational cost and enable faster inference [46, 21]. These include approaches using CLIP-based [37] multi-modal architectures trained on symbolic expressions and numerical data that could be used for candidate generation with genetic programming-based SR methods [31, 28, 42]. However, they require large datasets and computational time to train the models from scratch.

Importantly, most of the past works have not emphasized low-latency requirements where the time constraint for accurate symbolic expression is *in seconds*, not minutes or hours. The few methods for fast SR [30] tend to have their performance degrade for more realistic, noisy data. Hence, SR remains challenging in interactive, real-time or iterative scenarios, potentially limiting its utility in applications such as adaptive scientific experimentation and close to real-time decision making in high-stakes environment.

In this work, we have identified three insights to achieve performant rapid equation discovery. First, rather than working with raw numerical data, *image representations of numerical data* can aid SR. Humans use plots to quickly extract key trends and identify candidate equations. A similar approach might be adapted for multi-modal large language models. Images, even complex human-uninterpretable plots, can efficiently summarize mathematical trends with many variables while remaining readable to well-trained models, enabling better candidate equations generation.

Second, *existing pre-trained image and text encoders can be leveraged to efficiently build foundation models for SR*, given plots and symbolic equations. Building on rapidly improving open-sourced image and text encoders rather than separately training SR-specific models from scratch, can produce better models with less data and computational resources. These encoders may have been pre-trained to extract relevant features that are useful for SR (e.g., shape features in plots for the image encoder or math operator relationships for the text encoder), and hence only require fine-tuning with a small amount of data to become effective.

Third, to rapidly optimize for the symbolic equation that best approximates a dataset with desired properties (e.g., complexity), we consider whether *query-efficient approaches such as Bayesian Optimization (BO) could be used to significantly speed up* the search process, when used with SR foundation models. BO methods [14] allows for reduced calls to expensive fit procedures, which is a natural combination with population-based algorithms such as Grey Wolf Optimizer (GWO) [32] to enable rapid equation discovery.

Combining these insights, we propose README (<u>R</u>apid <u>E</u>qu<u>a</u>tion <u>D</u>iscovery with <u>M</u>ultimodal <u>E</u>ncoders), a framework for SR that uses (1) an informative, compressed image representation of numerical data (Sec. 3.1); (2) an efficiently-trained transformer-based model built on top of pre-trained image and text encoders ($\sim 60\times$ less training data compared to past works) (Sec. 3.2); and (3) a novel combination of BO and GWO for a rapid, effective optimization process (Sec. 3.3), to (4) achieve state-of-the-art and robust SR results for challenging settings with realistic, noisy settings and tight time constraints ($\leq$ 10s) (Sec. 4).

## 2 Problem formulation

**SR inference phase.** Consider a target system that is governed by an underlying equation $y = f(x)$, where $y \in \mathbb{R}$, $x \in \mathbb{R}^n$, and $f(x)$ is a function that can be symbolically expressed as a composition of math operators. Given an inference dataset $\mathcal{D}$ consisting of a set of noisy $m$ observations $\{(x_i, \tilde{y}_i)\}_{i=1}^{m}$ where $\tilde{y}_i = y_i + \epsilon_i$, the SR task is to obtain a symbolic expression for the underlying function $f(x)$ that is the most accurate while prioritizing parsimonious expressions. Specifically, our accuracy goal is to find a symbolic expression $G^* \in \mathbb{G}$, where $\mathbb{G}$ is the space of all valid symbolic expressions consisting of symbolic representations of input variables and math operators for the task under consideration, that represents a function $g(x)$ with the maximum $R^2$ value [2] over a test dataset $\mathcal{D}_t$ generated from the same underlying phenomenon as $\mathcal{D}$.

In practice, during the inference phase we aim to achieve the best SR expression subjected to two additional desiderata. First, we prefer equations that are more parsimonious (i.e., $G$ that is less complex as evaluated by the number of nodes in its expression tree, details in Sec. 4.3.1) as they tend to be more interpretable, but there is typically a trade-off between the achievable accuracy and parsimony of the expression $G$. Hence, we will evaluate methods on the parsimony of their proposed expressions, and use accuracy-parsimony Pareto plots to analyze how well the methods balance this trade-off. Second, the SR methods should have low *inference runtime*, as many practical scenarios may have strict time budgets. We evaluate the methods based on fixed, short time budgets (e.g., $10 - 30s$) in our experiments (Sec. 4).

**Training phase for foundation models.** We consider the realistic setting where we can generate synthetic training datasets independently from inference phase data (i.e., $\{\mathcal{D}_i, F_i\}_i$ where $F_i$ are ground truth expressions of datasets $\mathcal{D}_i$) to train SR foundation models. Given the cost of high-quality data generation, another desiderata is for methods that require foundation model training to use as little training data as possible to achieve good performance (Table 1).

## 3 Method Overview

README framework consists of three key components:

1. **Data processing.** [Sec. 3.1] For both inference and training, we have a data processing step $\mathcal{P}$ that converts each dataset $\mathcal{D}_i$ from raw numerical data to a single image $I$, i.e., $\mathcal{P}(\mathcal{D}_i) \rightarrow I_i$. As explained later, the image representation has several key benefits over raw numerical data.
2. **Model architecture.** [Sec. 3.2] The processed data will then pass through the README model, which consists of a pair of pre-trained image $\mathcal{I}$ and text $\mathcal{T}$ encoders as well as a text decoder $\mathcal{W}$ that has been fine-tuned by a set of labeled training data $\{\mathcal{D}_i, F_i\}_i$ during the training phase. The README training process combines the feature extraction capabilities of the image encoder with the mathematical knowledge embedded in a pre-trained math text encoder, to obtain an informative embedding space $\mathbb{S}$ that the image encoder maps datasets to (i.e., $I(\mathcal{D}_i) \rightarrow s_i$, where $s_i \in \mathbb{S}$) for inference optimization.
3. **Inference optimization.** [Sec. 3.3] Given a dataset $\mathcal{D}$, we will use the trained README model to generate an initial candidate set, followed by our README optimization process to search for the best point $s^* \in \mathbb{S}$ that can be decoded to obtain the best symbolic expression $\mathcal{W}(s^*) \rightarrow G^*$ to fit on $\mathcal{D}$. The base optimizer used is the Grey Wolf Optimizer (GWO), though for ultra-rapid scenarios ($\leq$ 10s) we employ the novel Grey Wolf Optimizer with Bayesian Optimization (GWOBO).

### 3.1 Data processing: Working with images

Unlike existing SR methods, README works by first converting raw numerical data to images. This is inspired by humans' capabilities to more rapidly infer patterns and guess candidate symbolic expression skeletons (i.e., expression forms without specific numerical constants) for data by visualizing it, rather than just going through raw numerical data (e.g., the oscillatory curves of a 1D sin wave are clearly recognizable when visualized), though visualization and interpretation quickly become challenging for high dimensions.

However, the growing capabilities of Multi-modal Large Language Models (MLLMs) suggests that their image encoders may have powerful feature extraction capabilities developed from large-scale training on diverse image datasets that may also be useful in capturing relevant patterns from data plots. If so, it may still be viable to use images to summarize relevant information from high-dimensional data, and use them for SR. Such images may even not appear human-interpretable, but could possibly be effectively used by fine-tuned image encoders and customized decoders. Hence, in README, for both model training and inference, we map every dataset $\mathcal{D}_i$ to a corresponding image plot $I_i$ through a standardized data processing step $\mathcal{P}$.

[2] Note that $G^*$ is not unique if only the accuracy desiderata is considered (e.g., superfluous terms could be added to any expression to represent the same function $g(x)$), but the parsimony desiderata will mitigate this.

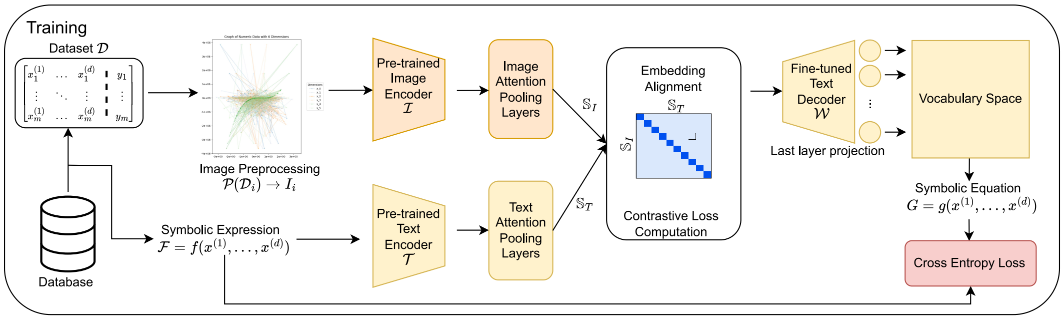

Training
Dataset $\mathcal{D}$
$\begin{bmatrix} x_1^{(1)} & \dots & x_1^{(d)} & y_1 \\ \vdots & \ddots & \vdots & \vdots \\ x_m^{(1)} & \dots & x_m^{(d)} & y_m \end{bmatrix}$
Image Preprocessing
$\mathcal{P}(\mathcal{D}_i) \to I_i$
Pre-trained Image Encoder $\mathcal{I}$
Image Attention Pooling Layers
$\mathbb{S}_I$
Embedding Alignment
$\mathbb{S}_T$
$\mathbb{S}_I$
Contrastive Loss Computation
Fine-tuned Text Decoder $\mathcal{W}$
Last layer projection
Vocabulary Space
Symbolic Equation
$G = g(x^{(1)}, \dots, x^{(d)})$
Cross Entropy Loss
Database
Symbolic Expression
$\mathcal{F} = f(x^{(1)}, \dots, x^{(d)})$
Pre-trained Text Encoder $\mathcal{T}$
Text Attention Pooling Layers
$\mathbb{S}_T$

Figure 1: An overview of the README architecture. See Sec. 3.1 and Sec. 3.2 for details.

To demonstrate this, we propose to start with the most basic plotting approach: in a single graph, we generate and overlay line plots for each dimension of the data. Fig. 1 shows an example plot. While the plots may not seem directly interpretable by humans and may also not uniquely represent a single symbolic expression (i.e., several expressions may correspond to the same plot), the general shape and features of the aggregated line plots, including information such as the axis magnitudes, provide sufficient details to significantly reduce the candidate search space and inform the optimization process for SR. This is similar to how humans can guess the expression skeleton but not necessarily the exact expression with its constants. In the README framework, the image is used only to narrow the search space for a more efficient optimization process to perform SR, as we will elaborate in Sec. 3.2 and Sec. 3.3.

Furthermore, this approach also helps to standardize the input format (i.e. 1 image) across datasets with varying number of datapoints and dimensions. In contrast, such variations in numerical data would lead to very different token lengths, leading to problems such as context length issues for past SR foundation model works [31] that constraints their applicability to datasets with larger sizes or dimensionality.

## 3.2 Model architecture and training process

A key innovation in our README model architecture component is our combination of the feature recognition capabilities of pre-trained image encoders with the mathematical knowledge contained in text encoders to generate an informative embedding space for SR. Note that the naive approach of directly using MLLMs for SR do not perform well (see App. F), hence past works [21, 31], have largely resorted to training transformers from scratch. Our approach allow us to obtain significantly better performance in SR with less training data. The README model architecture is adapted from the basic CLIP MLLM architecture [38]:

1. **Image encoder.** The image encoder $\mathcal{I} : \mathbb{I} \to \mathbb{S}_I$ maps each image plot $I_i = \mathcal{P}(\mathcal{D}_i) \in \mathbb{I}$ to its embedding vector representation $s_i \in \mathbb{S}$. Any general-purpose pre-trained encoder can be used, such as open-sourced `ViT` models [11] which are trained on diverse image data. These models have powerful image feature recognition capabilities (e.g., earlier model layers), and although their original embedding space $\mathbb{S}_I$ would not have the right structure for our SR tasks (see Sec. 4.2), they could be efficiently fine-tuned on our type of images from the data generation step.
2. **Text encoder.** The text encoder $\mathcal{T} : \mathbb{G} \to \mathbb{S}_T$ maps symbolic equations to its continuous embedding space $\mathbb{S}_T$. Crucially, we propose to use a text encoder pre-trained on math such as `MathBERT` [41] as it would contain inductive biases regarding symbolic expressions and have a relevant, well-structured embedding space $\mathbb{S}_T$ for SR that can guide the training of $\mathbb{S}_I$ for SR.
3. **Aligned embedding space.** The embedding spaces $\mathbb{S}_I$ and $\mathbb{S}_T$ are then aligned through joint contrastive learning, similar to the approach in CLIP [37]. We are primarily aiming for $\mathbb{S}_I$ to inherit the relevant SR structure from $\mathbb{S}_T$ and the training data, while preserving the image encoder's feature extraction capabilities. In Sec. 4.2, we provide illustrations that this happens in our experiments. Additional training and architecture details are in App. B.
4. **Text decoder.** The final component is a text decoder $\mathcal{W} : \mathbb{S}_I \to \mathbb{G}$ that maps points in the aligned embedding space $\mathbb{S}$ to symbolic expressions. To improve decoding performance for symbolic regression, we adopt an expression decoder [21], which overlays a Transformer atop the numeric

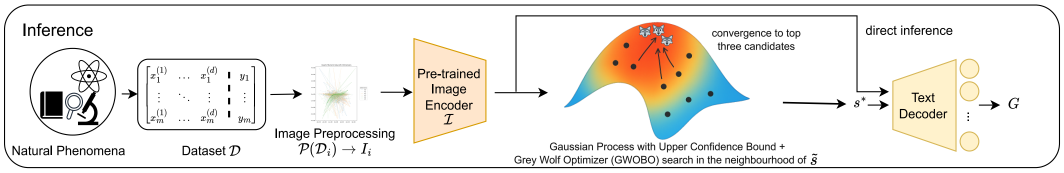

Inference
Natural Phenomena
Dataset $\mathcal{D}$
Image Preprocessing
$\mathcal{P}(\mathcal{D}_i) \rightarrow I_i$
Pre-trained Image Encoder $\mathcal{I}$
convergence to top three candidates
direct inference
Gaussian Process with Upper Confidence Bound +
Grey Wolf Optimizer (GWOBO) search in the neighbourhood of $\tilde{s}$
$s^*$
Text Decoder
$G$

Figure 2: README Inference Optimization process. See Sec. 3.3 for details.

encoder to translate encodings into symbolic expressions. Similar to prior work [31, 37], we first train the decoder with both encoders frozen, and then fine-tune all components jointly.

Leveraging the inductive biases of pretrained encoders, along with the more efficient image format, reduces data requirements while improving symbolic regression performance. As shown in Table 1, **README**, trained on far fewer synthetic pairs, still outperforms SNIP [31], which relies on numerical encoders. Ablation studies (Section B.2) further confirm the effectiveness of image encoders over numeric ones, demonstrating the value of inductive biases from graphs.

## 3.3 Inference optimization

In README, inference consists of two processes (Fig. 2).

**Inference decoding process.** We first convert the numerical dataset $\mathcal{D}$ into a plot image (Sec. 3.1), before mapping it through the image encoder to its embedding space representation $\tilde{s} = \mathcal{I}(\mathcal{P}(\mathcal{D}))$. In some cases, direct decoding using our text decoder $\mathcal{W}(\tilde{s})$ would already achieve a sufficiently good symbolic expression $\widetilde{G}$ for the dataset. However, README is designed to have the decoder just find the right symbolic expression skeleton, before doing 'constants optimization' via BFGS [13] similar to past works [21] where numerical constants in the expression may possibly be refined based on some metric (e.g., $R^2$) evaluated over $\mathcal{D}$. The decoding process is summarized in Algorithm 1.

**Inference optimization process.** The optimization process involves searching in the image embedding space $\mathbb{S}_I$ for the best point $s^*$ that would be decoded to the best symbolic expression $G^*$. For most settings, we do this by first generating a candidate population within a region around $s^*$, and using the Grey Wolf Optimizer (GWO) [32] to find $s^*$. Given the advantages from README's data processing and model, applying a vanilla GWO optimizer would typically already give SOTA performance (see Sec. 4 for details). However, under very tight time constraints, e.g., $\leq 10s$ where none except one of our benchmark methods manage to finish running, we need a faster optimization process. The bottleneck for GWO lies in the evaluation of entire population's fitness score, which requires running the decoding process to get a symbolic expression and compute its $R^2$.

Table 1: Comparison between README and SNIP models pretrained on different volumes of synthetic data. Mean R$^2$ Test Score shown is for real-world Physics-Informed dataset. Results for other datasets are in Appendix C.5.1, with further discussion on evaluation and metrics in Section 4.3.

| Model | Pretraining Data | Mean $R^2_{\text{Test}}$ |
|---|---|---|
| README | ∼1 million pairs | $0.984 \pm 0.004$ |
| SNIP | ∼60 million pairs | $0.883 \pm 0.091$ |

Hence, we propose a hybrid GWO algorithm (GWOBO), that employs Bayesian Optimization (BO) as a supporting subroutine for GWO to (1) train and provide a Gaussian Process (GP) surrogate model to model the fitness value ($R^2$) given any $s$, and (2) pick the top three wolfs ($\alpha, \beta, \gamma$) that will influence the exploration of the rest of the population (see Algorithm 2). Specifically, we iteratively run GWO and BO: after each iteration of GWO, we train the GP with past decoded points and run BO with an Upper Confidence Bound (UCB) acquisition function (see App. D for details) to pick the top three wolfs for the next GWO iteration. The top three wolfs will have their fitness score evaluated by the decoding process, while the rest of the population will have their fitness scores estimated using the GP trained from the BO process. In high-resource settings, the decoder can also be parallelized across multiple GPUs, which allows a larger candidate pool to be explored and improves performance.

## 4 Experimental results

### 4.1 Experimental setup

**Datasets.** We evaluate README on three datasets with varying characteristics: the Strogatz dataset consisting of synthetic data of 2-state dynamic models [44, 25], the CP3-Bench astrophysical dataset [45] of synthetic data based on cosmological equations with added noise and varying precision, and the Physics Informed dataset (PIED) from physics-informed experimental design [18] consisting of collated real-world, noisy experimental data. These datasets cumulatively provide 61 real-world regression problems (see App. C for details).

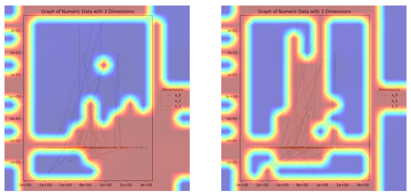
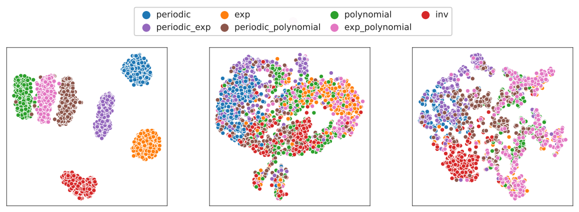


Figure 3: Image encoder attention rollout before (left) and after (right) training. Red indicates higher attention.



(a) Text (MathBERT) (b) ViT (pre-train) (c) ViT (trained)

Figure 4: t-SNE plots of text and image embeddings.

**Models.** We primarily used ViT-Base ([11]) as the image encoder and MathBERT [41] as the text encoder for our experiments, but also observed strong results with encoders from other model families (see App. B.2 for ablation results). The ViT-Base model as been pre-trained on large-scale image datasets for strong pattern recognition, while MathBERT has been trained on math texts, allowing it to better capture the structure and semantics of symbolic expressions.

**Benchmarks.** README is benchmarked against 9 algorithms, including Operon [9], Interaction-Transformation Evolutionary Algorithm (ITEA) [3], Genetic Programming Gene-pool Optimal Mixing Evolutionary Algorithm for Genetic Programming (GPGOMEA) [47], and Fast Function Extraction (FFX) [30] from SRBench [25], and Meidani et al.'s Symbolic-Numeric Integrated Pretraining (SNIP). The selected algorithms are efficient and effective, and represent a diverse range of SR approaches.

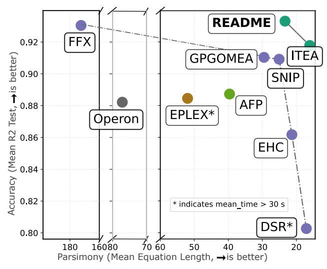
**Evaluation.** We evaluate algorithms over three metrics, as described in Sec. 2. First, we evaluate the **accuracy** of the generated expression $\tilde{G}$ and its associated function $g(x)$ by computing its $R^2$ over test data points. Secondly, expression **complexity** or *parsimony* is evaluated as the number of nodes in its associated expression tree. A smaller expression complexity, or a more parsimonious expression, is better. Lastly, wall-clock **time** is used a measure of the speed of the SR algorithm. This is measured as the time taken to train the algorithm. A smaller wall-clock time is better. All experiments are repeated with five random seeds.

### 4.2 Image encoder and embedding space structure

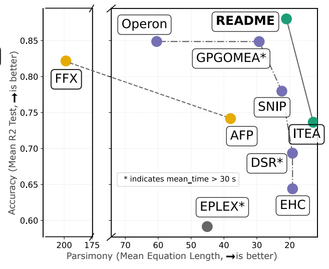
**Attention visualization.** We first analyze how the image encoder processes the input image plots, by visualizing the attention via attention rollout in Fig. 3. Attention rollout involves recursively multiplying attention weights across all layers [1], where we selected the minimum attention weight over all heads at each layer. We observed that the trained image encoder correctly focuses on important areas of the image plot, such as the graph, axes and legend, compared to the pre-trained ViT which misses portions of the graph and focuses on irrelevant blank spaces.

**Families of equations.** The t-SNE visualization of MathBERT's embeddings in Fig. 4a shows a well-structured embedding space, where different families of equations are separately clustered. Combinations of different families also result in embeddings being close to the original family cluster, where the 'periodic_polynomial' and 'exp_polynomial' clusters are close to the 'polynomial' cluster.

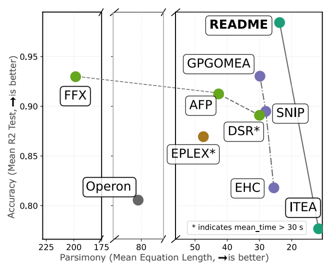
**Transferring pre-trained math structure.** The well-structured embedding space of MathBERT also transfers to the image encoder after training. While initial pre-trained ViT's embeddings do not have meaningful structures (Fig. 4b), a clearer separation between equation families emerges after training



(a) CP3 dataset.



(b) Strogatz dataset.



(c) PIED dataset.

Figure 5: Pareto plots for all algorithms.

(Fig. 4c). For example, the image embeddings of 'inv' equations become distinctly clustered, and 'periodic' and 'periodic_exp' equations are grouped together. Interestingly, 'periodic_polynomial' embeddings seem to be interpolated between 'periodic' and 'polynomial' embeddings, indicating the image encoder has recognized relationships between equation families.

## 4.3 Performance evaluation

We consider two experimental settings. First, the rapid setting, where algorithms have to be trained within a 30-second cut-off. Second, the ultra rapid setting, where algorithms have to be trained within a 10-second cut-off. Experimental details are in App. C.3.

### 4.3.1 Results for rapid setting

To evaluate both both accuracy and parsimony of the symbolic equations produced by each method, we plot Pareto plots for each dataset where the y-axis represents accuracy measured by mean $R^2_{Test}$ (larger is better) and the x-axis represents parsimony measured by mean equation length (smaller is better). The x-axis is plotted in descending order so that along both axes, points furthest from the origin are the best. We indicate Pareto frontiers in different colors. Points within the same front are non-dominated with respect to each other, meaning no method in the frontier outperforms another on both accuracy and parsimony. A higher Pareto frontier contains at least one model that Pareto-dominates a model on a lower frontier.

As shown in Fig. 5, README and ITEA are consistently in the top Pareto frontier. However, ITEA is in the top frontier mainly as it heavily biases towards shorter expressions – its accuracy values tend to be among the lowest, especially for the Physics Informed dataset. In contrast, README identifies parsimonious and accurate equations rapidly ($\leq$ 30s) and consistently lies on the first Pareto frontier. It achieves the highest mean $R^2_{\text{Test}}$ among all algorithms, demonstrating strong overall accuracy.

A particularly informative comparison is that between README and SNIP, as both share similarities in being SR foundation model methods that uses GWO during inference, but with SNIP relying on numerical encoders trained from scratch with $\sim 60\times$ more data while README uses the novel image plots data processing, image/text encoders and training strategy described in Sec. 3. Despite using less training data (see Table 1), README was able to consistently outperform SNIP across all datasets in both accuracy and parsimony (i.e., README Pareto-dominates SNIP), demonstrating its advantages of the README framework.

App. C.5.2 shows the detailed results on accuracy and parsimony for all methods across the three datasets. Among the methods, GPGOMEA, DSR, and EPLEX Regressor had actually exceeded the 30-second limit, but we still reported their results for analysis – GPGOMEA for the Strogatz dataset, and DSR and EPLEX across all three, sometimes taking up to $4\times$ the time budget to produce a result. However, despite taking longer time, these methods still underperform README which ran within the 30-second time budget. Running configurations for all algorithms are provided in App. C.4.1.

### 4.3.2 Evaluation results for ultra rapid setting

Next, we analyze the ultra-rapid setting that requires methods to complete inference within 10 seconds. This setting is motivated by applications requiring low-latency predictions of physical movement, such as physics checking in synthetic video generation or real-time decision support, where inference must be done within a matter of seconds. We evaluate methods on the Physics Informed dataset (PIED), which is from real-world experiments and hence serves as a test of method robustness in practical, noisy environment.

To achieve ultra-rapid inference, we use our **GWOBO** optimization process when running README as described in Sec. 3.3, which speeds up inference while still achieving good results. Among the benchmarks, only README with GWOBO and FFXRegressor are able to consistently return symbolic expressions under the ultra-rapid setting time limit of 10 seconds. In this setting, README continues to outperform FFXRegressor in both parsimony and accuracy. README achieved an $R^2_{\text{Test}}$ of 0.958 with average equation length of 18 terms, while FFXRegressor only achieved $R^2_{\text{Test}}$ of 0.930 with significantly worse average equation length of 219 terms.

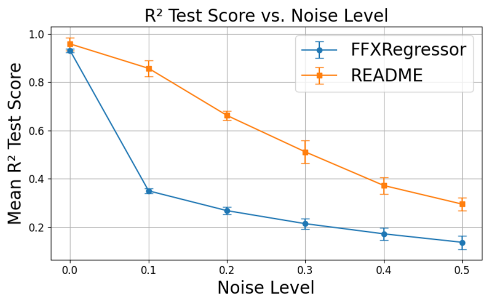


Figure 6: Mean $R^2_{\text{Test}}$ vs. target noise for FFXRegressor (orange) and README (blue) in the ultra-rapid setting ($< 10$s).

In addition, we tested the methods' sensitivity to noise by adding Gaussian noise to data observations $\tilde{y}$ and evaluating their performance. Fig. 6 shows how the accuracy ($R^2_{\text{Test}}$) of both methods changes over increasing noise level (i.e., the Gaussian noise standard deviation is varied from 0.1 to 0.5 times the root mean square of the observation values). Note that README demonstrates greater robustness to noise with less performance degradation as noise is increased, e.g., README 's accuracy only decreased from 0.958 to 0.857 when noise of 0.1 noise level is added, but FFXRegressor's accuracy had a much larger drop from 0.930 to 0.350.

## 4.4 Demonstration of Equation Prediction for Noisy Real Experimental Data

To demonstrate the practicality of our framework for an ultra-rapid setting, we analyze two real-world videos where object motion must obey physical laws. The task is for the methods to use the first 75% of the time-sequenced video data to generate an interpretable symbolic equation that can predict the final 25% of the video. This tests the methods' **extrapolation capability on real-world noisy and limited data** and showcasing its potential for applications in motion prediction. Note that the data would include not just physical noise (e.g. air resistance), but noise during both the filming (e.g. camera motion) and video processing (e.g. object detection and trajectory extraction) stages, making this a very challenging problem. We adopt a 10s runtime constraint to reflect real-time applications such as physics validation in synthetic video generation and decision-making systems that require relatively low latency and physically accurate predictions.

We developed two pipelines to extract object coordinates over time and apply symbolic regression to model their trajectories. For the *Pendulum Swinging* video, we used Tracker software [6] with basic techniques such as template matching to estimate and track the pendulum bob across 311 frames. For the *Ping Pong Ball Bouncing* video, we used a YOLOv8n model [19] to detect the ball in each frame, extracted the center of its bounding box, and obtained 85 position points. Details are in App. C.7.

| **Algorithm** | **Pendulum Swinging** | | **Ping Pong Ball Bouncing** | |
|---|---|---|---|---|
| | $R^2$ | Equation Length | $R^2$ | Equation Length |
| **README** | $\mathbf{0.686 \pm 0.263}$ | $15.80 \pm 1.79$ | $\mathbf{0.862 \pm 0.204}$ | $24.90 \pm 6.92$ |
| FFX | $0.012 \pm 0.016$ | $81.10 \pm 59.31$ | $0.000^* \pm 0.000$ | $103.00 \pm 0.00$ |

Table 2: Performance of README and FFX on two real-world videos under a 10-second time constraint. *Negative $R^2$ values were clipped to zero as they indicate performance worse than predicting the mean.

For these experiments, only README, FFX and ITEA managed to complete within 10s for both videos, while GPGOMEA and Operon only managed one (see Table 2). Note that README was the only method that could perform the task well, while the other benchmarks could barely fit the final 25% of the video with very low $R^2$ scores, which underscores how challenging the task is due to the very limited, noisy data and short inference time. Allowing for more time budget will have README achieve $R^2 > 0.9$ while continuing to outperform benchmarks.

## 5 Conclusion

We introduced README, a framework for rapid equation discovery that uses (1) an informative, compressed image representation of numerical data; (2) an efficiently-trained transformer-based model built on top of pre-trained image and text encoders ($\sim 60\times$ less training data compared to past works); and (3) a novel combination of BO and GWO for a rapid, effective optimization process to achieve state-of-the-art and robust SR results for challenging settings with realistic, noisy settings and tight time constraints (<10s).

This work represents a first step toward quick and reliable symbolic regression that can be used as a module within real-world tasks. Potential applications include computer vision and robotics, where real-time, interpretable physics validation and decision-making are essential. This approach also holds promise for domains such as video analytics and synthetic video generation, where low-latency fast symbolic regression is crucial.

## Acknowledgements

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-PhD/2023-01-039J) and is part of the programme DesCartes which is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.

## References

[1] Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197, 2020.

[2] Mahmoud Al Najar, Rafael Almar, Erwin Bergsma, Jean-Marc Delvit, and Dennis Wilson. Improving a shoreline forecasting model with symbolic regression. In *ICLR 2023 Workshop on Tackling Climate Change with Machine Learning*, 2023.

[3] Guilherme Seidyo Imai Aldeia and Fabrício Olivetti de França. Interaction-transformation evolutionary algorithm with coefficients optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 2274–2281, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Nikola Anđelić, Ivan Lorencin, Vedran Mrzljak, and Zlatan Car. On the application of symbolic regression in the energy sector: Estimation of combined cycle power plant electrical power output using genetic programming algorithm. *Engineering Applications of Artificial Intelligence*, 133:108213, 2024.

[5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 936–945. PMLR, 18–24 Jul 2021.

[6] Douglas Brown. Tracker video analysis and modeling tool. `https://opensourcephysics.github.io/tracker-website/`, 2024. Accessed: 2025-05-23.

[7] Steven L. Brunton, J. Nathan Kutz, Krithika Manohar, Aleksandr Y. Aravkin, Kristi Morgansen, Jennifer Klemisch, Nicholas Goebel, James Buttrick, Jeffrey Poskin, Adriana W. Blom-Schieber, Thomas Hogan, and Darren McDonald. Data-driven aerospace engineering: Reframing the industry with machine learning. *AIAA Journal*, page 1–26, July 2021.

[8] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[9] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon c++: An efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, page 1562–1570, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[12] Jake Fitzsimmons and Pablo Moscato. Symbolic regression modeling of drug responses. In *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, pages 52–59, 2018.

[13] R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, USA, 1987.

[14] Roman Garnett. *Bayesian Optimization*. Cambridge Univ. Press, 2022.

[15] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[16] Arthur Grundner, Tom Beucler, Pierre Gentine, and Veronika Eyring. Data-driven equation discovery of a cloud cover parameterization, February 2024.

[17] Roger Guimerà, Ignasi Reichardt, Antoni Aguilar-Mogas, Francesco A. Massucci, Manuel Miranda, Jordi Pallarès, and Marta Sales-Pardo. A bayesian machine scientist to aid in the solution of challenging scientific problems. *Science Advances*, 6(5):eaav6971, 2020.

[18] Apivich Hemachandra, Gregory Kang Ruey Lau, See-Kiong Ng, and Bryan Kian Hsiang Low. PIED: Physics-informed experimental design for inverse problems. In *The Thirteenth International Conference on Learning Representations*, 2025.

[19] Glenn Jocher, Jiayu Qiu, and Ayush Chaurasia. Ultralytics yolov8. `https://github.com/ultralytics/ultralytics`, 2023.

[20] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[21] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers, April 2022.

[22] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *International Joint Conference on Artificial Intelligence*, 1989.

[23] John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, Jun 1994.

[24] J.R. Koza. Genetically breeding populations of computer programs to solve problems in artificial intelligence. In *[1990] Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 819–827, 1990.

[25] William La Cava, Kourosh Danai, and Lee Spector. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence*, 55:292–306, 2016.

[26] William La Cava, Thomas Helmuth, Lee Spector, and Jason H. Moore. A probabilistic and multi-objective analysis of lexicase selection and $\epsilon$-lexicase selection. *Evol. Comput.*, 27(3):377–402, September 2019.

[27] William La Cava, Lee Spector, Kourosh Danai, and Matthew Lackner. Evolving differential equations with developmental linear genetic programming and epigenetic hill climbing. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, page 141–142, New York, NY, USA, 2014. Association for Computing Machinery.

[28] Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. Prose: Predicting operators and symbolic expressions using multimodal transformers, 2023.

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

[30] Trent McConaghy. *FFX: Fast, Scalable, Deterministic Symbolic Regression Technology*, pages 235–260. Springer New York, New York, NY, 2011.

[31] Kazem Meidani, Parshin Shojaee, Chandan K. Reddy, and Amir Barati Farimani. SNIP: Bridging Mathematical Symbolic and Numeric Realms with Unified Pre-training. In *The Twelfth International Conference on Learning Representations*, October 2023.

[32] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[33] Hirotaka Moriguchi and Hod Lipson. Learning symbolic forward models for robotic motion planning and control. volume ECAL 2011: The 11th European Conference on Artificial Life of *Artificial Life Conference Proceedings*, page 86, 08 2011.

[34] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. In *Advances in Neural Information Processing Systems*, pages 10203–10214, 2018.

[35] Brenden K. Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, October 2020.

[36] Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR, July 2021.

[39] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[40] Michael D. Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, page 543–544, New York, NY, USA, 2010. Association for Computing Machinery.

[41] Jia Tracy Shen, Michiharu Yamashita, Ethan Prihar, Neil Heffernan, Xintao Wu, Ben Graff, and Dongwon Lee. Mathbert: A pre-trained language model for general nlp tasks in mathematics education. *arXiv preprint arXiv:2106.07340*, 2021.

[42] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific Equation Discovery via Programming with Large Language Models, June 2024.

[43] Guido F. Smits and Mark Kotanchek. *Pareto-Front Exploitation in Symbolic Regression*, pages 283–299. Springer US, Boston, MA, 2005.

[44] Steven H Strogatz. *NONLINEAR DYNAMICS AND CHAOS, THIRD EDITION: With applications to physics, biology, chemistry,... And engineering, third edition, student's solution*. 2024.

[45] Mattias E. Thing and Sofie M. Koksbang. cp3-bench: A tool for benchmarking symbolic regression algorithms tested with cosmology, 2024.

[46] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. https://arxiv.org/abs/2106.14131v1, June 2021.

[47] Marco Virgolin and Peter A. N. Bosman. Coefficient mutation in the gene-pool optimal mixing evolutionary algorithm for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 2289–2297, New York, NY, USA, 2022. Association for Computing Machinery.

[48] Marco Virgolin and Solon P Pissis. Symbolic regression is NP-hard. *Transactions on Machine Learning Research*, 2022.

[49] Ylva Wahlquist, Jesper Sundell, and Kristian Soltesz. Learning pharmacometric covariate model structures with symbolic regression networks. *Journal of Pharmacokinetics and Pharmacodynamics*, 51(2):155–167, Apr 2024.

[50] Changxin Wang, Yan Zhang, Cheng Wen, Mingli Yang, Turab Lookman, Yanjing Su, and Tong-Yi Zhang. Symbolic regression in materials science via dimension-synchronous-computation. *Journal of Materials Science & Technology*, 122:77–83, 2022.

[51] Yiqun Wang, Nicholas Wagner, and James M. Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, Sep 2019.

[52] Zhixin Zhang and Zhiyong Chen. Modeling and control of robotic manipulators based on symbolic regression. *IEEE Transactions on Neural Networks and Learning Systems*, 34(5):2440–2450, 2023.

## A Related works

**Regression-based models** Regression-based approaches, as their name suggests, use regression on a fixed basis to find an accurate representation of the input and output data of a system. Regression-based approaches tend to focus on using regularization to find a parsimonious basis [8, 10]. However, they predefine the structure of the equation they aim to find, reducing the SR problem into one solving a system of linear equations [29]. This makes regression-based approaches very fast, but limits the generalizability of regression-based approaches. For example, McConaghy's Fast Function Extraction [30] uses regularization to prune the search space of functions, and is a fast and deterministic algorithm for solving symbolic regression algorithms.

**Genetic programming-based models** These include seminal works by Koza [22, 24, 23], which represent each approximation of an unknown equation as a genetic program with a tree-like data structure, with traits (or nodes in the tree) representing functions or operations and variables representing real numbers. The fitness of each genetic program is its prediction error. Fitter genetic programs undergo a set of transition rules comprising selection, crossover, and mutation to find the optimal equation form iteratively. Genetic programming algorithms perform well in SR tasks as the transition rules allow for large variations in the population to adequately explore the search space.

Recent SR algorithms that use genetic programming to tackle common issues such as coefficient optimization include for example, de Franca and Aldeia's Interaction-Transformation Evolutionary Algorithm (ITEA) [3] which uses a search space which contains only mathematical expressions described as an affine combination of nonlinear transformations of different interactions between the original variables. ITEA then uses a mutation-based evolutionary algorithm to search for the optimal coefficients to express a linear relationship between the nonlinear transformations and the target variable. Likewise, Virgolin et al.'s GP-GOMEA [47] searches for optimal values of coefficients by estimating interdependencies between model components and using this information to cross-over interdependent components en block, to preserve their concerted action improving mutation in genetic programming, and La Cava et al's EPLEXRegressor [26] uses lexicase selection as a parent selection method that considers training cases individually, rather than in aggregate, to select elite parents for mutation.

Genetic programs may greedily mimic nuances of the unknown equation [43], limiting generalisability. David Goldberg [15] therefore proposed to use Pareto optimization to balance the objectives of fit and parsimony in SR. At each iteration, the fittest genetic programmes lie on the non-dominated Pareto-frontier. Other works that use the Pareto frontier to evolve a population include Schmidt and Lipson's age-fitness Pareto (AFP) optimization regressor [40].

Lastly, some genetic algorithms explicitly minimize a target. For example, Burlacu et al.'s Operon [9], a genetic programming symbolic regression algorithm written in `C++`, minimizes speed, while La Cava et al.'s epigenetic hill climbing symbolic regression algorithm (EHC) [27] minimize complexity of the equation and computational cost.

However, the transition rules of genetic programming algorithms mean that they are by design highly sensitive to hyperparameters and do not scale well to high-dimensional data [36]. This motivates the study of other types of symbolic regression algorithms.

**Foundation model-based models.** Deep learning algorithms are a recent advancement in the field of symbolic regression. An early example of a deep learning approach to symbolic regression is Petersen's Deep Symbolic Regression [36] which uses a recurrent neural network to emit a distribution over tractable mathematical expressions and employ a novel risk-seeking policy gradient to train the network to generate better-fitting expressions. Deep learning approaches have evolved following the progress in the field. Likewise, a few works [17] have also looked into adopting a Bayesian approach to symbolic regression, where a prior can be established based on a pool of past expressions which incorporates some domain knowledge, as well as naturally encode some balance between model complexity represented by the prior and data fit. Radford et al. [37], in 2021, proposed multimodal architectures trained on symbolic expressions and numerical data to speed up genetic programming-based symbolic regression methods, while Kamienny et al. [21] introduced the use of transformers for symbolic regression to directly predict symbolic equations and Biggio et al. [5] popularized the use of pre-trained transformers for symbolic regression. As the proposed architectures have grown bigger, the amount of data required to train these models has also grown. Pre-trained transformers start with a robust understanding of general symbolic patterns and syntax, and can be fine tuned to specific tasks such as for SR with less task-specific data. Since the model has already learned generic features of mathematical equations, the optimization process during fine-tuning focuses on symbolic regression-specific nuances. This significantly reduces training time and computational costs, and the pre-trained transformers converge faster during evaluation as they were trained on richer datasets. Meidani et al.'s SNIP [31] proposed training numeric and symbolic encoders jointly to produce a structured latent space that could be used for cross-domain tasks such as symbolic regression. Our framework README builds on all these foundational works by (1) introducing an informative, compressed image representation of numerical data that can be efficiently used in our framework for symbolic regression, (2) using pre-trained image and text encoders along with customized components to significantly reduce the training data and resources needed, (3) a novel method GWOBO that enables symbolic regression at the ultra-rapid setting ($< 10s$) that have not been explored before in past works, and (4) achieving significantly performance improvements over past methods.

## B Additional training and architecture details

### B.1 Model Training Details

**Training Data** To train our model, we generated synthetic pairs of numeric and symbolic data using the publicly available codebase in [21], following the data generation settings used by SNIP [31]. This includes operator downsampling and restricting expressions to at most 10 input dimensions. The only difference is that we generated a total of ~1 million (image, equation) pairs for training, whereas SNIP used ~60 million pairs to pretrain their numeric and symbolic encoders.

**Numeric Data Visualization** For each equation, input data $x$ with dimensionality $n \leq 10$ was generated, comprising 200 data points represented as $200 \times n$ matrices. Each input dimension was paired with targets $y$, represented as a $200 \times 1$ vector. Each input dimension was plotted individually against the target $y$ using Matplotlib, assigning different colors for clarity. Each graph includes the dimensionality information in its title. Figure 7 shows a sample graph from our dataset, illustrating how patterns are captured across different dimensions.

**Equation Representation** Expression trees were converted into their equivalent infix notation, providing readable symbolic equations. Each visual plot and corresponding symbolic equation formed a training pair.

#### B.1.1 Image Encoder

We employed a pre-trained Vision Transformer model `google/vit-base-patch16-224-in21k` [11]. This model, trained extensively on diverse image data, excels in pattern recognition.

This approach offers two primary advantages:

1. **Pattern Recognition**: The model swiftly identifies patterns within numeric data visualizations, leveraging robust feature extraction capabilities from its pre-training.
2. **Regularization Effect**: By plotting each input dimension against the target in the same visualization, the model is naturally regularized, treating all dimensions uniformly. This method helps prevent overfitting to any specific dimension.

We also experimented with the larger `google/vit-huge-patch14-224-in21k` model and observed improved results, as expected due to the increased model capacity. However, for the purpose of balancing performance and computational efficiency in our experiments, we chose to use the smaller base Vision Transformer for performance evaluation.

#### B.1.2 Text Encoder

The symbolic equations were encoded using `tbs17/MathBERT` [41], a model specifically pre-trained on mathematical text, enabling effective interpretation and encoding of symbolic equations. Equations were provided directly in infix notation, aligning well with MathBERT's pre-training on mathematical textbooks and notations. Utilizing plain text inputs avoids constraints typical of tree-based representations, offering greater flexibility and leveraging the inductive biases inherent to MathBERT.

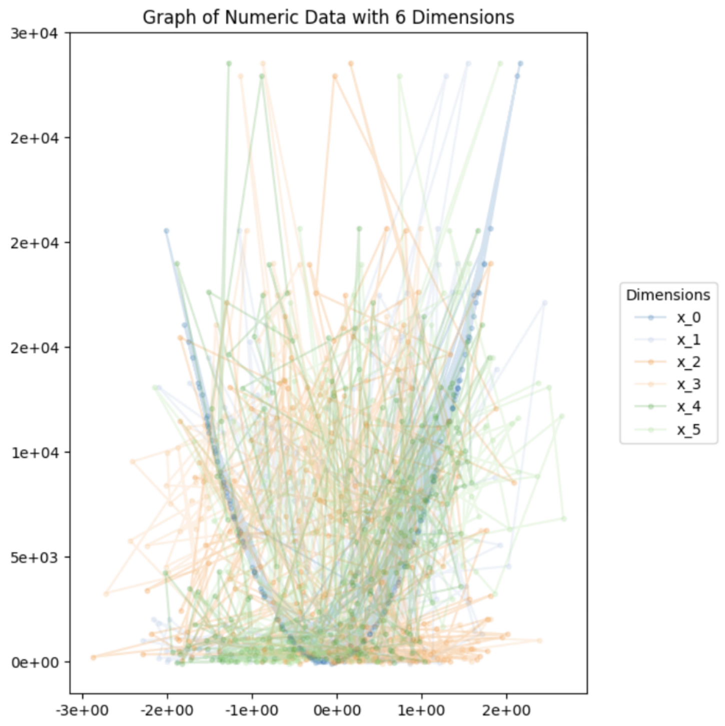


Figure 7: Sample visualization of numeric data with multiple dimensions plotted against the target. Patterns across dimensions are captured effectively.

### B.1.3 Aligned Embedding Space

The README model aligns numeric and symbolic representations in a shared latent space. Inspired by the joint training approach used in CLIP [37], README optimizes a symmetric cross-entropy loss over similarity scores. A contrastive loss based on the InfoNCE objective [34] effectively aligns embeddings of matching numeric-symbolic pairs while pushing apart non-matching pairs.

The loss function is defined as:

$$\mathcal{L} = -\sum_{(v,s)\in B} \Big[\log \mathrm{NCE}(Z_S, Z_V) + \log \mathrm{NCE}(Z_V, Z_S)\Big] \tag{1}$$

where $B$ represents a batch of (symbolic, numeric) data pairs, and $\mathrm{NCE}(Z_S, Z_V)$ and $\mathrm{NCE}(Z_V, Z_S)$ are the symbolic-to-numeric and numeric-to-symbolic contrastive losses, respectively. The symbolic-to-numeric contrastive loss is computed as:

$$\mathrm{NCE}(Z_S, Z_V) = \frac{\exp\left(Z_S \cdot Z_V^+/\tau\right)}{\sum_{Z\in\{Z_V^+, Z_V^-\}} \exp\left(Z_S \cdot Z/\tau\right)}$$

where $\tau$ is a temperature parameter, $Z_V^+$ represents positive numeric embeddings that correspond to the symbolic embedding $Z_S$, and $Z_V^-$ are negative embeddings from other batch data. This symmetric contrastive loss encourages alignment of numeric and symbolic pairs while separating unrelated pairs.

### B.1.4 Text Decoder

For decoding symbolic equations, we also adopted the decoder architecture detailed in [21], consisting of 16 transformer decoder layers. This architecture effectively leverages attention mechanisms to autoregressively generate equations from the aligned embedding representations, benefiting from its deep, layered structure which facilitates complex symbolic regression tasks.

Following prior work [31], training is conducted in two stages. First, the decoder is trained with the image and text encoders frozen, allowing it to learn how to decode from the latent space. Next, the encoders and decoder are fine-tuned together, so the representations become better suited for decoding symbolic equations. The decoder is supervised using cross-entropy loss over the target symbolic sequence, encouraging accurate reconstruction of symbolic expressions from the shared representation.

### B.2 Ablation Studies

For the ultra-rapid setting, where the models are expected output an expression within 10 seconds, we introduce a key contribution: a novel hybrid algorithm that combines the Grey Wolf Optimizer with Bayesian Optimization (GWOBO) to efficiently identify high-quality symbolic expressions under tight runtime constraints.

As shown in Section 4.3.2, our model is also more robust to noise compared to FFXRegressor across varying noise levels. Note that in the section we showed GWOBO results with a candidate set size of 70.

We show that increasing the number of wolf candidates leads to substantial gains in performance while maintaining the same setup described in Section 4.3.2. Specifically, we continue to select the top 3 candidates based on Upper Confidence Bound (UCB) scores, computed using a Gaussian Process with an RBF kernel as detailed in D. Only these top 3 are decoded using Algorithm 1, while the remaining candidates are evaluated using surrogate scores from the GP.

These combined real and surrogate scores are then used to update the population via GWO. As shown in Table 3, GWOBO consistently outperforms pure GWO across all candidate configurations.

| **Candidates** | **GWO** | | **GWOBO** | |
|---|---|---|---|---|
| | $R^2$ | Equation Length | $R^2$ | Equation Length |
| 10 | 0.865 ± 0.058 | 18.34 ± 1.35 | **0.880 ± 0.037** | **16.54 ± 2.36** |
| 30 | 0.903 ± 0.046 | 19.27 ± 0.76 | **0.934 ± 0.044** | **17.05 ± 1.43** |
| 50 | 0.924 ± 0.037 | 18.24 ± 1.43 | **0.937 ± 0.026** | **17.26 ± 1.33** |
| 70 | 0.946 ± 0.029 | 17.29 ± 1.50 | **0.958 ± 0.027** | **16.67 ± 1.57** |

Table 3: Comparison of GWO and GWOBO at different candidate counts under ultra rapid setting (10 seconds). Each entry reports mean ± standard deviation. Best values are bolded.

We also ablated the impact of the encoder modality on the SRBench *Strogatz* (synthetic) and *Blackbox* (real-world) datasets. We compared using an *image encoder* against a *numeric encoder*, both paired with the symbolic encoder. As shown below, the image encoder yields significantly better performance, highlighting that inductive biases captured from graphs provide a more effective representation for symbolic regression.

Table 4: Image Encoder + Symbolic Encoder

| **Dataset** | $R^2_{\text{Test}}$ | **Parsimony** |
|---|---|---|
| Strogatz | 0.9425 | 31.14 |
| Blackbox | 0.6092 | 44.19 |

Table 5: Numeric Encoder + Symbolic Encoder

| **Dataset** | $R^2_{\text{Test}}$ | **Parsimony** |
|---|---|---|
| Strogatz | 0.8575 | 30.36 |
| Blackbox | 0.4577 | 44.96 |

We further compared different encoder architectures, using encoders trained on a smaller set of 512,000 pairs for quick evaluation on the same datasets. As shown in Table 6, ViT-base with

MathBERT outperforms CLIP with T5. This is likely due to MathBERT's strong inductive bias for mathematical structure based on its pretraining [41]. While ViT is not specifically trained on math-related content [11], it performed better than CLIP in this setting. We also evaluated ViT-huge and observed marginal gains, but selected ViT-base to ensure efficiency during timed experiments.

| Algorithm | Mean $R^2$ Test Score | Mean Equation Length |
|---|---|---|
| **vit-huge-with-mathbert** | $\mathbf{0.767 \pm 0.016}$ | $\mathbf{14.69 \pm 1.60}$ |
| vit-base-with-mathbert | $0.765 \pm 0.012$ | $18.56 \pm 0.82$ |
| clip-with-t5 | $0.738 \pm 0.018$ | $15.13 \pm 0.75$ |

Table 6: Comparison of ViT-based and multimodal models on symbolic regression. Best $R^2$ and smallest Equation Length are bolded. Higher is better for $R^2$, lower is better for Equation Length.

## C Additional Experimental Details

### C.1 Algorithms for Inference Decoding and Optimization

**Algorithm 1** README inference decode algorithm

```
1: Input: Image encoder $\mathcal{I}$, Inference dataset $\mathcal{D}$
2: Output: Symbolic expression $\widetilde{G}$, Associated MSE loss on inference dataset $\bar{L}$
3: $I = \mathcal{P}(\mathcal{D})$ //Process dataset to image plot
4: $\tilde{s} = \mathcal{I}(I)$ //Pass image through image encoder
5: $G = \mathcal{W}(\tilde{s})$ //Decode to symbolic expression
6: Run BFGS($G, \mathcal{D}$) [13] to optimize for MSE loss $L$ evaluated over $\mathcal{D}$ to obtain $\tilde{G}$ and final loss $\tilde{L}$
7: Return $\tilde{G}$ and $\tilde{L}$
```

**Algorithm 2** README inference optimization process

```
1: Input: Image encoder $\mathcal{I}$, Inference dataset $\mathcal{D}$, Target loss $\bar{L}$, Max iterations $T$, GWO Pop size $M$
2: Output: Best-fit symbolic expression and loss $r^* = (G^*, L^*)$
3: Init GWO pop $P = \{s_0^k\}_{k=1}^M$ (see App. C.3) and GP
4: Decode $P$ via Alg.1 (lines 5–6) to get $(\tilde{G}_k, \tilde{L}_k) \forall s_k \in P$
5: for $t = 1, 2, \dots, T$ do
6:     Select top three wolves from P, $(s_t^\alpha, s_t^\beta, s_t^\gamma)$, based on UCB criterion
7:     Perform GP regression with $\{\{(s_i^a, R^2(s_i^a))\}_{a \in [\alpha,\beta,\gamma]}\}_{i=0}^{t-1}$
8:     Decode $s_t^a$ with Alg.1 (lines 5–6) to obtain $r_t^a = (\tilde{G}_t, \tilde{L}_t)$, $a \in [\alpha, \beta, \gamma]$
9:     Obtain corresponding $R^2$ score $R^2(s_t^a)$ for the expression $\tilde{G}_t$
10:    Store $r^* \in \{r_i\}_{i=1}^t$ with the lowest $\tilde{L}^*$.
11:    Exit if $L^* < \bar{L}$
12:    Run GWO to update $P$, with decoded loss (line 8) for top 3 wolves, and GP estimated loss for other wolves.
13: end for
14: Return $r^*$
```

### C.2 Dataset Details

To comprehensively evaluate our symbolic regression framework, we curated a diverse set of 61 problems drawn from publicly available data sources in multiple scientific domains. These datasets were selected to balance a range of characteristics, including equation complexity, noise levels, and real-world relevance. They span both simulated and experimentally grounded physical systems, allowing us to assess model performance in controlled and practical scenarios.

### C.2.1 Strogatz Dataset

This dataset comprises 14 canonical equations modeling nonlinear dynamical systems, originally drawn from the textbook *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering* by Steven H. Strogatz [44, 25]. These systems describe a range of physical and biological processes such as population dynamics, chemical oscillators, and mechanical systems, and are commonly used in the literature to benchmark symbolic regression algorithms due to their compact closed-form representations and interpretable dynamics.

### C.2.2 CP3-Bench

This dataset consists of 28 equations derived from a diverse collection of real-world and simulated problems across physics, engineering, and environmental science. These problems were introduced as part of the CP3-Bench benchmark [45], which was designed to evaluate the capability of scientific equation learning models in recovering compact symbolic expressions from noisy data. The dataset includes systems such as gas solubility estimation, diffusion, and biochemical rate equations, and reflects varying levels of complexity and noise, making it a rigorous testbed for symbolic regression.

### C.2.3 Physics Informed Dataset

To evaluate the effectiveness of our model on physics-informed problems, we curated a dataset comprising 19 equations. These equations span simulated, experimentally validated, and real-world scenarios, as detailed below:

- **Groundwater Flow (11 equations):**
  These equations model steady-state unconfined groundwater flow and are sourced from the paper *"Investigating Steady Unconfined Groundwater Flow using Physics Informed Neural Networks"* by Mohammad Afzal Shadab, Dingcheng Luo, Yiran Shen, Eric Hiatt, and Marc Andre Hesse.
  GitHub: `https://github.com/dc-luo/seepagePINN/tree/main`
- **Chromatography (4 equations):**
  These equations originate from the experimental validation study *"Can a Computer 'Learn' Nonlinear Chromatography?: Experimental Validation of Physics-Based Deep Neural Networks for the Simulation of Chromatographic Processes"* by Sai Gokul Subraveti, Zukui Li, Vinay Prasad, and Arvind Rajendran. The equations simulate nonlinear solute transport in chromatographic columns.
- **Fluid Dynamics (2 equations):**
  We include the *nikuradse_1* and *nikuradse_2* equations, which describe the friction factor for turbulent flow in rough pipes based on experimental studies by Johann Nikuradse. These expressions capture the nonlinear relationship between the Darcy friction factor, Reynolds number, and relative roughness in turbulent pipe flow.
- **Pendulum Motion (2 equations):**
  These equations are derived from video recordings of a swinging pendulum captured using the Tracker software. The motion was tracked using a fixed camera setup, and position-time data was extracted to recover the underlying physical relationship governing the pendulum's oscillatory dynamics.

## C.3 Experiment Settings

All experiments in section 4.3 were conducted on 2× AMD EPYC 7763 64-Core CPUs and 1× NVIDIA L40 GPUs (CUDA 12.1, Driver 545.23.06), and running Ubuntu 22.04.3 LTS. All software was implemented in Python 3.11.2 using PyTorch 2.0.0, Transformers 4.44.2, and BoTorch 0.11.3.

For each problem described in each dataset (see Appendix C.2.1), we applied a 75%–25% train-test split. To ensure consistent computational constraints across problems, if a problem contained more than 200 training points, the training set was randomly subsampled to retain only 200 points. In experiments involving noise, Gaussian noise with the specified standard deviation was added to the target values of the training set. All experiments were repeated across five random seeds: 23654, 15795, 860, 5390, and 16850. These seeds controlled both the train-test split and the noise sampling, ensuring that our model was evaluated on different subsets of the data in each problem and that its performance was robust to variation in data sampling.

For our experiments under the *rapid* and *ultra-rapid* settings, we initialized both GWO and GWOBO with 71 wolf candidates: 70 perturbations around the base latent encoding plus the original encoding.

## C.4 Evaluation Details

### C.4.1 Other benchmarks configurations

The benchmarks used were McConaghy's FFX Regressor [30], de Franca and Aldeia's ITEA [3], Virgolin et al.'s GP-GOMEA [47], La Cava et al.'s EPLEX [26], Schmidt and Lipson's AFP regressor [40], La Cava et al.'s EHC regressor [27], Burlacu et al.'s Operon [9], and Meidani et al.'s SNIP [31]. The benchmarks were run using their default settings in SRBench. Hyperparameter tuning was skipped. This section clarifies these configurations.

McConaghy's FFX Regressor [30] has no tunable parameters. de Franca and Aldeia's ITEA [3] has a default population size of 1000 and 5000 generations. The minimum and maximum exponents of the interactions are $-1$ and $+1$. The minimum and maximum number of terms in an expression is 2. The number of non-zero exponents in each term of the initial population is 1. The transformation functions supported are the identity function, the hyperbolic tangent function, the sine function, the cosine function, the logarithmic function, the exponential function, and the square root function. Virgolin et al.'s GP-GOMEA [47] sets a budget of 500000 evaluations over a single run (as opposed to interleaved multiple runs). It uses the default functions for addition, subtraction, multiplication and division. The initialized maximum tree height is 4. La Cava et al.'s EPLEX [26] uses a selection mechanism called `epsilon_lexicase`. It uses 500 generations, with 500 individuals in the genetic programming population. The genetic programming algorithm also forces survival of the best individual in the population. The maximum number of nodes at the initialization of the genetic program is 20, and increases to 64 for the rest of the program. Schmidt and Lipson's AFP regressor [40] uses the parametric hill climber algorithm. It uses 250 generations, with 1000 individuals in the genetic programming population. The genetic programming algorithm also forces survival of the best individual in the population. The maximum number of nodes at the initialization of the genetic program is 20, and increases to 64 for the rest of the program. La Cava et al.'s EHC regressor [27] uses an epigenetic hill climbing algorithm. It uses 100 generations, with 1000 individuals in the genetic programming population. The genetic programming algorithm also forces the survival of the best individual in the population. The maximum number of nodes at the initialization of the genetic program is 20, and increases to 64 for the rest of the program. Burlacu et al.'s Operon [9] uses five local iterations with 10000 generations. It sets a maximum evaluation budget of $5 \times 10^5$. The population size is 500. The default allowed symbols are addition, subtraction, multiplication, and division. These benchmarks were run from the SRBench github, maintained by Cava Lab, at `https://github.com/cavalab/srbench`. The number of trials and number of jobs are set to 1. The code is run locally as opposed to on LPC. The time limit is set depending on the time budget of the experiment. The `skip_tuning` hyperparameter is used to skip tuning. Meidani et al.'s SNIP [31] uses the grey wolf optimizer with a population of 50. It uses a beam search size of 2 with a stopping criterion of $R^2 = 0.9998$. The maximum iteration budget is changed from 80 to a time budget depending on the experiment.

### C.4.2 README configurations

In the rapid setting, each algorithm was allocated a runtime of **30 seconds** to improve the $R^2$ training fit. For README, we used the Grey Wolf Optimizer (GWO) to explore the neighborhood around a base latent encoding. A total of 70 additional candidates were generated by adding scaled Gaussian noise to the base encoding, resulting in 71 latent vectors (1 original + 70 perturbed). In each iteration, all 71 candidates were decoded into symbolic expressions and evaluated on the test set (see Algorithm 1). Their scores were then used to perform a population update using the GWO algorithm. This full decoding strategy was feasible due to the relatively generous runtime budget.

In the ultra rapid setting, each algorithm was given a strict time limit of **10 seconds** to maximize the $R^2$ training fit. We enabled README to operate effectively within this constraint by introducing a novel GWOBO algorithm, detailed in Appendix D. GWOBO combines Grey Wolf Optimization with Bayesian optimization by scoring candidate embeddings using a Gaussian Process with an Upper

Confidence Bound (UCB) acquisition function. Only the top 3 candidates are selected for decoding, significantly reducing computational cost compared to decoding all candidates with pure GWO. As shown in Appendix B.2, GWOBO achieves higher accuracy than the standard GWO within the tight time budget.

### C.5 Detailed Evaluation of Accuracy, Parsimony, and Runtime

#### C.5.1 Extended Results for Table 1

Table 1 compares README and SNIP on the Physics-Informed dataset. Here, we provide similar comparisons on the Strogatz and CP3-Bench datasets. Despite using 60 times less pretraining data, README consistently outperforms SNIP across all three datasets, highlighting the effectiveness of our README framework.

| Model | Pretraining Data | Mean $R^2_{\text{Test}}$ |
|---|---|---|
| README | $\sim$1 million pairs | $0.880 \pm 0.018$ |
| SNIP | $\sim$60 million pairs | $0.791 \pm 0.069$ |

Table 7: Comparison on the Strogatz dataset.

| Model | Pretraining Data | Mean $R^2_{\text{Test}}$ |
|---|---|---|
| README | $\sim$1 million pairs | $0.933 \pm 0.018$ |
| SNIP | $\sim$60 million pairs | $0.923 \pm 0.017$ |

Table 8: Comparison on the CP3-Bench dataset.

#### C.5.2 Detailed Results for the Rapid Setting (Section 4.3.1)

We provide a detailed explanation of the evaluation metrics used—Mean $R^2_{\text{Test}}$ for accuracy, Mean Equation Length for parsimony, and Mean Time for runtime—for all algorithms referenced in Figure 5.

To compute the $R^2_{\text{Test}}$ score, we follow the standard SRBench setup: each dataset is first split into 75% training and 25% testing. From the training split, we randomly subsample 200 points for training, and the fitted equation is evaluated on the held-out 25% to compute $R^2_{\text{Test}}$. The $R^2$ score measures the proportion of variance in the dependent variable that is predictable from the independent variables, with a score of 1.0 indicating perfect fit and lower values reflecting greater error.

Parsimony is measured by Equation Length, defined as the number of nodes in the expression tree. Each constant, variable, and operator (e.g., `"3.5"`, `"x_2"`, and `"mul"`) counts as a single node. Shorter equations are considered more parsimonious and are generally easier to interpret and more robust to noise.

Time refers to the duration each algorithm takes to produce a final equation, excluding all data preprocessing and splitting steps. It reflects the computational efficiency of the symbolic regression process itself.

The term "Mean" in all metrics indicates that values are averaged across all problems and five random seeds for each dataset. The random seed affects both the subsampling of 200 training points and the train-test split. The error bars shown represent the standard deviation across these five seeds.

As shown in Figure 5, **README** consistently lies on the first Pareto frontier across all three datasets, achieving the highest accuracy while remaining reasonably parsimonious. ITEA is also Pareto-optimal as it achieves lower accuracy but with smaller equation length, making it non-dominated in the accuracy-parsimony space. Tables 9, 10, and 11 report the detailed results for these metrics.

| Algorithm | Mean R$^2$ Test Score | Mean Equation Length | Mean Time (s) |
|---|---|---|---|
| **README** | $\mathbf{0.880 \pm 0.018}$ | $20.99 \pm 1.19$ | $24.95 \pm 1.77$ |
| OperonRegressor | $0.849 \pm 0.053$ | $60.46 \pm 2.25$ | $10.56 \pm 0.11$ |
| GPGOMEARegressor | $0.849 \pm 0.054$ | $29.27 \pm 3.46$ | $39.63 \pm 2.66$ |
| FFXRegressor | $0.822 \pm 0.063$ | $198.43 \pm 37.82$ | $6.65 \pm 9.41$ |
| SNIP | $0.791 \pm 0.069$ | $22.97 \pm 1.08$ | $26.42 \pm 0.44$ |
| AFPRegressor | $0.742 \pm 0.043$ | $38.00 \pm 6.08$ | $20.02 \pm 0.77$ |
| ITEARegressor | $0.736 \pm 0.052$ | $\mathbf{12.83 \pm 0.27}$ | $15.67 \pm 0.34$ |
| DSRRegressor | $0.693 \pm 0.067$ | $19.16 \pm 3.49$ | $138.12 \pm 3.58$ |
| EHCRegressor | $0.644 \pm 0.042$ | $19.13 \pm 1.57$ | $12.44 \pm 0.30$ |
| EPLEXRegressor | $0.591 \pm 0.076$ | $45.09 \pm 3.69$ | $60.74 \pm 1.18$ |

Table 9: Performance comparison on the Strogatz Dataset. Bolded entries indicate the best R$^2$ score and the most parsimonious model.

| Algorithm | Mean R$^2$ Test Score | Mean Equation Length | Mean Time (s) |
|---|---|---|---|
| **README** | $\mathbf{0.933 \pm 0.018}$ | $23.34 \pm 0.11$ | $23.04 \pm 1.07$ |
| FFXRegressor | $0.930 \pm 0.006$ | $172.45 \pm 15.37$ | $2.78 \pm 0.16$ |
| ITEARegressor | $0.918 \pm 0.002$ | $\mathbf{16.08 \pm 0.56}$ | $16.14 \pm 0.29$ |
| GPGOMEARegressor | $0.910 \pm 0.004$ | $29.51 \pm 0.83$ | $17.35 \pm 5.20$ |
| AFPRegressor | $0.887 \pm 0.003$ | $39.63 \pm 3.76$ | $25.13 \pm 0.63$ |
| EPLEXRegressor | $0.885 \pm 0.021$ | $51.96 \pm 1.45$ | $68.70 \pm 3.07$ |
| SNIP | $0.883 \pm 0.091$ | $27.25 \pm 0.92$ | $29.19 \pm 0.26$ |
| OperonRegressor | $0.882 \pm 0.013$ | $77.19 \pm 1.12$ | $11.15 \pm 0.06$ |
| EHCRegressor | $0.862 \pm 0.006$ | $21.31 \pm 1.14$ | $13.73 \pm 0.30$ |
| DSRRegressor | $0.803 \pm 0.005$ | $17.07 \pm 1.61$ | $136.84 \pm 0.59$ |

Table 10: Performance comparison on the CP3-Bench Dataset. Bolded entries indicate the best R$^2$ score and the most parsimonious model.

| Algorithm | Mean R$^2$ Test Score | Mean Equation Length | Mean Time (s) |
|---|---|---|---|
| **README** | $\mathbf{0.984 \pm 0.004}$ | $23.76 \pm 2.35$ | $28.91 \pm 0.12$ |
| GPGOMEARegressor | $0.930 \pm 0.003$ | $29.85 \pm 0.38$ | $21.26 \pm 3.54$ |
| SNIP | $0.923 \pm 0.017$ | $24.52 \pm 0.45$ | $28.08 \pm 0.24$ |
| FFXRegressor | $0.930 \pm 0.007$ | $198.41 \pm 15.47$ | $3.09 \pm 0.07$ |
| AFPRegressor | $0.912 \pm 0.007$ | $42.67 \pm 2.84$ | $26.06 \pm 1.48$ |
| DSRRegressor | $0.891 \pm 0.041$ | $30.00 \pm 0.00$ | $135.32 \pm 1.61$ |
| EPLEXRegressor | $0.869 \pm 0.026$ | $47.40 \pm 3.53$ | $74.29 \pm 0.32$ |
| EHCRegressor | $0.818 \pm 0.033$ | $25.53 \pm 1.75$ | $13.94 \pm 0.23$ |
| OperonRegressor | $0.806 \pm 0.082$ | $80.98 \pm 1.36$ | $14.47 \pm 0.15$ |
| ITEARegressor | $0.777 \pm 0.017$ | $\mathbf{11.67 \pm 0.40}$ | $16.77 \pm 0.46$ |

Table 11: Performance comparison on the Physics Informed Dataset. Bolded entries indicate the best R$^2$ score and the most parsimonious model.

### C.6 Latent Space Analysis

For Sec. 4.2, the equations are classified into the following family types.

- periodic: contains $\sin$ and/or $\cos$, and is periodic. E.g. $\sin(x_1) + \cos(x_2), \cos(x_1)\cos(x_2)$.
- exp: contains $\exp$ only. E.g. $\exp(x_1) + \exp(x_2)$.
- polynomial: contains $+$, $\times$, and/or the power function. E.g. $x_1^2 + x_2$.
- inv: contains the inverse function. E.g. $x_1^{-1}$.

- periodic_exp: contains $\sin, \cos$ and/or $\exp$. E.g. $\sin(x_1) + \exp(x_2)$.
- periodic_polynomial: contains $\sin, \cos, +, \times$, and/or power. E.g. $\sin(x_1^2) \times x_2$.
- exp_polynomial: contains $\exp, +, \times$, and/or power. E.g. $\exp(x_1 + x_2) + x_2^2$.

Examples provided are for a 2-dimensional input, where $x = [x_1, x_2]^\top$.

In Fig. 4, the t-SNE plots were generated with 512 equations and the t-SNE perplexity parameter was set as 30. We noted that 'periodic_polynomial' embeddings seem to be interpolated between 'periodic' and 'polynomial' embeddings for trained image embeddings, while this characteristic was not obvious for text embeddings from MathBERT. Thus, we hypothesize that some relationships between equation families may be more learnable when visualized in plots, as they may appear different symbolically but can be more easily captured by the image encoder through visual patterns.

### C.7 Video Analytics Pipeline

To evaluate our framework in ultra-rapid, real-world settings, we test it on two videos where object motion follows physical laws. Each method receives the first 75% of frames to derive an interpretable symbolic equation and is then tasked to extrapolate and predict the remaining 25

We implemented two pipelines to extract the position of objects from video frames: one based on template matching and the other using YOLOv8 for object detection. While these methods are applicable to any video source, we demonstrate their effectiveness using two specific examples. Template matching using the Tracker Software is showcased on a video of a pendulum swinging, and bounding box identification with YOLOv8 is demonstrated on a video of a ping pong ball dropping. If the object of interest is not supported by YOLOv8's predefined classes, template matching offers a flexible alternative for tracking custom objects.

#### C.7.1 Tracker Software

To extract motion data from real-world footage, we utilized the Tracker software [6], which is based on *template matching*. In our experiment, we used a publicly available video of a simple pendulum in motion to demonstrate the effectiveness of this pipeline.[3] Template matching works by selecting a region of interest and then scanning each subsequent frame to find the region that most closely resembles the original template. By identifying the best match in each frame, the software tracks the object's position over time.

Using this method, we tracked the pendulum bob across frames and extracted a total of 311 data points representing its *x* and *y* coordinates over time. The resulting $(x, y, \textit{time})$ coordinates were exported to a CSV file and used as input to our README model for symbolic regression.

#### C.7.2 YOLOv8

For automated tracking in a separate experiment, we used the pre-trained YOLOv8n model [19] to detect and track the trajectory of a falling ping pong ball from a publicly available YouTube video.[4] The video was trimmed to a 10-second segment from 348 to 358 seconds (5 minutes and 48 seconds to 5 minutes and 58 seconds). YOLOv8 identified the object of interest, classified as a `sports ball`, and recorded the center coordinates of its bounding box in each frame. In total, 85 data points were extracted.

We used the `yolov8n.pt` model with the detection class set to `sports ball` over the specified time window. The resulting $(x, y, \textit{time})$ coordinates were exported to a CSV file and used as input to our README model for symbolic regression.

## D GWOBO: Grey Wolf Optimizer with Bayesian Optimization

For our novel GWOBO algorithm, we begin by generating an initial population of latent encodings by perturbing a base encoding with scaled Gaussian noise, similar to the initialization used in the

[3]Video source: `https://www.youtube.com/watch?v=02w9lSii_Hs`

[4]Video source: `https://www.youtube.com/watch?v=pZlYl0l2lFs`

standard Grey Wolf Optimizer (GWO). Each candidate in this set is decoded using Algorithm 1 to obtain its symbolic expression and corresponding score.

To select the next embeddings to perform decoding on, we attempt to construct a Gaussian process (GP) surrogate to predict the score for an embedding. For each latent embedding in the original 512-dimensional space that we have already decoded, we compute its difference from the base encoding, then perform a Johnson-Lindenstrauss transformation [20] to project the difference down to a smaller 20-dimensional space. Given the transformed lower-dimensional vectors and their corresponding actual scores, we fit a GP with radial basis function (RBF) kernel with automatic relevance determination (ARD) and appropriate input and output normalization. The GP surrogate can then be used to compute the upper confidence bound (UCB) score for each latent embedding, which can be seen as an optimistic estimate of the actual score.

To reduce the high cost of decoding, in subsequent iterations only the **top 3 candidates** as ranked by UCB scores are decoded using Algorithm 1. Their actual scores are then used to update the GP model, while the remaining candidates are assigned surrogate scores predicted by the GP. These updated scores are then used by the GWO algorithm to perform a population update, guiding the search toward more promising regions. This allows reduction in the running time since the UCB scores can be computed much more rapidly compared to the true score while being reasonably accurate.

This loop continues until either the runtime exceeds the 10-second time budget or a sufficiently good score ($R^2$ train > 0.9998) is achieved. This hybrid approach, combining population-based candidate generation and optimization with GP-based surrogate modeling, enables strong performance while ensuring the entire inference process completes within the ultra-rapid 10-second runtime constraint.

We also conducted an ablation study on the number of initial candidates used to fit the GP, which is detailed in Appendix B.2.

## E Limitations and Broader Impacts

We introduced README, a framework for symbolic regression that leverages image representations of numerical and pre-trained multimodel foundation models for efficient learning. Compared to other foundation-model-based approaches, README requires significantly less training data and time, and can benefit from capability advancement of open-sourced text and image encoders. While a foundation model approach may allow for faster inference time and better performance, and is also capable of making full use of modern hardware such as GPUs, there are settings where this approach is less suitable, e.g., in Internet of Things (IoT) deployment settings where the hardware is constrained to lightweight devices/CPUs. As most works on symbolic regression do not consider very high-dimensional datasets, we have similarly only considered up to 10-dimensional problems. We leave it to future works to examine the performance of README in higher-dimensional problems.

As README allows users to rapidly identify equations to describe data, it has the potential to support applications such as interactive/iterative scenarios such as adaptive scientific experimental and close to real-time decision making. README might also be used as components in AI/machine learning systems where interpretability in terms of symbolic equations would be useful. While we expect that the majority of such applications will lead to societal benefits, there may be malicious actors who might come up with applications that are to the detriment of society – general regulations and efforts to prevent such abuse of AI/machine learning tools are needed.

## F Directly using multi-modal large language models for symbolic regression

To demonstrate that our approach is effective and necessary for symbolic regression, we used GPT-4o out of the box for a naive comparison. On the physics-informed dataset with real-world measurements, our README model achieved a Mean $R^2$ Test score of 0.958 under the ultra rapid 10 second setting with 71 candidates as detailed in B.2, while GPT-4o achieved only 0.015. This highlights that GPT-4o, used out of the box, is unable to perform symbolic regression meaningfully.

For GPT-4o, we followed the same 5-seed train-test splits described in Appendix C.3. We provided 200 subsampled data points and explicitly informed GPT-4o of the number of input dimensions, instructing it to return a valid NumPy expression that fits the data. Notably, GPT-4o was given an advantage by being explicitly told to produce equations with the correct number of input dimensions, a constraint that was not enforced for README. Table 13 shows the prompt and example outputs from GPT-4o on the Physics Informed Dataset (seed 23654), illustrating that although the expressions are syntactically valid, they fail to fit the data well.

| Algorithm | Mean $R^2$ Test Score | Mean Equation Length |
|---|---|---|
| README (71 initial candidates) | $\mathbf{0.958 \pm 0.027}$ | $16.67 \pm 1.57$ |
| GPT-4o | $0.015 \pm 0.023$ | $\mathbf{10.46 \pm 0.67}$ |

Table 12: Comparison between README and GPT-4o on Physics Informed dataset for ultra-rapid setting (10 seconds). Bold indicates the better score per metric.

| **Prompt** | **GPT-4o Discovered Equations for seed 23654** |
|---|---|
| You are given training data with input dimension = `{num_dim}`. The X array contains input points, and the Y array contains the corresponding target values. X = `{X_to_fit}` Y = `{Y_to_fit}` Provide a single NumPy compatible expression `f(x)` that takes an (n, `{num_dim}`) array `x` and returns an (n,) array of predictions. Reply with **only** the expression itself (e.g., `np.sin(x[:,0]) + 0.5*x[:,1]`), without any explanation or quotes. | `0.05 * np.sin(2 * np.pi * x[:, 0]) + 0.1` |
| | `0.5 * np.sin(x[:, 0]) + 0.1` |
| | `0.5 * np.cos(x[:, 0]) + 0.5` |
| | `0.5 * np.cos(x[:, 0]) + 0.5` |
| | `0.5 * np.tanh(x[:, 0]) + 0.3 * np.tanh(x[:, 1]) + 0.5` |
| | `0.5 * np.cos(x[:, 0]) + 0.1` |
| | `0.05 + 0.03 * np.tanh(2 * x[:, 0])` |
| | `np.where(x[:, 1] > 0, 0.001, np.exp(x[:, 0]) / (1 + np.exp(x[:, 0])))` |
| | `1.75 - 0.1 * np.tanh(x[:, 0])` |
| | `-0.1 * np.tanh(x[:, 0]) - 0.05` |
| | `0.5 * np.sin(x[:, 0]) - 0.5 * x[:, 0] - 0.5` |
| | `np.where(x[:, 1] > 0, 0.00005, np.exp(-x[:, 0]**2))` |
| | `0.05 + 0.02 * np.sin(2 * np.pi * x[:, 0])` |
| | `0.05 * np.sin(2 * np.pi * x[:, 0]) + 0.1 * np.cos(np.pi * x[:, 0]) + 0.1` |
| | `np.where(x[:, 1] > 0, 0.001, np.exp(-x[:, 0]**2))` |
| | `np.where(x[:, 1] > 0, 0.5 * (x[:, 0] + 1.5)**2, np.exp(-x[:, 0]**2))` |
| | `0.05 * np.sin(2 * np.pi * x[:, 0]) + 0.1 * np.cos(np.pi * x[:, 0]) + 0.1` |
| | `-100 * x[:, 0] + 700` |
| | `0.5 * np.cos(x[:, 0]) + 0.5` |
| | `np.exp(-x[:,0]**2)` |

Table 13: Prompt and discovered expressions for 19 symbolic regression problems in the Physics Informed dataset.