Transformer Semantic Genetic Programming for Symbolic Regression

Philipp Anthes
anthes@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

Dominik Sobania
dsobania@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

Franz Rothlauf
rothlauf@uni-mainz.de
Johannes Gutenberg University
Mainz, Germany

Abstract

In standard genetic programming (stdGP), solutions are varied by modifying their syntax, with uncertain effects on their semantics. Geometric-semantic genetic programming (GSGP), a popular variant of GP, effectively searches the semantic solution space using variation operations based on linear combinations, although it results in significantly larger solutions. This paper presents Transformer Semantic Genetic Programming (TSGP), a novel and flexible semantic approach that uses a generative transformer model as search operator. The transformer is trained on synthetic test problems and learns semantic similarities between solutions. Once the model is trained, it can be used to create offspring solutions with high semantic similarity also for unseen and unknown problems. Experiments on several symbolic regression problems show that TSGP generates solutions with comparable or even significantly better prediction quality than stdGP, SLIM_GSGP, DSR, and DAE-GP. Like SLIM_GSGP, TSGP is able to create new solutions that are semantically similar without creating solutions of large size. An analysis of the search dynamic reveals that the solutions generated by TSGP are semantically more similar than the solutions generated by the benchmark approaches allowing a better exploration of the semantic solution space.

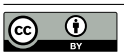
CCS Concepts

• **Software and its engineering** → **Genetic programming**; • **Computing methodologies** → *Supervised learning by regression*; • **Theory of computation** → *Program semantics.*

Keywords

Genetic Programming, Transformer Models, Semantic Operators, Symbolic Regression

ACM Reference Format:
Philipp Anthes, Dominik Sobania, and Franz Rothlauf. 2025. Transformer Semantic Genetic Programming for Symbolic Regression. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726412



GECCO '25, July 14–18, 2025, Malaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726412

1 Introduction

Standard Genetic Programming (stdGP) generates solutions by applying syntactic variation operators to existing solutions in an evolutionary process. The variation operators modify the structures of previously identified solutions to create new solutions that potentially solve the target problem more effectively. However, for most problems, it is not the structure of the solutions that determines their effectiveness in solving the target problem, but rather the behavior of the solution induced by its structure. This includes symbolic regression (SR), which searches for mathematical functions that have a particular solution behavior that best captures the underlying data. This solution behavior defines which outputs a solution generates for given inputs [23, 24]. Unfortunately, variation operators of stdGP completely ignore how these modifications affect the behavioral properties.

To overcome this limitation, GP approaches have been developed that aim to generate new offspring that have similar solution behavior (semantics) to their predecessors [15, 24, 29, 32]. These semantic-based GP methods include geometric-semantic-genetic programming (GSGP), which modifies solutions directly on their semantics through geometric-semantic operators (GSOs). Although GSOs effectively guide the search in the semantic space of solutions, they are limited by the fact that GSOs are basically linear combinations of solution structures. As a result, GSGP always creates new offspring larger than their predecessors and tends to produce rapidly (in the worst case exponentially) growing solutions of high complexity during search [24, 32]. To mitigate this growth, variants of GSGP introduce simplification techniques, such as SLIM_GSGP [31], which continuously deflates solutions. Nevertheless, GSGP remains strongly tied to the preceding solution structures through the linear combination-based GSOs and covers only a few predefined modifications to generate similar semantics. GSGP (like most other GP variants) does not explicitly address the fact that semantics can be represented by a wide range of structurally different representations (genotypes). Consequently, developing search operators that are able to guide the evolutionary search in the semantic space of solutions and simultaneously generate these semantic similarities through flexible structural modifications is a viable research goal.

In recent years, generative transformers have proven their ability in various domains to capture contextual relationships between sequences and generate new ones that match the desired output, regardless of their structural composition [2, 3, 36]. This paper proposes TSGP (Transformer Semantic Genetic Programming), a novel approach that considers semantic similarity within the variation process by using a generative transformer as variation operator. To apply TSGP, we train a transformer model once to identify semantic similarities between mathematical functions. After training

the model, it can be used as a variation operator to sample new solutions of semantic similarity in a GP-like search. We apply TSGP to symbolic regression problems, which is a representative and relevant problem for GP approaches. We hypothesize that (1) a transformer is well suited for learning semantic similarities between mathematical functions and that (2), when integrated as a variation operator in GP, it can effectively generate functions that are semantically similar, leading to a more efficient search than traditional approaches.

We compare the performance of TSGP with stdGP, SLIM_GSGP, and two model-based search approaches, DSR and DAE-GP, across five black-box SR data sets. Our results demonstrate that TSGP evolves functions with comparable or even better prediction quality on unseen data compared to the baseline methods within a fixed budget of iterations. A subsequent analysis of TSGP, stdGP, and SLIM_GSGP shows that TSGP generates significantly smaller solutions than SLIM_GSGP. Furthermore, the solutions generated by TSGP exhibit a higher semantic similarity than the solutions generated by stdGP. To the best of our knowledge, TSGP is the first model-based GP approach to successfully integrate the semantic similarities of solutions in the variation process. Thus, TSGP addresses the calls for variation operators that consider the semantics of solutions during variation [14, 15, 24, 33].

Sect. 2 provides an overview of semantic approaches that incorporate semantic similarity into the variation process, as well as model-based approaches that use neural networks as search operators or to generate solutions for SR in a single forward pass. Sect. 3 introduces TSGP, which generates solutions with a semantic-aware transformer. Sect. 4 outlines the experimental design, benchmark problems, and methods. Furthermore, it presents the results of our comparative study, including additional analyses of the similarity of the generated solutions. We discuss the findings in Sect. 5 and end with concluding remarks.

2 Related Work

Semantic GP. In recent years, several methods have been introduced that focus on the semantics of solutions. Some semantic-based GP approaches focus on the variation process by controlling the semantic similarity of new solutions. First approaches indirectly influence the variation process by discarding offspring that significantly differ in semantics from their predecessors [15, 29, 33].

More advanced approaches, such as Geometric-Semantic Genetic Programming (GSGP), incorporate semantics directly into the variation process [24]. GSGP utilizes geometric-semantic operators (GSOs), such as semantic crossover, which generates a weighted combination of parent solutions, and semantic mutation, which extends the solution by randomly generated solution structures that are guaranteed to modify the semantics of the solution in a weighted way. GSOs ensure that new solutions maintain semantic similarity. The results show that GSOs outperform standard variation operators [24]. With the efficient implementation of Vanneschi et al., GSGP has proven to be applicable in a variety of real-world applications, including pharmacokinetics, financial data analysis, and several other domains [4, 22, 32, 34]. However, GSGP causes the offspring solutions to grow exponentially over generations. To counteract this growth, various studies have suggested simplifying the resulting offspring [21, 24, 31]. This includes approaches such as GSGP-Red, which simplifies solutions by aggregating repeating structures and coefficients in the solution after applying GSOs [21]. A recently published approach, SLIM_GSGP, introduces a new deflating mechanism that reduces the size of the offspring in the variation process by subtracting a randomly selected component of the solution that was previously added to the solution [31]. In addition to the deflate operator, SLIM_GSGP uses the standard operators of GSGP, which are referred to as inflate operators (semantic crossover and semantic mutation). The size of the offspring is regulated by balancing the probability of applying the inflate and deflate operators [31].

Neural Network-Guided Search. Neural networks have been proposed as search operators by actively optimizing the trainable model parameters during the search process. Examples include Deep Symbolic Regression (DSR) [26] or Denoising Autoencoder GP (DAE-GP) [40].

DSR [26] uses recurrent neural networks (RNN) to learn sampling policies through reinforcement learning (RL). The RNN autoregressively generates solutions, which are evaluated for their fit to the underlying data. Based on a reward function, the network parameters are updated to maximize the reward, updating the sampling policy to produce solutions of higher prediction quality. In subsequent versions of DSR, several components are added to improve the quality of the solutions, such as a large-scale pre-training phase and a neural-guided GP search at the decoding stage [18].

In contrast, DAE-GP utilizes a denoising autoencoder LSTM to model the distribution of solutions, which is integrated as a variation operator in GP [40]. After each selection step, the LSTM is trained to reconstruct the promising solutions by first encoding them into a latent representation and then decoding them back into their original form. Once the LSTM has learned the population's distribution, the LSTM acts as a variation operator, by applying small perturbations to solutions and passing them through the network, generating new solutions with similar properties. For SR problems, DAE-GP has demonstrated its ability to produce solutions of similar prediction quality compared to standard operators, while significantly reducing the complexity and size of the resulting structures [38].

Pre-trained Neural Networks. Besides search-based methods, transformer-based approaches have been developed that generate solutions for SR in a single forward pass. The transformers are trained using large synthetically generated data sets consisting of numerical input values, mathematical functions, and their resulting outputs. During training, the input-output pairs are passed to the transformer, which aims to reconstruct the corresponding mathematical function token-by-token. When applied to SR problems, these approaches generate solutions by passing the input-output pairs of the analyzed data set directly to the transformer. Initial approaches in this area generate function skeletons with placeholders for constants, which are subsequently optimized using non-linear optimization techniques [1, 30]. Subsequent approaches integrate the generation of constants directly into the decoding process, refining the sampled constants afterwards [10, 35]. Some approaches further improve equation generation by incorporating search strategies such as Monte Carlo Tree Search (MCTS) [11, 28]. For example, Transformer-Based Planning for Symbolic Regression (TSPR) [28]

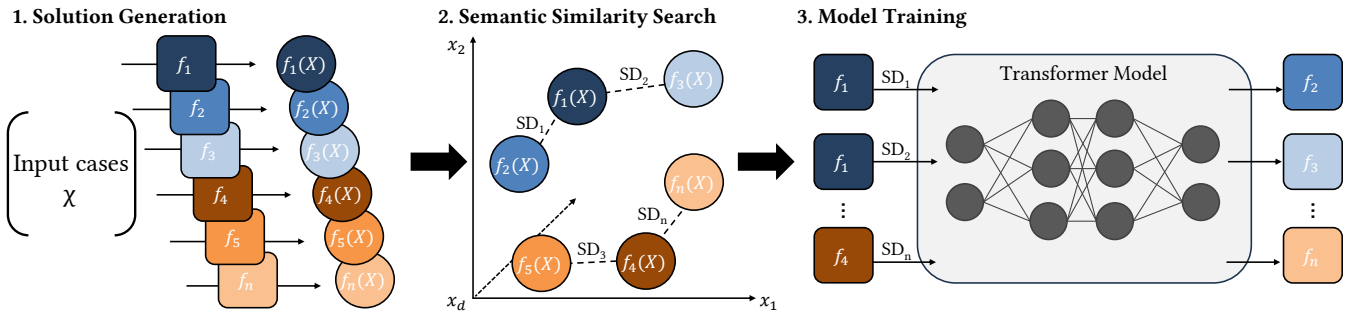


Figure 1: Model Building of TSGP. First, solutions are generated and their semantics are determined. Subsequently, solutions with similar semantics are identified in a similarity search and are passed as input-output pairs with their semantic distance (SD) to train a transformer to generate solutions of similar semantics.

combines a pre-trained end-to-end SR model [10] and MCTS predicting a solution, thus actively guiding the search for functions in a more effective manner.

3 The TSGP Approach

Transformer Semantic Genetic Programming (TSGP) is a novel GP approach that replaces the standard variation operators of GP with a semantic-aware transformer. The trained transformer receives a parental solution as input and returns an offspring solution with similar behavioral properties. Before the transformer is incorporated into GP, it is trained in a modeling phase with a large number of synthetic pairs of similar solutions. Model building is required only once; afterwards, the trained model can be applied to unknown and unseen test problems.

3.1 Model Building

Model building consists of three steps, as shown in Figure 1. The first phase involves generating a wide range of synthetic solutions and evaluating their semantics. In the second step, solutions with similar semantics are identified through a similarity search based on the semantic properties of the solutions. In phase three, solutions with similar semantics are passed to the transformer as input-output pairs for training. Each of these steps is explained in detail below.

Solution Generation. As we focus in our study on SR problems, solutions are mathematical functions f, which map a set of inputs $\mathcal{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ to a corresponding set of outputs $f(\mathcal{X}) = \{f(\mathbf{x_1}), \ldots, f(\mathbf{x_m})\}$, where each $\mathbf{x_i} \in \mathbb{R}^d$ is a vector of d dimensions (number of features of the problem). To obtain a large set of mathematical functions that well cover the search space, we apply stdGP to random synthetic SR problems.[1] For each synthetic SR problem, stdGP returns a set of functions $F = \{f_1, f_2, \ldots, f_n\}$ where n is the total number of unique functions generated. StdGP uses double tournament selection that balances the accuracy and simplicity of generated functions, penalizing overly complex functions during the search process [20]. The input-output data of the synthetic SR problems are derived from randomly generated linear regression models. The data sets are standardized, so that each feature has a mean of 0 and a standard deviation of 1. To increase the diversity of the functions generated by stdGP, we apply Gaussian noise to the data sets.

Once the set of functions F is generated, we calculate the semantics of the n solutions. The semantics $s(f)$ of a function f can be approximated by its output values $f(\mathcal{X})$, provided that $\mathcal{X}$ extensively covers the space of possible inputs [24, 29]. To achieve this, we evaluate each function on randomly generated, standardized inputs, where each of the d input features has a mean of 0 and a standard deviation of 1, ensuring that the semantics comprehensively capture the function's behavior across standardized inputs.

Semantic Similarity Search. Two functions $f_i, f_j \in F$ are considered semantically similar if their semantics, $s(f_i)$ and $s(f_j)$, differ only slightly [29]. The similarity is quantified by a semantic distance, which we define as the Euclidean distance between their semantic representations $\mathrm{SD}(f_i, f_j) = \|s(f_i) - s(f_j)\|_2$ [24].

To identify semantically similar functions, we apply a k-nearest neighbor (k-NN) search to the set F. For each function $f_i \in F$, k-NN returns a set of k neighbors $N_k(f_i)$ with minimal semantic distance

$$N_k(f_i) = \underset{\{F' \subset F, |F'|=k\}}{\operatorname{argmin}} \sum_{f \in F'} \mathrm{SD}(s(f_i), s(f)). \tag{1}$$

Thus, the semantic similarity search identifies the k functions that are semantically the most similar to f_i. For each of the k neighbors, a pair of functions is created which is used as training data for the transformer model, where f_i serves as input and all $f \in N_k(f_i)$ as corresponding outputs. The similarity search is performed for all functions $f_i \in F$, resulting in $n \times k$ possible input-output training pairs. To improve efficiency, we use the FAISS library [7], which reduces computational costs by clustering semantics and measuring distances only within each cluster.

Model Training. The model is trained to learn what constitutes semantic similarities between functions. Thus, all input-output pairs $\{f_i, f_o\}$, which are returned by a similarity search, are used as training data for the transformer. Since mathematical functions in GP are represented as parse trees, and transformers are designed to operate on tokens, we convert each parse tree into a structured sequence of tokens to make it compatible with the input format of the transformer. This transformation is achieved by enumerating the corresponding expression tree in prefix order, traversing the tree from top to bottom and left to right [26, 40].

[1] Instead of using stdGP, any other method that returns a large number of solutions covering the relevant search space is also applicable.

For the transformer model, we adopt the standard encoder-decoder architecture proposed by Vaswani et al. [36]. The encoder processes the input solution sequence f_i and converts it to the latent space of the transformer. From this latent space, the decoder auto-regressively generates a new solution sequence f_o token by token, guided by both the latent representation of the encoder and the previously sampled token of the sequence. During training, the model predicts the next token of the sequence based on its trainable model parameters θ. The goal of training is to identify the model parameters θ^* that minimize the error in predicting the next token of the sequence. To enable the transformer to learn varying degrees of semantic similarity between solutions, we provide not only f_i as input, but also the semantic distance, $\text{SD}(s(f_i), s(f_o))$ to the target output f_o as an additional input to the encoder and decoder layer.

3.2 Model Sampling

TSGP solves symbolic regression problems by using a semantic-aware transformer as a variation operator, which samples new solutions during the search. Instead of applying standard variation operators of GP, TSGP uses the transformer model to create offspring solutions from parental solutions.

Since we included the semantic distance $\text{SD}(s(f_i), s(f_o))$ as an additional input for the transformer model during training, we can adjust the semantic similarity of the generated solution by providing the desired semantic distance SD^{d} as an additional input. This allows us to control the step size in the semantic space during the search.

The transformer samples an offspring solution token-wise, based on the sequential representation of the mathematical input functions. We stop the sampling process of a solution as soon as an EOS (end-of-sequence) token is sampled or the maximum number of allowed tokens is reached. To ensure that the resulting token sequence represents a valid tree structure, we use the syntax control suggested by Wittenberg et al. [37]. Syntax control modifies the probabilities of selecting tokens during the sampling process by assigning a probability of zero to any tokens that would result in an invalid tree structure.

4 Experiments and Results

This section starts with the experimental settings, including the chosen benchmark problems, the TSGP configuration, and the baseline approaches used for comparison. We then evaluate the effectiveness of TSGP on five black-box regression problems and analyze its search behavior by examining the size of the solutions and the variation behavior.

4.1 Experimental Settings

We compare the performance of TSGP with stdGP, SLIM_GSGP, DSR, and DAE-GP on black-box regression problems, where the ground truth model is not known, taken from Penn Machine Learning Benchmarks (PMLB) [27]. PMLB is a recognized library of benchmark problems widely used in SR research, including SRBench [16]. For our experiments, we select all available black box data sets with four features where a sufficient number of samples (> 100) is available. The selected data sets include four real-world regression problems ERA (1000 samples), ESL (488 samples), Galaxy (323 samples), and LEV (1000 samples), as well as one synthetic data set, Pollen (3848 samples). We standardize the data sets by scaling each feature and target variable to have a mean of 0 and a standard deviation of 1, resulting in all variables being on the same scale. Owen et al. showed that such a standardization has positive effects on prediction quality and solution size [6, 25]. We use the evolutionary computation framework DEAP [8] for TSGP, stdGP, and DAE-GP. The transformer of TSGP and the LSTM of DAE-GP are implemented using Keras [5]. For SLIM_GSGP and DSR, we rely on the available Python implementations [26, 31]. All general parameter settings are specified in Table 1.

Table 1: Configuration parameters and values for the evaluated methods.

Parameter	Value
Initialization	Ramped Half-and-Half
Primitive Set	$\{V, ERC, +, -, \times, \%\}$
ERC Range	[-0.5, 0.5], stepsize 0.1
Population Size	100
Generations	50
Selection	Tournament selection of size 5
Evaluation Metric	Root Mean Squared Error (RMSE)
Runs	30

For GP-based methods, initial populations are generated using Ramped Half-and-Half (RHH). For TSGP, stdGP, and DAE-GP, which are based on the DEAP framework, we choose an initialization depth between 2 and 5. For SLIM_GSGP, we rely on the default initialization values as specified by the framework. We set the maximum allowed tree depth during a run to 17 [13]. The terminal set consists of all feasible variables $V = \{v_1, v_2, v_3, v_4\}$ that correspond to the dimensionality $d = 4$ of the data set and ephemeral random constants (ERCs), which are generated within the range of [-0.5, 0.5] at a step size of 0.1. The function set consists of the binary operators addition, subtraction, multiplication, and protected division. Division by zero is avoided by replacing the resulting value with 1. The population size is set to 100 and the search is conducted for 50 generations. We use tournament selection with tournament size 5 as selection method. The data sets are partitioned into training and test sets in a 50/50 ratio. The quality of a solution is evaluated using the Root Mean Squared Error (RMSE)

$$\text{RMSE} = \frac{\|\mathbf{Y} - f(\mathbf{X})\|_2}{\sqrt{m}}, \tag{2}$$

which is the L2 norm between the target output vector of the data set $\mathbf{Y}$ and the output vector generated by the function $f(\mathbf{X})$, normalized by the total number of observations m in the data set.

To create the training data for the transformer of TSGP, we apply stdGP to 50 synthetic regression problems, each consisting of four features (following the selection of the test problems). We perform stdGP runs with a population size of 2,000 until a sufficient number of functions are generated. All other GP search parameters are the same as defined in Table 1. As described in Sect. 3.1, for each generated function, its k-nearest neighbors (k=3) are determined based on their semantic distance (SD). An input-output pair is formed for each function and one of its k-neighbors if their semantic

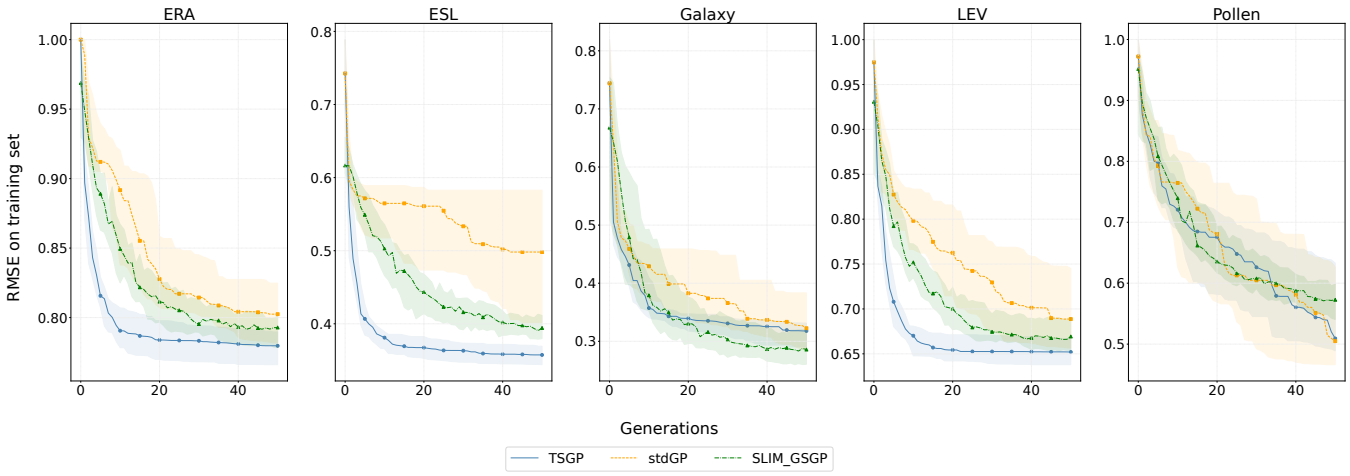


Figure 2: Median training RMSE of the best solution of TSGP, stdGP, SLIM_GSGP over generations for all analyzed black-box data sets.

similarity SD ≠ 0 and SD < 100. With this procedure, we generate a total of 5 mio semantically similar function pairs that are used to train the transformer.

The transformer architecture is adapted from Vasawni et al. [36], with 8 attention heads and a hidden dimension of 128. We use a smaller architecture with 2 encoders and decoder stacks; the input length is limited to 100 tokens. The transformer is trained using the AdamW optimizer [19] with a fixed learning rate of 10^{-3} for 8 epochs. Systematic hyper-parameter tuning was not feasible due to the complexity of the task. It is important to note that the transformer is trained only once. The resulting trained model is then used consistently in all experiments. For model sampling, we employ $SD^d = 0.1$.

StdGP uses subtree crossover and subtree mutation as variation operators. Subtree crossover is applied with a probability of 90%, with an internal node bias of 10% towards selecting terminal nodes [13]. Subtree mutation is applied with a probability of 10%, where new full subtrees are generated with a depth between 0 and 2.

For SLIM_GSGP, we use the default configuration of the framework. Thus, the SLIM+SIG2 variant is employed and no geometric crossover is performed. Variation relies solely on the semantic mutation operator. The default inflation mutation rate is set to 0.2, defining the probability of selecting the inflation mutation over the deflate operation during the mutation process of an solution.

DSR follows the default configuration of the framework. Since the results are evaluated using RMSE, the negative RMSE is chosen as a reward function. In addition, the batch size is set to match the population size of the GP-based methods (100), and the number of iterations is set to the number of generations performed (50).

DAE-GP is implemented following the specifications outlined by Wittenberg [39]. The denoising autoencoder LSTM features a single hidden layer that dynamically adjusts the size of the hidden dimension to match the maximum size of the solutions. Training is conducted until the training error converges, utilizing Adam optimization [12] with a learning rate of $\alpha = 0.001$. Levenshtein tree edit is used for corruption, applying a fixed edit percentage of 5%.

Table 2: Median test RMSE of the best solution identified within 50 generations for TSGP, stdGP, SLIM_GSGP (SLIM), DSR, and DAE-GP (DAE) for the five test sets. Bold values denote the best prediction quality (lowest RMSE). Significant differences of the best results are indicated by the label symbols.

Data set	$_a$TSGP	$_b$stdGP	$_c$SLIM	$_d$DSR	$_e$DAE
ERA	$_{bde}$**0.797**	0.817	0.810	0.852	0.904
ESL	$_{bcde}$**0.379**	0.502	0.418	0.507	0.595
Galaxy	0.327	0.337	$_{de}$**0.305**	0.434	0.468
LEV	$_{bde}$**0.672**	0.703	0.681	0.773	0.842
Pollen	0.518	$_{de}$**0.514**	0.569	0.750	0.752

4.2 Prediction Quality

We analyze the prediction quality of TSGP and the baseline methods for the selected SR problems by calculating the RMSE of the best solutions, where a lower RMSE indicates a better prediction quality. The best performing solution on the training set is used to compute the RMSE on the test set. The test results for all methods across all standardized datasets are summarized in Table 2. These values represent the median of 30 independent runs. Bold values indicate the highest prediction quality, which are tested for statistically significant differences using the Wilcoxon rank sum test with a significance level of $\alpha = 0.05$. The method labels (a,b,c,d,e) highlight statistically significant differences [9, 17].

We find that TSGP outperforms all baseline methods, including stdGP, SLIM_GSGP, DSR, and DAE-GP, on the majority of the evaluated data sets. For the data sets ERA, ESL, and LEV, the solutions generated by TSGP are of significantly higher prediction quality than those generated by stdGP, DSR, and DAE. Especially on the ESL data set, TSGP performs significantly better than all other evaluated baselines. On the Galaxy dataset, SLIM_GSGP achieves solutions with the highest prediction quality, while for Pollen, stdGP generates the best solutions. However, for both datasets, no significant

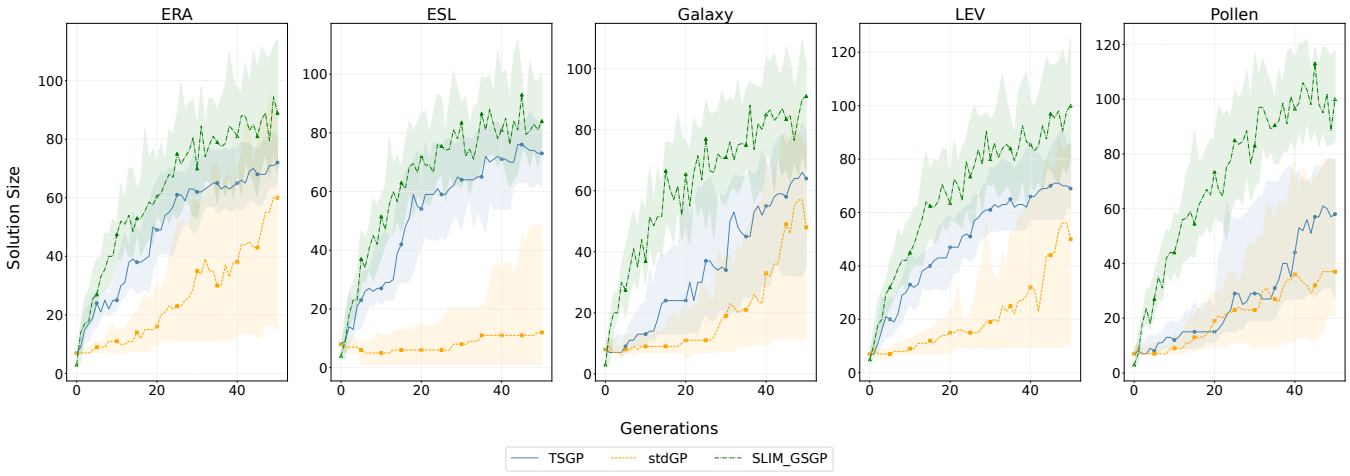


Figure 3: Median solution size of the best solution of TSGP, stdGP, SLIM_GSGP over generations for all analyzed black-box data sets.

differences to TSGP can be observed. The model-based approaches DSR and DAE-GP achieve significantly worse results on all data sets examined. Therefore, the remaining analysis focuses only on the comparison of TSGP with stdGP and SLIM_GSGP. We attribute the poor performance of DSR and DAE-GP to the fact that both adapt their neural networks within the search to find good solutions and, therefore, depend on a large amount of training data during the search. In contrast, TSGP is trained only once during a model-building phase on synthetic problems and can transfer its knowledge of useful variations to unseen datasets without requiring additional training.

Our results confirm the findings of Vanneschi [31] that semantic-based methods outperform standard operators in terms of prediction quality. Both TSGP and SLIM_GSGP consistently outperform stdGP across all datasets, except for Pollen, where TSGP achieves only comparable results. These results support the hypothesis that incorporating semantic operators into the variation process increases the effectiveness of the GP search and leads to improved prediction quality of the identified solutions.

To provide deeper insights into the search behavior of TSGP, we analyze the prediction quality of the best solutions over generations. Figure 2 shows the median RMSE of the training set (of 30 independent runs) over the number of generations. Additionally, the interquartile range (IQR) is included to quantify its variability. Note that the initial RMSE values of the first generation slightly differ between TSGP and stdGP compared to SLIM_GSGP due to differences in their frameworks and initialization.

The results indicate that TSGP finds high-quality solutions significantly faster than stdGP or SLIM_GSGP on the majority of data sets. This is demonstrated by ERA, ESL, and LEV, for which TSGP finds solutions with high prediction quality after only 10 generations, whereas the comparison methods converge noticeably slower. Especially on ESL, TSGP achieves a significantly lower RMSE after 10 generations (0.381) than stdGP (0.565) or SLIM_GSGP (0.503). On the Galaxy and Pollen data sets, TSGP converges at a rate similar to stdGP and SLIM_GSGP, showing no differences in convergence between the methods.

Table 3: Median solution size of the best solution identified within 50 generations by TSGP, stdGP, and SLIM_GSGP (SLIM). Values in bold indicate the smallest solutions. Significant differences with respect to all other methods are indicated by the label symbols.

Data set	$_a$TSGP	$_b$stdGP	$_c$SLIM
ERA	$_c$72	$_c$**60**	89
ESL	$_c$73	$_{ac}$**12**	84
Galaxy	$_c$64	$_c$**48**	91
LEV	$_c$69	$_c$**50**	100
Pollen	$_c$58	$_c$**37**	100

The IQR of TSGP across all data sets is noticeably smaller than stdGP and is similar in magnitude to that of SLIM_GSGP. This reduced spread indicates less variability in the prediction quality of the best solutions across the conducted runs. These results suggest that semantic-based methods, such as TSGP, are more stable in their exploration of the solution space. Unlike stdGP, their variation operators rely less on random perturbations of the solution behavior, leading to more consistent and reliable results over multiple runs.

4.3 Solution Size

Usually, not only prediction quality, but also size of the solutions is a decisive criterion. Simpler and more compact solutions are generally preferred, as large solutions tend to be more complex and can make human interpretation difficult [20, 31, 38]. Consequently, we quantify the solution size by counting the number of nodes of the best solutions. Table 3 lists the solution sizes of TSGP, stdGP, and SLIM_GSGP. As before, these values represent the median of 30 independent runs. Statistically significant differences are examined between all the evaluated methods and are marked with the

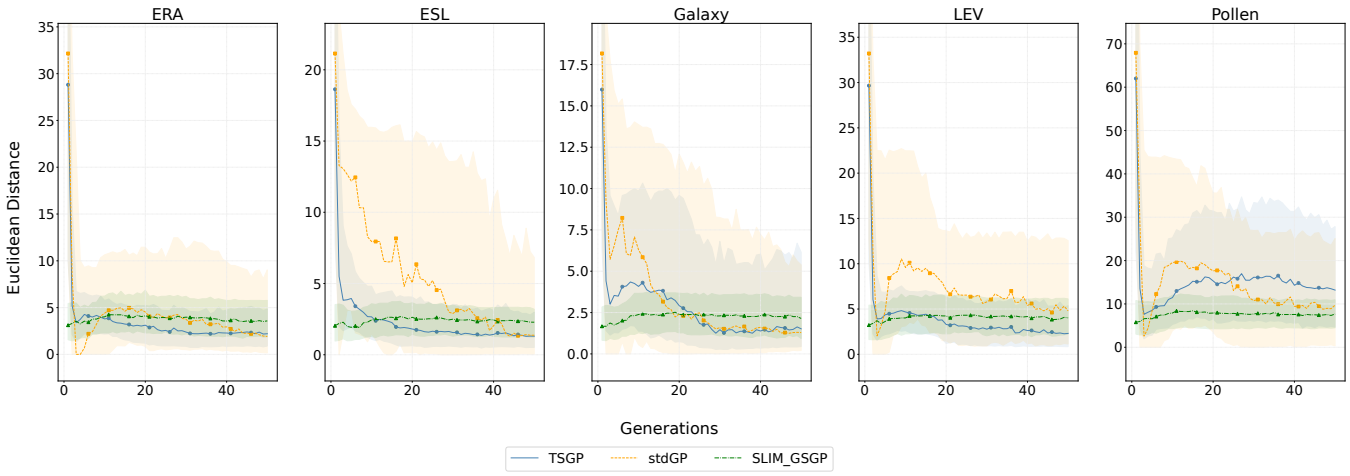


Figure 4: Median Euclidean Distance between the semantics $s(f_i)$ and $s(f_o)$ for TSGP, stdGP, SLIM_GSGP over generations for all analyzed black-box data sets.

respective labels (*a*,*b*,*c*). The smallest solution sizes are highlighted in bold.

We find that the best solutions found by stdGP have the smallest size, followed by TSGP and SLIM_GSGP. The small sizes of stdGP's solutions are expected, since their prediction quality is worse than those of the semantic methods, except on the Pollen data set. A comparison of the solution sizes between stdGP and the semantic methods shows statistically significant differences between stdGP and SLIM_GSGP, with all solutions of SLIM_GSGP being significantly larger than those of stdGP. However, no statistical differences in solution size were observed between TSGP and stdGP.

Interestingly, TSGP consistently finds statistically significantly smaller solutions than SLIM_GSGP across all data sets. This is particularly noteworthy because TSGP also generates solutions with higher prediction quality than SLIM_GSGP for most data sets. Compared to the geometric semantic operators of SLIM_GSGP, the transformer is able to generate semantic similarities through flexible structural modifications. Being trained on semantically similar pairs of solutions of different structural compositions, the semantic similarities can be generated by the transformer through various structural modifications. Therefore, TSGP does not depend on generating increasingly larger solutions within the search, but can generate semantically similar solutions of flexible structure and length. This is reflected in significantly smaller solutions generated across all data sets compared to SLIM_GSGP.

This is also evident when studying the sizes of the best solutions over generations, as shown in Figure 3. During the search, the best solutions found by TSGP are always smaller than the best solutions from SLIM_GSGP, although TSGP's solutions show higher prediction quality for the majority of the data sets. For the data sets where TSGP achieves higher prediction quality than the comparison methods (ERA, ESL, LEV), TSGP quickly identifies large solution structures with high prediction quality. However, for the data sets Galaxy and Pollen, where the solution quality of TSGP is comparable to stdGP and SLIM_GSGP, the increase in solution size for TSGP is more moderate and comparable to the increase for stdGP and is significantly smaller than Slim_GSGP with a decrease in solution size up to 42% on Pollen.

4.4 Variation behavior

To better understand the variation behavior of TSGP, we analyze the solutions sampled during the search by comparing each solution f_i with its offspring f_o over generations. We focus on the similarity of the solution behavior by measuring the distance between their semantics, $s(f_i)$ and $s(f_o)$, using the Euclidean distance. The semantics are approximated from the output of the functions when applied to the test set of the target problem. We only consider successful variations where the offspring is structurally different from its parent, which is almost always the case. Figure 4 plots the median distances of the variations over generations. The results are for 30 independent runs, with the IQR provided to analyze the variability of the semantic distances. Note that the magnitude of the Euclidean distance depends on the magnitude of the test set. Semantic similarities can therefore only be compared within a single data set, not across data sets.

For almost all data sets (except Pollen), TSGP's semantic-aware transformer produces offspring with the highest semantic similarity compared to stdGP and SLIM_GSGP. This is indicated by the fact that the median semantic distance for TSGP is lower than for stdGP or SLIM_GSGP for most generations. These results highlight the ability of transformers to recognize semantic variations of solutions during training and their ability to generate new solutions that exhibit similar behavioral characteristics.

Particularly noticeable is that the IQRs of the semantic methods are significantly smaller than those of stdGP with its syntactically based variation mechanisms. The semantic methods thus exhibit a more stable variation with regard to solution behavior. In particular, the 75th percentile of stdGP is significantly higher than that of the semantic methods. This indicates that the standard operators of GP often generate offspring that differ greatly from the solution behavior of the given input solution. This is to be expected, since

stdGP with mutation and crossover does not incorporate the behavior of solutions into the variation process, and thus often generates solutions that differ greatly in solution behavior. Semantic variation operators such as TSGP, on the other hand, consistently create solutions with more similar solution behavior in the majority of the population and thus transfer the favorable properties of the solutions more steadily to their offspring.

There are differences between the methods in the progression of behavioral similarity over generations. Especially at the beginning of the search, both TSGP and stdGP generate variations of high semantic distance. This is due to the random initialization of the population, which consists of solutions with very diverse solution behavior. These solutions initially have a low fit to the underlying problem, which means that TSGP, which is trained on solutions for synthetic SR problems, is unlikely to be able to generate solutions of low semantic distance to these random solutions at the beginning of the search. For data sets where TSGP can identify solutions with high prediction quality in a few generations, the semantic distance is significantly lower compared to the other methods. In these cases, TSGP effectively explores the search space and generates new solutions that retain useful properties from previous solutions. On the data sets Galaxy and Pollen, the semantic distance of TSGP is comparable to that of stdGP. Furthermore, the 75th percentile of these data sets is not as low as compared to stdGP as it is on the other data sets. This could explain why the prediction quality for these data sets is only comparable between TSGP and the other methods and does not outperform the prediction quality as on ERA, ESL, and LEV. Solutions are sought for these data sets, on which the transformer can generate only semantically similar structures to a limited extent. As a result, the search is not as effective as on the other data sets.

5 Conclusions and Future Work

We introduced TSGP, a novel semantic-based GP approach that uses a semantic-aware transformer as a variation operator. For TSGP, a transformer is trained once on synthetic data. The trained transformer can then be used as a search operator, which is able to generate favorable and semantic variations for unseen data sets.

For the majority of the analyzed black-box regression data sets, TSGP significantly outperforms stdGP, SLIM_GSGP, DSR, and DAE-GP in prediction quality and converges after only a few generations to high-quality solutions, significantly faster than the baseline methods. Furthermore, the solutions generated by TSGP are significantly smaller than those of SLIM_GSGP, with reductions up to over 40%, while maintaining solution quality that is comparable or even significantly better. We assume that this is due to the fact that the semantic-aware transformer in TSGP is more flexible in generating new semantic variations than SLIM_GSGP, where variations are based on linear combinations of solution structures. An analysis of the variation behavior shows that the TSGP is able to generate offspring with solution behavior that is significantly more similar to the behavior of standard GP operators.

As next steps, we will focus on extending TSGP to data sets of varying dimensionality to enable it to be applied to a wider range of data sets. Furthermore, we study the effects of training data on TSGP's variation process, as we believe that a greater variety of training data could have a positive impact on the generalizability of TSGP's high prediction quality across data sets. We also plan to systematically control the step size of the transformer during the search as larger step sizes SD^d would allow us to better escape local optima.

Acknowledgments

We would like to thank David Wittenberg for the valuable insights into DAE-GP and for providing his framework. We also like to thank Alina Geiger, Martin Briesch, and the entire team in Mainz for the inspiring discussions and thoughtful contributions.

References

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

[2] Martin Briesch, Dominik Sobania, and Franz Rothlauf. 2023. Large language models suffer from their own output: An analysis of the self-consuming training loop. *arXiv preprint arXiv:2311.16822* (2023).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Mauro Castelli, Leonardo Vanneschi, and Sara Silva. 2013. Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. *Expert Systems with Applications* 40, 17 (2013), 6856–6862.

[5] François Chollet et al. 2015. Keras. https://keras.io.

[6] Grant Dick, Caitlin A Owen, and Peter A Whigham. 2020. Feature standardisation and coefficient optimisation for effective symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 306–314.

[7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[8] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[9] Alina Geiger, Dominik Sobania, and Franz Rothlauf. 2023. Down-sampled epsilon-lexicase selection for real-world symbolic regression problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1109–1117.

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

[11] Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. 2023. Deep generative symbolic regression with Monte-Carlo-tree-search. In *International Conference on Machine Learning*. PMLR, 15655–15668.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] John R Koza. 1993. *On the programming of computers by means of natural selection*. MIT press.

[14] Krzysztof Krawiec and Pawel Lichocki. 2009. Approximating geometric crossover in semantic space. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. 987–994.

[15] Krzysztof Krawiec and Tomasz Pawlak. 2013. Approximating geometric crossover by semantic backpropagation. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 941–948.

[16] William La Cava, Bogdan Burlacu, Marco Virgolin, Michael Kommenda, Patryk Orzechowski, Fabrício Olivetti de França, Ying Jin, and Jason H Moore. 2021. Contemporary symbolic regression methods and their relative performance. *Advances in neural information processing systems* 2021, DB1 (2021), 1.

[17] William La Cava, Lee Spector, and Kourosh Danai. 2016. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 741–748.

[18] Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. 2022. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems* 35 (2022), 33985–33998.

[19] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[20] Sean Luke and Liviu Panait. 2002. Fighting bloat with nonparametric parsimony pressure. In *International conference on parallel problem solving from nature*. Springer, 411–421.

[21] Joao Francisco BS Martins, Luiz Otavio VB Oliveira, Luis F Miranda, Felipe Casadei, and Gisele L Pappa. 2018. Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In *Proceedings of the genetic and evolutionary computation conference*. 1151–1158.

[22] James McDermott, Alexandros Agapitos, Anthony Brabazon, and Michael O'Neill. 2014. Geometric semantic genetic programming for financial data. In *Applications of Evolutionary Computation: 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*. Springer, 215–226.

[23] Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. 2008. Semantic Building Blocks in Genetic Programming. In *Genetic Programming*, Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcázar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 134–145.

[24] Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. 2012. Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*. Springer, 21–31.

[25] Caitlin A Owen, Grant Dick, and Peter A Whigham. 2018. Feature standardisation in symbolic regression. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 565–576.

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

[27] Joseph D Romano, Trang T Le, William La Cava, John T Gregg, Daniel J Goldberg, Praneel Chakraborty, Natasha L Ray, Daniel Himmelstein, Weixuan Fu, and Jason H Moore. 2021. PMLB v1.0: an open source dataset collection for benchmarking machine learning methods. *arXiv preprint arXiv:2012.00058v2* (2021).

[28] Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. 2023. Transformer-based planning for symbolic regression. *Advances in Neural Information Processing Systems* 36 (2023), 45907–45919.

[29] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. 2011. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines* 12 (2011), 91–119.

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

[31] Leonardo Vanneschi. 2024. SLIM_GSGP: The non-bloating geometric semantic genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 125–141.

[32] Leonardo Vanneschi, Mauro Castelli, Luca Manzoni, and Sara Silva. 2013. A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In *European Conference on Genetic Programming*. Springer, 205–216.

[33] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. 2014. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines* 15 (2014), 195–214.

[34] Leonardo Vanneschi, Sara Silva, Mauro Castelli, and Luca Manzoni. 2014. Geometric semantic genetic programming for real life applications. *Genetic programming theory and practice xi* (2014), 191–209.

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

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[37] David Wittenberg. 2022. Using denoising autoencoder genetic programming to control exploration and exploitation in search. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 102–117.

[38] David Wittenberg and Franz Rothlauf. 2023. Small solutions for real-world symbolic regression using denoising autoencoder genetic programming. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 101–116.

[39] David Wittenberg, Franz Rothlauf, and Christian Gagné. 2023. Denoising autoencoder genetic programming: strategies to control exploration and exploitation in search. *Genetic Programming and Evolvable Machines* 24, 2 (2023), 17.

[40] David Wittenberg, Franz Rothlauf, and Dirk Schweim. 2020. DAE-GP: denoising autoencoder LSTM networks as probabilistic models in estimation of distribution genetic programming. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 1037–1045.