Asynchronous Neuro-Evolutionary Symbolic Regression with Maturity-Based Replacement

Jiří Kubalík
CIIRC, CTU in Prague, Czech Republic
jiri.kubalik@cvut.cz

Naďa Fučelová
CIIRC, CTU in Prague, Czech Republic
nada.fucelova@cvut.cz

Robert Babuška
Dept. of Cognitive Robotics, TU Delft, The Netherlands, CIIRC, CTU in Prague, Czech Republic
r.babuska@tudelft.nl

Abstract

We consider a neuro-evolutionary symbolic regression, an approach in which mathematical formulas are internally represented by feed-forward neural networks. Gradient-free global exploration of the network topology space is combined with gradient-based parameter learning. From the perspective of evolutionary algorithms, this approach has a potential weakness: newly generated networks produced by genetic operators usually require intensive parameter tuning. If this optimization has not yet converged, such networks may be prematurely discarded due to poor performance, thereby losing the opportunity to contribute their true potential to the evolutionary process. Such a waste of innovative, potentially promising topologies can reduce the algorithm's efficiency. To address this limitation, we propose a neuro-evolutionary algorithm that relies on asynchronous refinement of network parameters and a maturity-based replacement strategy. Thus, newly created networks are allowed to coexist and develop alongside their parental networks, while being compared only with peers of similar maturity. The proposed method was evaluated on six benchmarks and compared to the original evolutionary approach.

CCS Concepts

• **Computing methodologies** → **Genetic programming**; **Genetic algorithms**; **Neural networks**.

Keywords

Symbolic regression, Neuroevolution, Multi-objective optimization

ACM Reference Format:
Jiří Kubalík, Naďa Fučelová, and Robert Babuška. 2026. Asynchronous Neuro-Evolutionary Symbolic Regression with Maturity-Based Replacement. In *Genetic and Evolutionary Computation Conference (GECCO '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3795095.3805164

1 Introduction

Symbolic regression (SR) automatically constructs mathematical equations that best describe a given dataset. It relies on searching the space of free-form mathematical expressions, simultaneously optimizing both the model structure and its parameters.

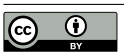
While genetic programming [3, 5, 15] was traditionally among the most widely used SR methods, the past decade has seen the emergence of approaches based on feedforward neural networks [13, 16, 21, 33], more recently, transformers [2, 20, 31, 32], and large language models [22, 28].

This paper addresses the neuro-evolutionary approach to symbolic regression in which mathematical formulas are internally represented by feed-forward neural networks. The method combines gradient-free evolutionary exploration of the network topology space with gradient-based parameter learning. Throughout the evolutionary process, new individuals are created from existing ones using genetic operators such as crossover and mutation, and are evaluated based on their performance. The best-performing models are kept in the population for the next generation.

Although this evolutionary procedure has proven effective in many domains, it suffers from the following limitation in symbolic regression. Structural changes introduced by genetic operators, particularly crossover, often modify the structure of newly generated models at the expense of their immediate performance. Such degradation arises because the coefficients require additional tuning. Consequently, model structures with strong long-term potential may be prematurely eliminated due to poor initial performance. As a result, the search process is driven by short-term performance gains at the expense of exploring individuals who can develop more promising solutions over the long term. If such solutions are suppressed, the population's diversity can be easily lost.

Various strategies have been proposed to address the above weakness, including diversity-preserving mechanisms, niching [8], and multi-objective optimization [16, 17]. These approaches can delay convergence and encourage exploration, but the survival of an individual is still judged at a single point in its lifetime, i.e., when it is created via genetic operators. Even if the individual were given the opportunity for deeper parameter tuning, it would be hard to define in advance the computational budget needed to properly tune its coefficients.

To address this challenge, we propose a new evolutionary framework for symbolic regression based on open-ended refinement of model parameters and a maturity-based replacement strategy, hereafter referred to as ANSR. The primary concept of this approach is that an individual's tuning is not limited to a one-time evaluation, but rather continues throughout their lifetime. From the moment of creation, each individual is subject to continuous optimization that remains active until there is clear evidence that further fine-tuning would no longer be valuable. With the aim of reducing the risk of prematurely discarding valuable structures and encouraging the



This work is licensed under a Creative Commons Attribution 4.0 International License.
GECCO '26, San Jose, Costa Rica
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2487-9/2026/07
https://doi.org/10.1145/3795095.3805164

exploration of structurally rich candidate solutions, offspring individuals are allowed to coexist and develop in parallel with the main population. Instead of making premature judgments, this strategy enables individuals to be refined to an extent that reveals their true quality – or the absence of it. The main contributions of the paper are:

- A novel neuro-evolutionary symbolic regression method featuring asynchronous lifelong optimization and a population replacement strategy that supports independent offspring development and more reliable search-space exploration. We call the method ANSR (Asynchronous Neuro-Evolutionary Symbolic Regression).
- Experimental results demonstrating that the proposed open-ended method outperforms standard neuroevolution in both solution quality and computational efficiency.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 defines the symbolic regression problem considered in this paper, and Section 4 describes the proposed method. Section 5 presents the experiments and the results obtained, and Section 6 concludes the paper.

2 Related work

Evolutionary algorithms (EAs) applied to symbolic regression typically fall into two categories: generational and steady-state. In generational EAs, the entire population of candidate solutions is replaced at each iteration. The offspring are produced through crossover and mutation, evaluated, and subsequently used to form the next generation. In contrast, steady-state EAs replace only a small subset of the population at a time. By allowing new offspring to coexist with older individuals, they maintain a continuously evolving population and ensure a smoother transition between generations. However, in both paradigms, an individual's survival is determined solely by its fitness after creation. This single-point evaluation tends to favor individuals who perform well early on, thereby biasing the search toward short-term improvements. Consequently, promising individuals with weak initial fitness but high long-term potential can be prematurely discarded.

To mitigate this issue, the age-layered population structure (ALPS) [10, 11, 23, 35] was proposed. ALPS defines age as the number of generations during which an individual's genetic material has persisted. The population is divided into age layers, with competition and reproduction restricted to individuals within the same age group. Meanwhile, new individuals are generated randomly and inserted into the youngest layer. By modulating the selection pressure with age, ALPS allows younger individuals to mature before competing directly with older, more optimized members of the population. A related idea was adopted in AmoebaNet [26], a neural architecture search method based on regularized (aging) evolution, in which new architectures are generated by mutation and the oldest individuals are removed from the population to promote continued exploration. Speciation in NEAT [29] follows a related principle: individuals are grouped according to structural similarity and compete primarily within their own species. This protects newly emerging topological innovations from early elimination and allows them to mature before competing with more optimized structures.

Since symbolic regression aims to identify models that fit the data well while remaining sufficiently simple, multi-objective evolutionary algorithms are widely employed. These approaches evaluate trade-offs between multiple objectives, most commonly the prediction error and model complexity [7, 16]. This strategy maintains a diverse set of candidate solutions, ranging from simple formulas to more complex, highly accurate models. Similarly, additional criteria have been incorporated as optimization objectives, including physical plausibility constraints [16, 17] and age [27]. By doing so, these methods aim to preserve diversity and prevent premature elimination of individuals.

An alternative approach to mitigating premature elimination is to allow each candidate to improve before its performance is assessed. To this end, hybrid genetic programming methods have been proposed that integrate parameter tuning directly into the evolutionary process, e.g., nonlinear least-squares parameter tuning in each generation [14]. Other methods perform gradient-based fine-tuning, such as the EN4SR method [16], which evolves neural network-style expressions using genetic operators and, within each generation, runs back-propagation to adjust the network's weights.

Recent research has moved beyond fixed-objective optimization toward open-ended evolution, emphasizing continual innovation over convergence. Instead of driving the population toward a single global optimum, open-ended algorithms aim to persistently generate novel and diverse candidates. Techniques such as novelty search [1, 9, 18] and quality diversity algorithms [4, 24, 25, 30, 34] have demonstrated the advantages of promoting exploration over exploitation, enabling the discovery of different solutions. These methods maintain a balance between adaptability and performance, ensuring that the system can respond effectively to dynamic environments.

3 Problem definition

The objective of our approach is to identify a compact and interpretable mathematical model that accurately approximates an unknown nonlinear mapping $\phi : \mathbb{R}^n \to \mathbb{R}$, with n the number of input variables. The desired model must balance predictive accuracy, adherence to domain-specific constraints, and structural sparsity.

Internally, the methods encodes mathematical equations using neural networks $f(x, w, \theta)$, parametrized by the connection weights w and topological parameters θ. We use the following optimization objectives:

(1) Validation accuracy, expressed by the root mean square error (RMSE) on a validation data set;
(2) Low model complexity, defined by the number of active links and units used in the network.

Additional objectives, such as physical constraint satisfaction, can be easily included, e.g., as in [16], but we omit them here for brevity.

4 Method

Following the neuro-evolutionary approach proposed in [16], we adopt the concept of master topology and subtopologies to represent and manipulate analytic expressions as neural networks. The master topology serves as a template for all candidate solutions, i.e., subtopologies created and evolved during the run. It is based on a fully connected feedforward structure with units (nodes)

implementing mathematical operators and elementary functions, augmented with *copy units* and the associated *skip connections*. The copy units in layer k just copy (via the skip connections) the outputs of the units from the previous layer $k-1$, so that the computation realized by some unit in an early layer may be reused in subsequent layers.

Initially, all weights (edges), except the skip connections, are set to be trainable via backpropagation. Skip connections can only assume values 0 or 1 and are never tuned. For the sake of pruning (complexity reduction), each edge can be marked as enabled or disabled, where disabled edges are excluded from training. A trainable edge transitions from the enabled state to the disabled state once its absolute weight drops below a user-defined threshold.

The goal is to evolve a subtopology that fits the training data well, is physically plausible, and has minimal complexity. This implies that most units and their input weights are eventually disabled and therefore do not contribute to the analytic formula represented by the subtopology.

Figure 1 shows an example of a subtopology with two hidden layers representing the expression $c_0 \cdot \frac{c_1 \cdot (c_2 \cdot x_0) \cdot (c_3 \cdot x_1)}{c_4 \cdot x_0 + c_5 \cdot x_1}$. Blue edges denote trainable links, and red edges are skip connections. After training, only the bold edges remain enabled; therefore, only these links and their associated units form the analytic formula. Still, expressions may be over-parametrized, as this one is. Symbolic simplification, explained later, is applied throughout the procedure.

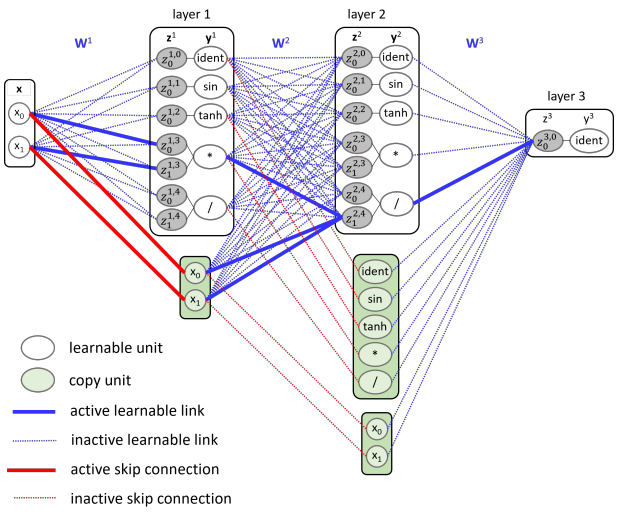


Figure 1: An example neural network representing the expression $y(x_0, x_1) = c_0 \cdot (c_1 \cdot (c_2 \cdot x_0) \cdot (c_3 \cdot x_1))/(c_4 \cdot x_0 + c_5 \cdot x_1)$.

Below, we describe extensions of the above method toward Asynchronous Neuro-Evolutionary Symbolic Regression (ANSR).

4.1 Individual

The basic data structure of the algorithm is an individual associated with its subtopology. Following [16], the refinement of an individual's parameters is performed in epochs, where the epoch length is determined by a user-defined parameter `EPOCH_ITERS`. Besides the subtopology, an individual contains the following additional elements:

- *age* – the total number of backpropagation (BP) iterations that the individual has consumed so far while tuning its subtopology parameters.
- *perfHistory* – a data structure that stores the evolution of the individual's performance metrics. Each record has the form [*age*, RMSE_v, RMSE_c, *complexity*, *weights*] and the records are ordered in ascending order of *age* at which they were obtained. Each entry also stores the subtopology weights corresponding to the recorded performance values. The first entry represents the individual's quality at the time of its creation, whether during initialization or resulting from crossover and mutation, while the last entry reflects its current quality. During each tuning epoch, records are maintained for two types of improvements: dominant and partial. A dominant improvement occurs when the performance metrics, RMSE_v and RMSE_c, obtained with the updated network weights dominate those of the original network entering the tuning epoch. A partial improvement is defined as any update that improves at least one performance metric of the network. At the end of each tuning epoch, the network weights and performance metrics corresponding to the most recent dominant improvement are stored as a new entry in the individual's *perfHistory*. If no dominant improvement occurs, the values from a randomly selected partial improvement are stored. If neither dominant nor partial improvement is detected, the original weights and performance metrics with which the individual entered the tuning epoch are stored, with only the *age* value updated accordingly.
- *subjectToTuning* – this Boolean parameter specifies whether the individual is allowed to undergo parameter tuning.
- *readiness* – this flag is set to *true* if the individual has completed at least n_r backprops, otherwise it is set to *false*; n_r is a user-defined parameter. The value of *true* indicates that the individual is an adult and therefore can participate in all routines described below. If it is *false*, the individual must not be discarded yet, regardless of its current performance.

4.2 Populations

The method maintains three populations of individuals: *mainpop*, *offsprings*, and *archive*. The first two populations have fixed size, whereas *archive* is not limited in size.

The *mainpop* contains individuals that are used to generate new individuals via crossover and mutation. Triggers for generating new solutions are described in Section 4.3. Each individual has a *true* or *false* value of *subjectToTuning*, which determines whether it undergoes further parameter tuning. In both cases, individuals remain in this population until they are removed either as a result of a merge operation or when they are dominated by another individual in *mainpop*.

Mutual dominance between two individuals is assessed using the standard concept of dominance: a solution x_1 dominates x_2 if x_1 is no worse than x_2 in all objectives and x_1 is strictly better than x_2 in at least one objective. In addition, the individual's age is incorporated into the dominance assessment as follows. An individual

x_1 can dominate an individual x_2 only if x_1 dominates x_2 based on performance records corresponding to $age = min(x_1.age, x_2.age)$. This ensures that individuals with fewer tuning epochs are not disadvantaged relative to those with more.

The procedure for merging *mainpop* with *offsprings* proceeds as follows. First, all individuals from *mainpop* are inserted into an intermediate population. Next, individuals from *offsprings* whose value in at least one performance metric out of $RMSE_v$, $RMSE_c$, and *complexity*, improves upon the worst value of that metric observed in *mainpop* are added to the intermediate population. Finally, MAINPOP_SIZE individuals are selected from the intermediate population using non-dominated sorting using all performance metrics.

The *offsprings* population contains newly generated individuals. All individuals in this population undergo epoch-wise parameter tuning until it is determined that a solution is either unsuitable for further tuning or that its quality is sufficient to be promoted to *mainpop*. The first case occurs when a given individual is dominated by another individual in the *offsprings* population. The second case occurs when an individual from *offsprings* dominates an individual in *mainpop*; in this case, it replaces the dominated individual in *mainpop*.

The *archive*, as the name suggests, contains non-dominated solutions among all solutions generated up to the given point in the computation. Individuals in this population do not actively participate in the optimization process. Only at the very end of the computation is the final solution selected from this set.

4.3 Pseudo-code

In this section, we describe the execution of the ANSR algorithm, with an emphasis on its main differences from the standard evolutionary algorithm SEASR, see Algorithms 1 and 2. The main feature of the ANSR algorithm is the asynchronous tuning of individuals in *mainpop* and *offsprings*. In Algorithm 1, actions related to this mechanism are highlighted in blue, and we use the abbreviation "apu" to denote an *asynchronous parallel processing unit*. In contrast, in the SEASR algorithm, individual tuning is performed synchronously, alternating between parallel tuning of all individuals in *mainpop* and parallel tuning of all individuals in *offsprings*. Accordingly, we use the abbreviation "spu" to denote a *synchronous parallel processing unit*.

The ANSR algorithm starts by initializing *mainpop*, which involves randomly generating MAINPOP_SIZE subtopologies and performing their initial parameter tuning for INIT_ITERS iterations. Next, the variables *apu.queue* and *toProcess* are initialized, where *apu.queue* is a queue of individuals to be processed through a single tuning epoch, and *toProcess* is a list of *mainpop* individuals that must undergo at least one tuning epoch before the merging of *mainpop* and *offsprings* can be performed. Both *archive* and *offsprings* are initially empty, and the number of consumed BP iterations, *iters*, is set accordingly. The equivalent of this initialization procedure in the SEASR algorithm corresponds to lines 1, 2, and 5.

The main loop runs until the stopping condition is met – in this case, until the total budget of BP iterations has been exhausted. At the beginning of each iteration, the algorithm waits until the tuning of at least one individual is completed. Once this occurs (line 11), the following actions are performed. First, the individual's

Algorithm 1: ANSR

Input: MAINPOP_SIZE ... size of *mainpop*
OFFSPRING_SIZE ... size of *offsprings*
INIT_ITERS ... initial BP iterations
EPOCH_ITERS ... nb. of BP iterations in learning epoch
OFFSPRING_ITERS ... BP iteration per offspring
MAX_ITERS ... total BP iterations per run

Output: best model evolved

```
1  mainpop.init();
2  apu.queue.init(mainpop)
3  toProcess.init(mainpop);
4  offsprings ← {}
5  archive ← {}
6  iters ← INIT_ITERS * MAINPOP_SIZE
7  while iters < MAX_ITERS do
8      if apu.completed.isEmpty() then
9          continue
10     iters ← iters + EPOCH_ITERS
11     ind ← apu.completed.pop()
12     ind.updatePerfHistory()
13     ind.updateSubjectToTune()
14     archive.update(ind)
15     if ind.memberOf == '-' then
16         if not(offsprings.isfull()) then
17             ind.memberOf ← 'offsprings'
18             offsprings.add(ind)
19             apu.queue.appendLeft(ind)
20     else if ind.memberOf == 'mainpop' then
21         child ← breed (mainpop)
22         child.memberOf ← '-'
23         apu.queue.appendLeft(child)
24         if ind.isDominatedBy(mainpop) then
25             mainpop.remove(ind)
26         else
27             apu.queue.appendRight(ind)
28         toProcess.remove(ind)
29         if toProcess.isEmpty() then
30             mainpop ← merge(mainpop, offsprings)
31             toProcess.init(mainpop);
32     else if ind.memberOf == 'offsprings' then
33         if not(ind.subjectToTuning) OR
            ind.isDominatedBy(offsprings) OR ind.age >
            OFFSPRING_ITERS then
34             offsprings.remove(ind)
35         else
36             apu.queue.appendLeft(ind)
37     if not(apu.queue.isEmpty()) then
38         ind ← apu.queue.pop()
39         apu.submit(ind, EPOCH_ITERS)
40 return archive.selectBest()
```

age and *perfHistory* are updated. Next, it is checked whether the individual is suitable for further tuning. Specifically, if the individual has not improved over the last k tuning epochs, its *subjectToTuning* parameter is set to *false*. Subsequently, the *archive* is updated with the current version of this individual.

Then, different actions are performed depending on the population to which the individual *ind* is currently assigned. If the individual is newly generated through crossover and mutation, it is added to *offsprings* and simultaneously inserted at the front of *apu.queue*. This increases the likelihood that the new individual will be tuned within the current `merge`-cycle. If the individual belongs to *mainpop*, a new individual is generated (line 21). This involves selection, crossover, and mutation. Here, we use the same genetic operators as in [16], in particular tournament selection and variation operators that modify both the topology and the weights of the subtopologies. Further details are beyond the scope of this paper; we refer the reader to [16]. The newly generated *child* is added to the front of *apu.queue*.

The individual *ind* is then compared against the other individuals in the population to determine whether any of them dominates it. If there exists an individual in the population that dominates the individual *ind*, then *ind* is removed from *mainpop*. Otherwise, the individual is added to the end of *apu.queue*. Note that the individual is compared with another individual based on the values observed at an age equal to the minimum of the ages of the two individuals. Therefore, the individual *ind* cannot be removed from the population if it is dominated by another individual that is older and achieved its superior performance values at a later age than the current age of *ind*. Finally, the individual *ind* is removed from the *toProcess* list, and it is checked whether *toProcess* is empty. If so, this triggers the `merge` operation, which is carried out according to the procedure described in Section 4.2, and *toProcess* is reinitialized with the updated contents of *mainpop*.

The last remaining option is that the individual *ind* is assigned to *offsprings*. In this case, it is checked whether the individual is still suitable to remain in *offsprings*. Specifically, it is tested whether the individual has its *subjectToTuning* attribute set to *false*, whether it is dominated by another individual in this population, or whether it has already used its total budget of BP iterations. If any of these conditions holds, the individual is removed from *offsprings*. Otherwise, it is added to the front of *apu.queue*, thereby granting it another tuning epoch.

At the end of the main loop, an individual is retrieved from the head of *apu.queue*, and its tuning epoch is initiated. Once the total budget for BP iterations across the entire run is exhausted, the final solution must be chosen from the *archive*. This is a multi-objective decision-making problem, as the models are evaluated using two quality indicators: RMSE and complexity. To select the final solution, we use a strategy inspired by the Minimal-Complexity Knee Selection method [36]. First, we calculate the bend angle for each model in the *archive* based on its adjacent left and right neighbors on the bi-objective front. Then, we use K-means to cluster the non-dominated models into k groups based on their bend angles and identify the cluster with the highest mean angle. From the winning cluster, we select the model with the lowest RMSE. Here, we use the validation RMSE and $k = 2$.

The main difference between the proposed `ANSR` method and `SEASR` lies in how newly generated candidate solutions are integrated into the optimization process. In `SEASR`, this occurs only during the `merge` step (line 25) and only after completing an initial tuning phase. In other words, the chances of each individual depend solely on how well its parameters can be tuned during this initial phase. In contrast, in `ANSR`, each new individual is given substantially more opportunity to reach a sufficiently good state before the decision is made on whether it survives. This concept is based on eliminating individuals that are demonstrably poor from *offsprings*, while better-performing individuals survive and are given more opportunity for further improvement.

To promote convergence toward simple expressions, i.e., small *complexity*, each tuning epoch is preceded by a symbolic simplification step applied to the expression represented by the individual's subtopology. This step is applied only when the individual's *complexity* falls below a user-defined threshold *maxToSimplify*, as simplification can become computationally expensive for larger networks. To do so, the mathematical expression represented by the individual is extracted and simplified using the SymPy[1] library. If the simplification is completed within the specified timeout, the simplified expression is then mapped back onto the individual's network structure. Both compared algorithms employ this simplification routine.

Algorithm 2: `SEASR`

Input: `MAINPOP_SIZE` ... size of *mainpop*
`INIT_ITERS` ... initial BP iterations
`EPOCH_ITERS` ... nb. of BP iterations in learning epoch
`MAX_ITERS` ... total BP iterations per run

Output: best model evolved

```
1  mainpop.init();
2  iters ← INIT_ITERS * MAINPOP_SIZE
3  while iters < MAX_ITERS do
       # synchronous tuning of mainpop individuals
4      toProcess.init(mainpop);
5      spu.submit(mainpop, EPOCH_ITERS)
6      while not(toProcess.isEmpty()) do
7          if spu.completed.isEmpty() then
8              continue
9          iters ← iters + EPOCH_ITERS
10         ind ← spu.completed.pop()
11         if iters ≥ MAX_ITERS then
12             break
13         toProcess.remove(ind)
       # generation and synchronous tuning of offsprings
         individuals
14     offsprings.breed(mainpop)
15     toProcess.init(offsprings);
16     spu.submit(offsprings, EPOCH_ITERS)
17     while not(toProcess.isEmpty()) do
18         if spu.completed.isEmpty() then
19             continue
20         iters ← iters + EPOCH_ITERS
21         ind ← spu.completed.pop()
22         if iters ≥ MAX_ITERS then
23             break
24         toProcess.remove(ind)
25     mainpop ← merge(mainpop, offsprings)
26 return mainpop.selectBest()
```

[1] https://www.sympy.org/

5 Experiments

We compare the proposed method with the standard neuro-evolution SEASR using three synthetic test cases from the literature, Nguyen-9c, Nguyen-10c, and Jin-6 [12, 19]. In addition, we include three physics-based test cases: parallel resistors, rotational motion under dry friction, and longitudinal drone dynamics. For the code, data, and additional information regarding the experiments, refer to our GitHub repository https://github.com/jirkakubalik/ANSR.

Note that the purpose of the comparison is to isolate the effect of asynchronous refinement. Therefore, we compare to the same algorithmic framework (SEASR) with synchronous updates. A direct comparison with GP-based methods would be non-trivial due to differences in representation (trees vs neural networks) and parameter-optimization mechanisms.

5.1 Experiment setup

Nguyen-9c (`Ng-9c`) is defined as $f(x_1, x_2) = \sin(1.5x_1) + \sin(0.5x_2^2)$. The training, validation, and test datasets were randomly sampled from a uniform distribution over (0, 1). All three datasets have 100 samples each. The master topology has two hidden layers and an identity output node. The hidden layers contain elementary functions {sin, tanh, square, ident, $*$}, each in two copies.

Nguyen-10c (`Ng-10c`) is defined as $f(x_1, x_2) = \sin(1.5x_1)\cos(0.5x_2)$ The training, validation, and test datasets were sampled in the same way as in the case of problem `Ng-9c`. The master topology used for this problem is the same as that used for `Ng-9c`.

`Jin-6` is defined as $f(x_1, x_2) = 1.35x_1x_2 + 5.5\sin((x_1 - 1)(x_2 - 1))$ The training, validation, and test datasets, all of size 100, were drawn uniformly from the interval (-3, 3). In this case as well, the same master topology as in the Nguyen problems was used.

Parallel resistors. Originally proposed in [3], this test case aims to learn a model of the equivalent resistance of two parallel resistors, $r = r_1r_2/(r_1 + r_2)$, from noisy data. The training set (400 samples) and the validation set (100 samples) are drawn uniformly from $\mathbb{D}_t = [0.0001, 20]$, using the above well-known formula as ground truth and adding noise to the samples. For this problem, we used the same master topology with three hidden layers as in [16]. The first two hidden layers consist of four elementary functions {sin, tanh, ident, $*$}, each function represented by two copies. The third hidden layer includes a single division unit.

Rotational motion under dry friction. Real-world data from a laboratory servo positioning system with adjustable friction are used to test the method on a system identification task of a highly nonlinear system. The setup consists of a voltage-controlled DC motor directly driving a vertical metal disc with an additional off-center weight, introducing the effect of gravity. The nominal system has minimal, predominantly viscous friction, proportional to the angular velocity. However, additional manually adjustable dry friction is introduced by pressing a mechanical brake against the back of the disc, which introduces non-trivial nonlinear behavior. The goal is to extract from the measured data a differential equation describing the disc's angular acceleration as a function of angular velocity and disc angle. The training set (851 samples) and the validation set (851 samples) were collected in a series of experiments in which the disc was driven to a non-zero initial condition (angle and angular velocity) and the resulting response for zero input (DC motor voltage) was recorded with a sampling period of 0.01 s. The data are inherently noisy, primarily due to the quantization of the incremental encoder measuring the disc angle. For this problem, we used a master topology with two hidden layers and a simple identity function in the output layer. In the hidden layers, three elementary functions were used, {sin, tanh, $*$}, each represented by two copies.

Drone dynamics. A custom-built quadcopter was controlled by a model predictive controller to follow a circular trajectory at constant altitude. The task is to use real-world data to construct a dynamic model. The measured states include the translational velocities (captured via an OptiTrack system in an indoor flying arena) and the body angles pitch, roll, and yaw (captured by an inertial measurement unit, IMU, of the drone), all expressed in the world frame. Both the training set and the validation set consist of 501 samples. The sampling period is $T_s = 0.03$ s. The data are inherently noisy, due to the IMU and OptiTrack sensor noise. The goal is to extract from the data a differential equation describing the translational acceleration as a function of the translational velocity and pitch. For this problem, we used a master topology with two hidden layers and a simple identity function in the output layer. In the hidden layers, four elementary functions were used, {sin, sign, square, $*$}, each represented by two copies.

5.2 Results and discussion

All results presented in this section are based on 30 independent runs for each experiment with a given method and dataset. First, we analyze the algorithms' performance in terms of the RMSE observed on test data (RMSE_t), independently of the final model selection method. For this purpose, we define for each problem a target region based on the maximum acceptable values of RMSE_t and model complexity (see Figure 2). Model complexity is defined as the number of active nodes in the network of a given solution. The target region serves as a filter to select relevant solutions from the set of all solutions in the final archive, aggregated across all 30 independent runs.

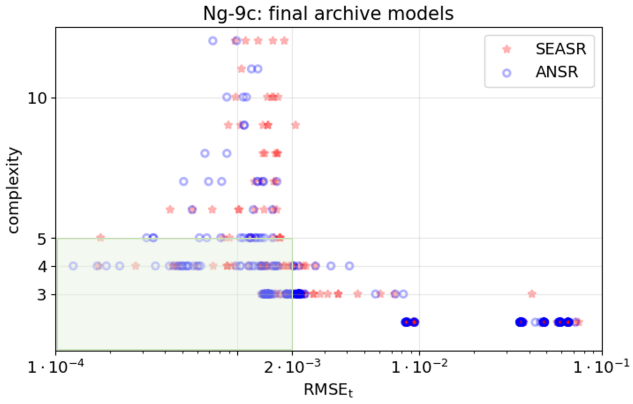


Figure 2: Example of a target region (green) for the `Ng-9c` problem.

Table 1 summarizes the success of the algorithms in finding accurate, yet simple solutions. We define three levels of RMSE_t for

the three lowest model complexities (starting from the minimum complexity required to represent the optimal solution for the given problem). The results are reported in the format "m/r", where m is the number of all models in the final archive that fall into the given region, and r indicates the number of runs in which at least one of the final archive solutions appears in the target region. It can be seen that ANSR consistently outperforms SEASR in terms of the number of models found with a given accuracy and complexity.

Table 2 provides another perspective on the performance of the algorithms by reporting the median RMSE_t values of the best models in the final archive with complexity less than or equal to a given maximum complexity. We report the p-value from the Wilcoxon rank-sum test and Cliff's delta [6] as a measure of effect size. As a general trend, we observe that ANSR produces models with lower RMSE_t than SEASR, although the difference is statistically significant only for Ng-9c, Jin-6, and drone.

Figure 3 presents the convergence curves of RMSE_t for individual runs of ANSR and SEASR (thin lines), along with their median curve (thick line), for the Ng-9c problem. The curves show the best RMSE_t value among models with complexity less than or equal to 5 in the archive as a function of the number of backpropagation iterations.

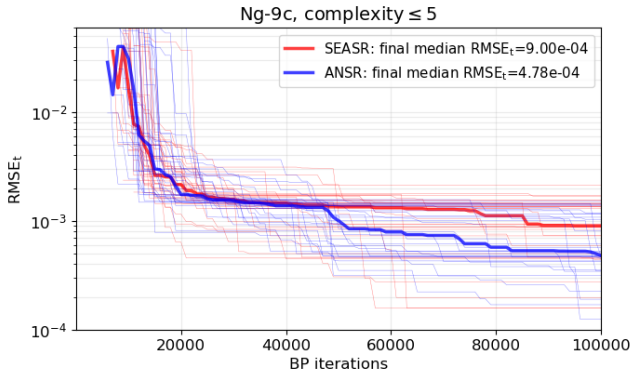


Figure 3: Convergence curves for models with maximum complexity 5 for the Ng-9c problem.

Table 3 reports the validation and test error, RMSE_v, RMSE_t, and the complexity of the final models selected by using the method described in Section 4.3. Median values over 30 independent runs are shown. Model complexity is expressed as the tuple (number of active nodes / number of active links). The results indicate that the final models are comparable across both methods.

5.3 Examples of models found by ANSR

For the three physical problems, we report typical examples of equations found by ANSR.

Within the parallel resistors dataset, a typical equation takes the form of this compact model:

$$f(r_1, r_2) = \frac{0.3523 r_1 r_2}{0.3508 r_1 + 0.3528 r_2} = \frac{r_1 r_2}{0.996 r_1 + 1.001 r_2},$$

with RMSE_v of 0.1227 and *complexity* of 4 active nodes and 7 active links. The expression closely matches the known analytical form $r = r_1 r_2/(r_1 + r_2)$, differing only by two close-to-one scaling

Table 1: Algorithm success in reaching the target regions in terms of maximum complexity and RMSE_t.

dataset	max complexity	max RMSE_t	ANSR	SEASR
Ng-9c	3	$1 \cdot 10^{-3}$	0 / 0	0 / 0
		$2 \cdot 10^{-3}$	45 / 28	27 / 19
		$3 \cdot 10^{-3}$	81 / 29	66 / 29
	4	$1 \cdot 10^{-3}$	21 / 17	13 / 13
		$2 \cdot 10^{-3}$	84 / 30	61 / 30
		$3 \cdot 10^{-3}$	117 / 30	105 / 30
	5	$1 \cdot 10^{-3}$	27 / 20	16 / 15
		$2 \cdot 10^{-3}$	102 / 30	63 / 30
		$3 \cdot 10^{-3}$	140 / 30	105 / 30
Ng-10c	3	$1 \cdot 10^{-4}$	4 / 4	0 / 0
		$5 \cdot 10^{-4}$	10 / 10	6 / 6
		$1 \cdot 10^{-3}$	10 / 10	7 / 7
	4	$1 \cdot 10^{-4}$	6 / 6	1 / 1
		$5 \cdot 10^{-4}$	14 / 14	12 / 10
		$1 \cdot 10^{-3}$	20 / 18	16 / 12
	5	$1 \cdot 10^{-4}$	6 / 6	1 / 1
		$5 \cdot 10^{-4}$	16 / 15	14 / 11
		$1 \cdot 10^{-3}$	30 / 23	23 / 16
Jin-6	4	$2 \cdot 10^{-3}$	6 / 6	2 / 2
		$3 \cdot 10^{-3}$	9 / 9	5 / 5
		$4 \cdot 10^{-3}$	19 / 19	12 / 11
	5	$2 \cdot 10^{-3}$	7 / 7	3 / 3
		$3 \cdot 10^{-3}$	11 / 11	7 / 7
		$4 \cdot 10^{-3}$	21 / 20	17 / 15
	6	$2 \cdot 10^{-3}$	7 / 7	3 / 3
		$3 \cdot 10^{-3}$	11 / 11	7 / 7
		$4 \cdot 10^{-3}$	21 / 20	17 / 15
resistors	4	$4 \cdot 10^{-3}$	3 / 3	2 / 2
		$5 \cdot 10^{-3}$	7 / 7	10 / 10
		$6 \cdot 10^{-3}$	8 / 8	13 / 12
	5	$4 \cdot 10^{-3}$	4 / 3	2 / 2
		$5 \cdot 10^{-3}$	8 / 7	10 / 10
		$6 \cdot 10^{-3}$	10 / 8	13 / 12
	6	$4 \cdot 10^{-3}$	3 / 3	2 / 2
		$5 \cdot 10^{-3}$	9 / 9	10 / 10
		$6 \cdot 10^{-3}$	11 / 10	13 / 12
friction	3	4.6	0 / 0	0 / 0
		4.75	2 / 2	1 / 1
		5.0	2 / 2	2 / 2
	4	4.6	4 / 3	4 / 3
		4.75	20 / 19	13 / 9
		5.0	35 / 20	25 / 17
	5	4.6	21 / 13	14 / 9
		4.75	50 / 24	40 / 19
		5.0	76 / 29	61 / 24
drone	2	1.6	23 / 17	6 / 6
		1.65	33 / 24	9 / 8
		1.7	33 / 24	9 / 8
	3	1.6	41 / 20	21 / 13
		1.65	60 / 29	38 / 24
		1.7	66 / 30	41 / 26
	4	1.6	75 / 26	45 / 18
		1.65	101 / 30	68 / 26
		1.7	109 / 30	73 / 27

Table 2: Comparison of the convergence statistics of ANSR and SEASR for different levels of model complexity.

dataset	max comp.	RMSE_t ANSR	RMSE_t SEASR	delta	p-value
Ng-9c	3	$1.47 \cdot 10^{-3}$	$1.84 \cdot 10^{-3}$	-0.33	0.024
	4	$5.72 \cdot 10^{-4}$	$1.35 \cdot 10^{-3}$	-0.31	0.036
	5	$4.78 \cdot 10^{-4}$	$9.0 \cdot 10^{-4}$	-0.35	0.017
Ng-10c	3	$3.46 \cdot 10^{-3}$	$3.6 \cdot 10^{-3}$	-0.09	0.540
	4	$7.26 \cdot 10^{-4}$	$1.09 \cdot 10^{-3}$	-0.20	0.187
	5	$4.98 \cdot 10^{-4}$	$6.37 \cdot 10^{-4}$	-0.22	0.139
Jin-6	4	$3.37 \cdot 10^{-3}$	$4.29 \cdot 10^{-3}$	-0.32	0.031
	5	$3.25 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	-0.26	0.076
	6	$3.25 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	-0.28	0.059
resistors	4	$3.8 \cdot 10^{-3}$	$4.12 \cdot 10^{-4}$	-0.10	0.49
	5	$3.79 \cdot 10^{-3}$	$4.12 \cdot 10^{-3}$	-0.12	0.42
	6	$3.79 \cdot 10^{-3}$	$4.12 \cdot 10^{-3}$	-0.06	0.71
friction	3	6.07	5.94	-0.01	0.98
	4	4.75	4.81	-0.20	0.19
	5	4.6	4.73	-0.25	0.10
drone	2	1.587	1.761	-0.6	0.0001
	3	1.574	1.595	-0.426	0.0045
	4	1.552	1.588	-0.433	0.0039

Table 3: Final model performance.

dataset	method	RMSE_v	RMSE_t	complexity
Ng-9c	ANSR	$1.54 \cdot 10^{-3}$	$1.45 \cdot 10^{-3}$	3.0 / 7
	SEASR	$1.57 \cdot 10^{-3}$	$1.47 \cdot 10^{-3}$	3.0 / 7
Ng-10c	ANSR	$2.62 \cdot 10^{-3}$	$3.10 \cdot 10^{-3}$	3.5 / 9
	SEASR	$1.39 \cdot 10^{-3}$	$1.33 \cdot 10^{-3}$	4.0 / 11
Jin-6	ANSR	1.25	1.40	3.0 / 9
	SEASR	1.25	1.37	3.0 / 10
resistors	ANSR	0.134	0.062	3.5 / 10
	SEASR	0.130	0.052	4.0 / 11
friction	ANSR	4.43	4.25	8.0 / 25
	SEASR	4.61	4.40	8.0 / 24
drone	ANSR	0.71	1.55	5.0 / 14
	SEASR	0.71	1.56	6.0 / 16

factors. This deviation is expected due to the noise in the training and validation data. Importantly, the resulting expression is both structurally simple and physically meaningful.

A typical equation found for the friction dataset is

$$\ddot{\alpha} = -73.1308 \sin(1.1965\,\alpha) - 37.2884 \tanh(0.764\,\alpha + 0.0644\,\dot{\alpha}) + 17.9042 \tanh\Big(-8.9004\ \tanh(-0.764\,\alpha - 0.0644\,\dot{\alpha}) - 7.1244\,\dot{\alpha}\Big) - 0.0489,$$

with RMSE_v of 5.4103 and *complexity* of 4 active nodes and 9 active links. As expected, the friction example yields a more complex expression involving nested hyperbolic tangent functions, which the method incorporated in the model to capture the complex dry friction behavior. Although the resulting model is not immediately interpretable, it remains compact relative to the underlying nonlinear dynamics and achieves a low validation error.

With the drone dynamics data, a typical example of an equation found for the longitudinal velocity in the x direction is:

$$\dot{v}_x = 8.4204\ \sin(1.571\,\theta) - 0.5027\,v_x + 14.9999\,\theta,$$

with RMSE_v of 0.7230 and *complexity* of 2 active nodes and 4 active links. The extracted model combines the sinusoidal dependence on the pitch angle with linear terms in velocity and angle. The presence of the $\sin(\theta)$ term is consistent with the well-known translational dynamics, while the linear velocity term captures aerodynamic drag. This parsimonious model with only two active nodes achieves low validation error, highlighting the effectiveness of the pruning and maturity-based selection mechanisms.

Overall, these examples demonstrate that ANSR can adapt the structural complexity of the discovered expressions to the problem's intrinsic difficulty, producing simple analytical forms when possible and richer nonlinear models when required. This behavior aligns with the objective of balancing accuracy and interpretability and supports the claim that asynchronous lifelong refinement enables the discovery of high-quality symbolic models that might otherwise be discarded prematurely.

6 Conclusions and future research

This paper introduced ANSR, an asynchronous neuro-evolutionary symbolic regression algorithm with maturity-based replacement. By allowing individuals to undergo lifelong parameter refinement, the proposed method reduces premature elimination of structurally promising candidates. The maturity-aware dominance comparison ensures fair competition between individuals at comparable stages of optimization, while asynchronous tuning improves the effective use of the available computational budget.

Experimental results on both groups of datasets demonstrate that ANSR outperforms a standard neuro-evolutionary baseline in terms of solution quality and convergence efficiency. These results confirm that delaying selection pressure through maturity-based replacement leads to more reliable exploration of the search space and improved final models. The strongest gains are observed for the drone dynamics, which are accurately described by a parsimonious, nearly linear model.

Future work will focus on hyperparameter tuning and analysis, as well as a broader class of symbolic representations, including systems with multiple inputs and outputs. We will also investigate the algorithm's scalability with respect to the number of variables and dataset size.

Finally, note that this approach may be broadly applicable to optimization problems solved using EAs, where the iterative backpropagation routine is replaced by local optimization guided by problem-specific low-level heuristics (e.g., 3-opt for routing problems).

Acknowledgments

This work has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101070254 CORESENSE. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the Horizon Europe programme. Neither the European Union nor the granting authority can be held responsible for them. This work was also partly supported by the European Union under the project ROBOPROX (reg. no. CZ.02.01.01/00/22_008/0004590).

References

[1] Ryan Bahlous-Boldi, Maxence Faldor, Luca Grillotti, Hannah Janmohamed, Lisa Coiffard, Lee Spector, and Antoine Cully. 2025. Dominated Novelty Search: Rethinking Local Competition in Quality-Diversity. arXiv:2502.00593 [cs.NE] https://arxiv.org/abs/2502.00593

[2] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. Neural Symbolic Regression that Scales. arXiv:2106.06427 [cs.LG] https://arxiv.org/abs/2106.06427

[3] Iwo Błądek and Krzysztof Krawiec. 2019. Solving symbolic regression problems with formal constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Prague, Czech Republic) *(GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 977–984. doi:10.1145/3321707.3321743

[4] J. P. Bruneton. 2025. Enhancing Symbolic Regression with Quality-Diversity and Physics-Inspired Constraints. arXiv:2503.19043 [cs.NE] https://arxiv.org/abs/2503.19043

[5] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. 2020. Operon C++: an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 1562–1570. doi:10.1145/3377929.3398099

[6] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494–509.

[7] Miles Cranmer. 2023. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. arXiv:2305.01582 [astro-ph.IM] https://arxiv.org/abs/2305.01582

[8] Fabricio Olivetti De Franca. 2023. Fighting Underspecification in Symbolic Regression with Fitness Sharing. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (Lisbon, Portugal) *(GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 551–554. doi:10.1145/3583133.3590525

[9] Fabricio Olivetti de França and Gabriel Kronberger. 2025. Improving Genetic Programming for Symbolic Regression with Equality Graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference* (NH Malaga Hotel, Malaga, Spain) *(GECCO '25)*. Association for Computing Machinery, New York, NY, USA, 989–998. doi:10.1145/3712256.3726383

[10] J. C. Felix-Saul, Mario García-Valdez, Juan J. Merelo Guervós, and Oscar Castillo. 2024. Extending Genetic Algorithms with Biological Life-Cycle Dynamics. *Biomimetics* 9, 8 (2024). doi:10.3390/biomimetics9080476

[11] Gregory S. Hornby. 2006. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (Seattle, Washington, USA) *(GECCO '06)*. Association for Computing Machinery, New York, NY, USA, 815–822. doi:10.1145/1143997.1144142

[12] Zhengyao Huang, Daniel Zhengyu Huang, Tiannan Xiao, Dina Ma, Zhenyu Ming, Hao Shi, and Yuanhui Wen. 2025. Improving Monte Carlo Tree Search for Symbolic Regression. *ArXiv* abs/2509.15929 (2025). https://api.semanticscholar.org/CorpusID:281411604

[13] Samuel Kim, Peter Y. Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Ceperic, and Marin Soljacic. 2019. Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery. *IEEE Transactions on Neural Networks and Learning Systems* 32 (2019), 4166–4177. https://api.semanticscholar.org/CorpusID:209140148

[14] Michael Kommenda, Gabriel Kronberger, Stephan Winkler, Michael Affenzeller, and Stefan Wagner. 2013. Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation* (Amsterdam, The Netherlands) *(GECCO '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1121–1128. doi:10.1145/2464576.2482691

[15] John R. Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA.

[16] Jiří Kubalík and Robert Babuška. 2025. Neuro-Evolutionary Approach to Physics-Aware Symbolic Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference* (NH Malaga Hotel, Malaga, Spain) *(GECCO '25)*. Association for Computing Machinery, New York, NY, USA, 1264–1272. doi:10.1145/3712256.3726434

[17] Jiří Kubalík, Erik Derner, and Robert Babuška. 2020. Symbolic regression driven by training data and prior knowledge. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 958–966. doi:10.1145/3377930.3390152

[18] Joel Lehman and Kenneth O. Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.* 19, 2 (June 2011), 189–223. doi:10.1162/EVCO_a_00025

[19] Wenqiang Li, Weijun Li, Lina Yu, Min Wu, Linjun Sun, Jingyi Liu, Yanjie Li, Shu Wei, Yusong Deng, and Meilan Hao. 2024. A Neural-Guided Dynamic Symbolic Network for Exploring Mathematical Expressions from Data. arXiv:2309.13705 [cs.LG] https://arxiv.org/abs/2309.13705

[20] Yanjie Li, Jingyi Liu, Min Wu, Lina Yu, Weijun Li, Xin Ning, Wenqiang Li, Meilan Hao, Yusong Deng, and Shu Wei. 2025. MMSR: Symbolic regression is a multi-modal information fusion task. *Information Fusion* 114 (2025), 102681. doi:10.1016/j.inffus.2024.102681

[21] Georg Martius and Christoph H. Lampert. 2016. Extrapolation and learning equations. arXiv:1610.02995 [cs.LG] https://arxiv.org/abs/1610.02995

[22] Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. 2024. In-Context Symbolic Regression: Leveraging Large Language Models for Function Discovery. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*. Association for Computational Linguistics, 589–606. doi:10.18653/v1/2024.acl-srw.49

[23] Andrew Pozzuoli and Brian Ross. 2023. Increasing Features in MAP-Elites Using an Age-Layered Population Structure. In *2023 IEEE Congress on Evolutionary Computation (CEC)*. 1–8. doi:10.1109/CEC53210.2023.10254093

[24] Justin Pugh, Lisa Soros, and Kenneth Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* 3 (07 2016). doi:10.3389/frobt.2016.00040

[25] Chao Qian, Ke Xue, and Ren-Jian Wang. 2024. Quality-diversity algorithms can provably be helpful for optimization. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence* (Jeju, Korea) *(IJCAI '24)*. Article 773, 9 pages. doi:10.24963/ijcai.2024/773

[26] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 587, 10 pages. doi:10.1609/aaai.v33i01.33014780

[27] Michael D. Schmidt and Hod Lipson. 2010. Age-fitness pareto optimization. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (Portland, Oregon, USA) *(GECCO '10)*. Association for Computing Machinery, New York, NY, USA, 543–544. doi:10.1145/1830483.1830584

[28] Samiha Sharlin and Tyler R. Josephson. 2025. In Context Learning and Reasoning for Symbolic Regression with Large Language Models. arXiv:2410.17448 [cs.CL] https://arxiv.org/abs/2410.17448

[29] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evol. Comput.* 10, 2 (June 2002), 99–127. doi:10.1162/106365602320169811

[30] Yongjun Sun, Yinxia Wu, and Zujun Liu. 2024. An improved differential evolution with adaptive population allocation and mutation selection. *Expert Systems with Applications* 258 (2024), 125130. doi:10.1016/j.eswa.2024.125130

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

[32] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. 2024. SymFormer: End-to-End Symbolic Regression Using Transformer-Based Architecture. *IEEE Access* 12 (2024), 37840–37849. doi:10.1109/ACCESS.2024.3374649

[33] Matthias Werner, Andrej Junginger, Philipp Hennig, and Georg Martius. 2021. Informed Equation Learning. arXiv:2105.06331 [cs.LG] https://arxiv.org/abs/2105.06331

[34] Meng Xu, Yi Mei, Fangfang Zhang, and Mengjie Zhang. 2022. Genetic programming with diverse partner selection for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 615–618. doi:10.1145/3520304.3528920

[35] Kaifeng Yang, Bernhard Werth, and Michael Affenzeller. 2025. *Age-Layer-Population-Structure with Self-adaptation in Optimization*. 3–11. doi:10.1007/978-3-031-83885-9_1

[36] Hengzhe Zhang, Qi Chen, Bing Xue, Wolfgang Banzhaf, and Mengjie Zhang. 2024. Improving Generalization of Evolutionary Feature Construction with Minimal Complexity Knee Points in Regression. In *Genetic Programming*, Mario Giacobini, Bing Xue, and Luca Manzoni (Eds.). Springer Nature Switzerland, Cham, 142–158.