Neuro-Evolutionary Approach to Physics-Aware Symbolic Regression

Jiří Kubalík
Czech Institute of Informatics, Robotics, and Cybernetics
CTU in Prague, Prague, Czech Republic
jiri.kubalik@cvut.cz

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

Abstract

Symbolic regression is a technique that can automatically derive analytic models from data. Traditionally, symbolic regression has been implemented primarily through genetic programming that evolves populations of candidate solutions sampled by genetic operators, crossover and mutation. More recently, neural networks have been employed to learn the entire analytical model, i.e., its structure and coefficients, using regularized gradient-based optimization. Although this approach tunes the model's coefficients better, it is prone to premature convergence to suboptimal model structures. Here, we propose a neuro-evolutionary symbolic regression method that combines the strengths of evolutionary-based search for optimal neural network (NN) topologies with gradient-based tuning of the network's parameters. Due to the inherent high computational demand of evolutionary algorithms, it is not feasible to learn the parameters of every candidate NN topology to the full convergence. Thus, our method employs a memory-based strategy and population perturbations to enhance exploitation and reduce the risk of being trapped in suboptimal NNs. In this way, each NN topology can be trained using only a short sequence of back-propagation iterations. The proposed method was experimentally evaluated on three real-world test problems and has been shown to outperform other NN-based approaches regarding the quality of the models obtained.

CCS Concepts

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

Keywords

Symbolic regression, Neuroevolution, Physics-aware modeling, Multi-objective optimization

ACM Reference Format:
Jiří Kubalík and Robert Babuška. 2025. Neuro-Evolutionary Approach to Physics-Aware 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/10.1145/3712256.3726434

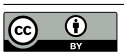
1 Introduction

Symbolic regression (SR) is a data-driven technique that creates models in the form of mathematical expressions. It has achieved remarkable results in various nonlinear modeling problems [10, 29, 31, 35]. Traditionally, SR has been implemented using genetic programming (GP) [1, 6, 7, 18, 31, 32]. An evolutionary algorithm refines a population of candidate solutions over multiple generations, guided by a selection mechanism that favors high-quality solutions over weaker ones. New candidate models are generated through genetic operations such as crossover and mutation. More recent GP-based methods also incorporate the loss function gradient to fine-tune the model's internal coefficients [17, 33, 45].

Symbolic regression offers several benefits compared to other data-driven modeling techniques. Unlike deep neural networks, which typically require large amounts of data to perform well, SR can generate accurate models even from a limited number of training samples [11, 42]. Furthermore, SR is well suited to integrate prior knowledge about the system being modeled [6, 21, 22]. This is valuable when the dataset does not uniformly cover the input space or when there are no data samples in certain regions of the input space. Even with a sufficiently large and diverse dataset, methods that focus solely on training error minimization can produce inaccurate models. For instance, in the case of dynamic system models, errors typically occur in steady-state behavior or incorrect models of local dynamics.

Although GP-based symbolic regression methods are widely used, they have several drawbacks. One major issue is the tendency of the model to grow in size and complexity without a corresponding performance improvement, a phenomenon referred to as code bloat [34]. In addition, these methods become inefficient for large numbers of variables and samples in the training dataset. This is because they continuously evolve and evaluate an entire population of formulas. Finally, fine-tuning the model coefficients solely through genetic operators proved to be quite challenging for the pure GP methods.

To address the challenges outlined above, various methods have recently been developed that leverage neural networks (NNs) to learn analytic formulas through gradient-based optimization techniques [8, 15, 26, 30, 40, 44]. These methods share a common principle: analytic models are represented by a heterogeneous NN, whose units perform mathematical operations and elementary functions such as {+, -, *, /, sin, exp, etc.}. Network weights are optimized using standard gradient-based algorithms to minimize the training error while significantly reducing the number of active units, i.e., the units that contribute to the output, through regularization. The resulting NN then represents a parsimonious analytic formula. Individual approaches differ primarily in how they guide the learning process toward the final model. Another NN-based symbolic regression method for constructing physically plausible models using minimal training data and prior system knowledge was introduced



This work is licensed under a Creative Commons Attribution 4.0 International License.
GECCO '25, July 14–18, 2025, Malaga, Spain
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726434

in [23]. It employs an adaptive weighting scheme for balancing multiple loss terms and an epoch-wise learning strategy to minimize the risk of getting stuck in suboptimal local optima. Unlike the aforementioned NN-based approaches, this one incorporates prior knowledge about the model sought, making it more robust for practical deployment in real-world applications.

These NN-based approaches also have notable drawbacks: they are inefficient in reducing the initial redundant NN architecture to a final sparse form, and gradient-based learning is inherently prone to premature convergence to suboptimal topologies, making it difficult to transition to a better one.

This paper introduces a novel neuro-evolutionary SR algorithm, EN4SR, which integrates the advantages of the two main SR approaches mentioned above. This method combines an evolutionary search for optimal neural network topologies with gradient descent-based optimization of the network weights. In addition, it employs a technique we term *weight memory* and regular population perturbation to improve exploration and minimize the likelihood of getting stuck in suboptimal neural network topologies. The candidate NN models generated through this process adopt the architecture and the loss functions used in gradient-based learning inspired by those introduced for the N4SR method [23].

The main challenge is to efficiently sample and refine candidate NN topologies despite the increased computational demands that naturally arise from the nature of evolutionary algorithms. The contributions of the paper are:

- Design of a neuro-evolutionary SR method that combines advantages of the gradient-free evolutionary approach to generate candidate NN topologies and the gradient-based tuning of the NN weights.
- Design of a memory-based strategy to facilitate effective use of crossover and mutation operators regarding the weights of NN structures modified by these operators.
- The EN4SR method proved to be as good as or better than the pure NN-based approach in terms of model accuracy and computational demands while outperforming the other method in the frequency of produced high-quality solutions.
- The EN4SR method has been shown to significantly outperform the GP-based approach in terms of the quality of generated models.

The EN4SR algorithm was evaluated on four test problems and compared to other existing methods, among which is the NN-based N4SR serving as the baseline for comparisons with our neuro-evolutionary approach. In addition, an ablation study demonstrated the effect of some of the essential components of the algorithm.

The paper is organized as follows. Section 2 surveys the related work. Then, Section 3 outlines the particular SR problem considered in this work. The proposed method is described in Section 4 and Section 5 presents the experiments and the results obtained. Finally, Section 6 concludes the paper.

2 Related work

One of the first works on NN in symbolic regression is Equation Learner (EQL), a multilayer feedforward network with sigmoid, sine, cosine, identity, and multiplication units [26]. The network is trained using the Adam stochastic gradient descent algorithm [16] with a Lasso-like objective combining L_2 training loss and L_1 regularization. It uses a hybrid regularization strategy that starts with several update steps without regularization, followed by a regularization phase to enforce a sparse network structure to emerge. In the final phase, regularization is disabled, but the L_0 norm of the weights is still enforced, i.e., all weights close to 0 are set to 0. In [30], an extended version $\text{EQL}^{\div}$ was proposed, adding to the original EQL division units in the output layer. Extension of EQL with other deep learning architectures and $L_{0.5}$ regularization were also proposed [15].

OccamNet [8] is another multilayer NN architecture partially inspired by EQL. It represents a probability distribution over functions, using a temperature-controlled connectivity scheme with skip connections similar to those in DenseNet [13]. It uses a probabilistic interpretation of the softmax function by sampling sparse paths through a network to maximize sparsity. The Mathematical Operation Network (MathONet) proposed in [44] also uses an EQL-like NN architecture. The sparse subgraph of the NN is sought using a Bayesian learning approach that incorporates structural and nonstructural sparsity priors. The pruning algorithm for symbolic regression (PruneSymNet) proposed in [43] combines a greedy pruning algorithm with beam search to iteratively extract symbolic expressions from a differentiable neural network while maintaining its accuracy. This approach addresses the limitations of greedy algorithms in finding optimal solutions by generating multiple candidate expressions during pruning and selecting the one with the smallest loss.

A different class of NN-based SR approaches uses transformers such as GPT-2 [27]. They train the transformer model using a large amount of training data, where each sample is typically a tuple of the form (formula, data sampled from the formula). During inference, the transformer model constructs the formula based on a data set query. A discussion of the advantages and drawbacks of this approach is beyond the scope of this work. We refer interested readers to [3–5, 9, 36, 37].

Note that no prior knowledge is used in the approaches described above. When learning the model, these methods focus solely on minimizing the error computed on the training data. Consequently, they are likely to produce physically inconsistent results and do not generalize well to out-of-distribution samples. A recent trend in the literature addresses this shortcoming by increasing the focus on integrating traditional physics-based modeling approaches with machine learning techniques [39, 41]. An interesting approach in this direction is the physics-informed neural network framework that trains deep neural networks to solve supervised learning tasks with relatively small amounts of training data and the underlying physics described by partial differential equations [24, 28].

Combining SR with prior knowledge is still a relatively new research topic. In [6], counterexample-driven symbolic regression based on counterexample-driven genetic programming [19] was introduced. It represents domain knowledge as a set of constraints expressed as logical formulas. A Satisfiability Modulo Theories solver is used to verify whether a candidate model meets the constraints. A similar approach called Logic Guided Genetic Algorithms (LGGA) was proposed in [2]. Here, domain-specific knowledge is called auxiliary truths (AT), which are mathematical facts known a priori about the unknown function sought. ATs are represented

as mathematical formulas. Candidate models are evaluated using a weighted sum of the classical training mean squared error and the truth error, which measures how much the model violates given ATs. Unlike CDSR, LGGA performs computationally light consistency checks via AT formula evaluations on the data set.

In [21, 22], a multi-objective SR method was proposed, which optimizes models with respect to training accuracy and the level of compliance with prior knowledge. Prior knowledge is defined as nonlinear inequality or equality constraints the system must obey. Synthetic constraint samples (i.e., samples not measured on the system) are generated for each constraint. Then, the constraint violation error measures how much the model violates the desired inequality or equality relations over the constraint samples.

The informed EQL (iEQL) proposed in [40] uses expert knowledge in the form of permitted or prohibited equation components and a domain-dependent structured sparsity prior. The authors demonstrated in experiments that iEQL learned interpretable and accurate models. Shape-constrained symbolic regression was introduced in [12, 20, 25]. It allows for the inclusion of vague prior knowledge by measuring properties of the model's shape, such as monotonicity and convexity, using interval arithmetic. It has been shown that introducing shape constraints helps to find more realistic models.

3 Problem definition

Consider a regression problem where a nonlinear model of an unknown function $\phi : \mathbb{R}^n \to \mathbb{R}$ is sought, with n being the number of input variables. We seek a maximally sparse neural network model that minimizes the error observed on the validation data set and maximizes its compliance with the constraints imposed on the model, i.e., the desired properties of the model. The constraint satisfaction measure is calculated on a set of samples explicitly generated for each type of constraint, as proposed in [21, 22]. The sparsity of the model is quantified by the number of active links and units in the neural network (see the definitions in Section 4.1).

Training, validation, and constraint data sets – D_t, D_v, D_c – are used when learning and evaluating the generated models. The training data set contains samples of the form $\mathbf{d}_i = (\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is sampled from the training domain $\mathbb{D}_t$. The validation data set contains samples of the same form as D_t sampled from $\mathbb{D}_t$ such that $D_v \cap D_t = \emptyset$. The constraint data adopt the constraint representation and evaluation scheme as proposed in [22]. The set D_c contains synthetic constraint samples generated for each constraint on which the constraint violation errors will be calculated.

4 Method

The proposed algorithm generates candidate NN topologies using genetic operators and fine-tunes them via gradient-based backpropagation. Due to the inherent computational demands of evolutionary algorithms, it is not feasible to learn the parameters of each generated NN topology to full convergence. Instead, each NN topology is trained using only a short sequence of backpropagation iterations, while promising NN topologies are further refined in each generation.

Below, we describe the fundamental concepts of this method: the *master topology*, a template from which candidate *subtopologies* are derived using crossover and mutation operators; a strategy to store and reuse the weights of the best-performing subtopologies found so far to accelerate convergence to sparse subtopologies; the genetic operators of crossover and mutation. Finally, we present a pseudocode of the method. The source code is publicly available at https://github.com/jirkakubalik/EN4SR.

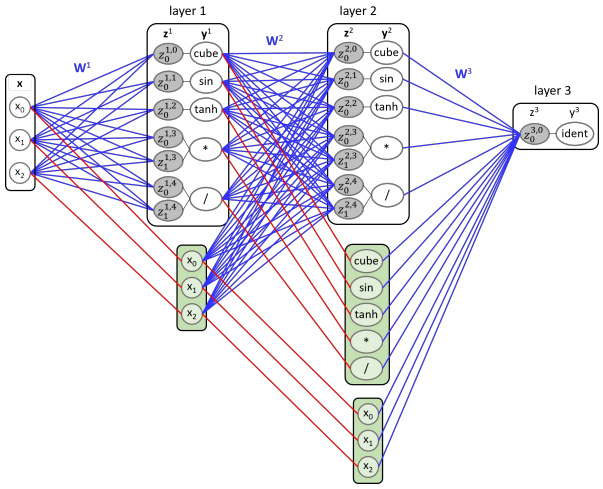


Figure 1: Master topology with two hidden layers as proposed for N4SR. The blue lines mark links with learnable weights. The red lines represent skip connections leading from the source units of layer $k-1$ to the copy units in layer k. For simplicity, this scheme does not show the bias links leading to every z-node.

4.1 Master Topology

The master topology represents the largest available network topology that can be used to encode the analytic expressions sought. It serves as a structural template for deriving subtopologies. We adopt the multilayer feedforward NN architecture designed for N4SR in [23], see Figure 1. It is a heterogeneous NN with units implementing mathematical operators and elementary functions, such as {+, −, *, /, sin, exp, etc.}.

Each hidden layer contains *learnable units* and *copy units* (the term introduced in [40]). The learnable units are units whose input weights, denoted as *learnable weights*, can be tuned by gradient-based learning (the links are shown in blue). All learnable weights of the master topology are collected in the set W_l. The copy units in layer k are just copying outputs of all units from the previous layer $k-1$, unit by unit. Each copy unit in layer k has just a single input link, the *skip connection* (the links shown in red), leading to its source unit in layer $k-1$. The skip connection values are set to 0 or 1 and are not subject to gradient-based learning. With the skip connections, simple structures in shallow layers can be efficiently learned and used in subsequent layers. All skip connection weights are collected in W_s. The set of all master topology weights is $\mathbf{W}$.

Units represent elementary unary functions, e.g., sin, cube, tanh, and binary functions (operators) such as multiplication * and division /. The internal structure of the unit consists of an *activation*

function (denoted as 'y' in Figure 1) and one or two *z-nodes*, depending on the activation function's arity, serving as operands to the activation function. The z-nodes calculate an affine transformation of the output of its previous layer. Each learnable unit i in layer j calculates its output as

$$y^{j,i} = g(z_0^{j,i}), \text{ for unary elementary function } g \tag{1}$$

and

$$y^{j,i} = h(z_0^{j,i}, z_1^{j,i}), \text{ for binary elementary function } h \tag{2}$$

where $z_{_}^{j,i}$ is calculated using the previous layer's output $\mathbf{y}^{j-1}$ as

$$z_{_}^{j,i} = \mathbf{W}_l^{j,i}\,\mathbf{y}^{j-1} + b^{j,i} \tag{3}$$

with learnable weights $\mathbf{W}_l^{j,i}$ and bias $b^{j,i}$. Likewise, in N4SR, singularity units are allowed at any network layer.

In the sequel, we use the following terminology:

- *Active link* – a unit's input link[1] whose weight $w \in \mathbf{W}$ has a positive absolute value. Links with a weight value equal to zero are considered *inactive*.
- *Active unit* – a unit for which it holds that it has at least one active link and it is connected to the network output via a sequence of other active units. Thus, active units contribute to the network output. Units that do not contribute to the network output are considered *inactive*.
- *Enabled link* – a unit's input link whose learnable weight $w \in W_l$ is updated through gradient-based learning. It can be both active and inactive.
- *Disabled link* – a unit's input link whose weight is set to zero and is not considered during gradient-based learning.
- θ^a – a user-defined threshold that specifies the minimum absolute value a weight must have to be considered significant, see Section 4.5.
- *Enabled unit* – a unit with at least one enabled link.
- *Disabled unit* – a unit whose all learnable weights are set to zero and cannot be updated through gradient-based learning.

4.2 Subtopology

A subtopology is a NN derived from the master topology. Initially, it has all the units defined in the master topology at its disposal. When initializing a new subtopology, all units are active, all learnable units and links are enabled, and the learnable link values are initialized randomly. All skip connection values are set to 1. Throughout the evolutionary process, only a subset of the units and their input links remain active. Any learnable link can be switched from an enabled to a disabled state if its absolute weight value drops below the threshold θ^a at a certain point of the computation, see Section 4.5. A unit can change its state from enabled to disabled and vice versa, either directly by the mutation operator, see Section 4.4, or indirectly when all its input links become disabled.

The role of the multi-objective evolutionary algorithm described below is to create novel subtopologies by combining and modifying the current ones. The standard stochastic gradient descent (SGD) algorithm, Adam [16], is used to train the weights of the generated subtopologies, using the following three loss functions adopted from [23]. Here, we only briefly characterize the individual loss functions and their components, for more details see [23].

$$\begin{aligned} \mathcal{L}_1 &= \mathcal{L}^t + \mathcal{L}^s \\ \mathcal{L}_2 &= \mathcal{L}^t + \mathcal{L}^s + \mathcal{L}^c \\ \mathcal{L}_3 &= \mathcal{L}^t + \mathcal{L}^s + \mathcal{L}^c + \mathcal{L}^r \end{aligned} \tag{4}$$

The loss functions are composed of the following terms:

- Training loss, $\mathcal{L}^t$ – the root-mean-square error (RMSE) calculated on the training data set D_t.
- Singularity loss, $\mathcal{L}^s$ – the root-mean-square *singularity error* calculated on all available data samples, $D = D_t \cup D_v \cup D_c$, for all active singularity units. The singularity error determines to what extent the singularity units correctly deal with the singularity values of their operands.
- Constraint loss, $\mathcal{L}^c$ – this term accumulates the error of the model in terms of violating the prior knowledge. Typical constraints are the monotonicity of the function with respect to a given variable over a specific interval of its values, the symmetry of the model with respect to its arguments, etc.
- Regularization loss, $\mathcal{L}^r$ – this term drives the learning process toward a sparse neural network representing a concise analytic expression. Here, we adopt the smoothed $L_{0.5}^*$ regularization, proposed in [15], applied to the set of active links of the subtopology.

Different loss functions and different numbers of learning steps are used in various stages of the algorithm, as shown in Algorithm 2.

4.3 Weight memory

An important component of this method is a strategy for managing and reusing the information acquired during the evolutionary process. This information is stored in *memory* and reused by the crossover and mutation operators; see Section 4.4.

4.3.1 Memory of z-node weights. Knowledge is stored in the *memory* as the weights observed in the recent best-performing subtopologies. Memory is a structure containing a list of up to s_{hist} records for every unit of the master topology, where s_{hist} is a user-defined parameter. Each record is a tuple of z-node weights of the given learnable unit, i.e., a singleton with a single vector of weights and a pair of two weight vectors for unary and binary function units, respectively. For copy units, the record is a scalar of either 0 or 1. Note that the memory may contain an empty list of records for some unit if the unit has not been active in any subtopology used for memory updates.

4.3.2 Memory management. The memory is updated in each generation of the evolutionary algorithm. For the update, the well-performing and parsimonious subtopologies are selected from the current population of subtopologies using the following procedure:

(1) Extract the set A of non-dominated subtopologies w.r.t. the validation RMSE, the singularity, and constraint losses. The regularization loss is omitted.
(2) Given the subtopologies in A, choose a set B of solutions that have all its performance values below the median value of the subtopologies stored in A.

[1] By the input link of a unit, we always mean the input links to the z-node(s) of that unit. An exception is the copy unit, which does not have a z-node and instead has a single input edge pointing directly to the corresponding unit in the previous layer.

(3) Calculate the third quantile q_3 of the complexity of the solutions in B. Select a set C consisting of subtopologies whose complexity is less than or equal to q_3.

Steps 1) and 2) ensure the quality of selected solutions, while step 3) prevents the selection of overly complex solutions despite their good performance.

All active unit weights of the subtopologies in C are used to update the memory in two steps. First, all records currently present in the memory originating from a subtopology that is dominated by at least one subtopology in C are removed from the memory. Then, all subtopologies s in C are processed so that a new record for every active unit of the given subtopology s is added to the memory if and only if the subtopology s is not dominated by any subtopology currently represented in the memory by its records, and the size of the current list of records of the respective unit is less than s_{hist}.

4.4 Genetic Operators

This section describes the genetic operators designed to sample new subtopologies with the use of the memory of weights. When two subtopologies s_1 and s_2 are crossed over, a new subtopology is generated in a layer-by-layer and a unit-by-unit manner while the child's z-node weights of the unit $u_{j,i}$ at i^{th} position in j^{th} layer are determined using the following structured rule where p_l, p_i, and p_h are user-defined parameters

(1) If $rand() < p_l$, the subtopology s_1 is considered the lead; otherwise the s_2 is the lead. The lead subtopology determines how the weights will be generated for the given child unit.
(2) If $rand() < p_i$ and the unit $u_{j,i}$ is active in the lead subtopology then the weights of the lead's unit $u_{j,i}$ will be inherited. Otherwise, the weights will be taken from the memory with the probability p_h, or new weights will be randomly generated with the probability $1 - p_h$.

The following two types of mutation operators are used to alter subtopologies:

- Unit status mutation – this operator randomly selects one of the learnable units, which can be either enabled or disabled, and swaps its status. If a disabled unit turns to an enabled one, with probability p_h, its weights are taken randomly from the memory; otherwise, new weights are randomly generated. If an enabled unit turns to a disabled one, all its input weights are set to zero.
- Unit weights mutation – this operator randomly selects one of the active units and changes its z-node weights either to the weights taken from the memory (with probability p_h) or to the new randomly generated ones.

The role of the memory of weights is to accelerate gradient learning of the newly generated subtopologies and to speed up the overall convergence to the final sparse ones.

4.5 Pseudo-code

The pseudo-code of the entire method is shown in Algorithm 1. It starts by generating a population of random subtopologies, initializing the weight memory, and the best-so-far set of non-dominated solutions based on the initial population. Non-dominated sorting is used that considers three subtopologyperformance criteria – validation RMSE, constraint RMSE, and the number of active links. Weights of the subtopologies are learned for N_t steps using $\mathcal{L}_1$ loss function.

The computation then proceeds in two nested loops: the outer loop represents stages, and the inner loop iterates over generations within each stage. Each stage begins with a perturbation of all subtopologies in the current population (line 5). The perturbation means that all units are enabled, and their weights are assigned values the same way as in the mutation operator. This operator prevents irreversible stagnation in a suboptimal subtopology. After the subtopology is perturbed, its weights are trained for N_t steps using the $\mathcal{L}_1$ loss function. A set of best stage subtopologies, $stageBest$, is initialized using the perturbed population.

The body of the stage starts with generating new subtopologies using crossover and mutation operators (line 9). Newly created subtopologies undergo only a few gradient learning steps (N_n is typically an order of units) with $\mathcal{L}_1$ loss function. In this way, many candidate subtopologies can be generated. Out of the set of new subtopologies, the best ones are chosen for the $interPop$ using the non-dominated sorting. Then, all subtopologies in $interpop$ undergo gradient learning for a larger number of steps (N_t) using $\mathcal{L}_3$ loss function. Moreover, the non-dominated subtopologies of the current population are tuned for N_f steps using $\mathcal{L}_2$ loss function. This means that the longer a subtopology remains among the non-dominated solutions in the population, the more gradient learning iterations it receives (in short batches). Next, the fine-tuned non-dominated solutions plus the solutions from the $interpop$ are merged with the current population, resulting in the new pop for the next iteration of the stage. Finally, the set of the best stage subtopologies is updated using the new population, and the memory of weights is updated based on the $stageBest$. Once the stage finishes, the set of overall best subtopologies is updated using the $stageBest$ solutions.

In the end, the algorithm returns the best subtopology, which is the least complex one that has all its performance criteria values below the median of the $overallBest$.

5 Experiments

5.1 Algorithms compared

We compared EN4SR with the NN-based algorithms N4SR-ACYE and EQL$^{\div}$, and a multi-objective GP-based algorithm using a local search procedure to estimate the coefficients of the evolved linear models mSNGP-LS, proposed in [22]. Note that N4SR-ACYE is the best-performing variant of the N4SR algorithm [23]. We also test the EN4SR variant that does not use the weight memory, denoted as EN4SR-base. Only EN4SR, EN4SR-base, N4SR-ACYE, and mSNGP-LS can use prior knowledge about the model sought. The three algorithms compared were used with the same parameter setting as in [23].

5.2 Test problems

Three problems, resistors, magic, and magman adopted from [23] were used for proof-of-concept experiments. In addition, we applied the method to a real-world problem of dynamic system identification of a quadcopter. Below, we briefly summarize the problem

Algorithm 1: EN4SR

Input: Master ... master topology
POPSIZE ... population size
R, G_R ... number of stages, stage generations
N_n, N_t, N_f ... newborn, tuning, fine-tuning backprop iterations

Output: best subtopology evolved

```
pop.init(L1);
memory.init(pop); overallBest.init(pop)
r ← 0
while r ≤ R do
    pop.perturb(L1)
    stageBest.init(pop)
    g ← 0
    while g ≤ G do
        interPop ← pop.generateSubtopologies(Nn, L1)
        interPop.runSGD(Nt, L3)
        nondominated ← pop.getNonDominated()
        nondominated.runSGD(Nf, L2)
        pop.merge(interPop, nondominated)
        stageBest.update(pop)
        memory.update(stageBest)
        g ← g + 1
    overallBest.update(stageBest)
    r ← r + 1
return overallBest.selectBest()
```

specifics and data related to the first three problems; details can be found in [23].

Resistors. This problem was originally proposed in [6]. The task is to find a model of the equivalent resistance of two resistors in parallel, $r = r_1 r_2/(r_1 + r_2)$, given noisy data D_t (400 samples) and D_v (100 samples) sampled uniformly from $\mathbb{D}_t = [0.0001, 20]$ Ω plus additional constraint samples D_c representing prior knowledge: symmetry with respect to the arguments ($f(r_1, r_2) = f(r_2, r_1)$), diagonal property ($r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2}$), and upper-bound property ($f(r_1, r_2) \leq r_1, f(r_1, r_2) \leq r_2$). The $\mathbb{D}_t$ interval also serves as the interpolation interval $\mathbb{D}_i$ to test the interpolation performance of generated models. The extrapolation performance of the models is tested on data sampled from the extrapolation interval $\mathbb{D}_e = [20.0001, 40]$, see Figure 2(a).

Magic. This problem considers the tire-road interaction model, where the longitudinal force $F(\kappa)$ is the following function of the wheel slip κ (aka the 'magic' formula): $F(\kappa) = m\,g\,d\,\sin(c\,\arctan(b(1-e)\kappa + e\,\arctan(b\kappa)))$, where b, c, d and e are road surface-specific constants. Here, we consider the *reference model* used in [38] with $m = 407.75$ kg, $g = 9.81\,\mathrm{m}\cdot\mathrm{s}^{-1}$, and the slip force parameters $(b, c, d, e) = (55.56, 1.35, 0.4, 0.52)$. A data set of 110 samples was generated on $\mathbb{D}_t = [0, 1]$ using the reference model and divided into D_t and D_v in the ratio of 4:1. The data are intentionally sampled unevenly. The whole $\mathbb{D}_t$ interval is divided into $\mathbb{D}_i = [0, 0.02] \cup [0.2, 0.99]$ and $\mathbb{D}_e = [0.03, 0.1]$, with well-represented $\mathbb{D}_i$ and poorly-represented $\mathbb{D}_e$ in D_t, see Figure 2(b). The data deficiency is compensated by using prior knowledge of the three types: the function is zero for $\kappa = 0$, it has a positive second derivative in the right part of $\mathbb{D}_i$, and it is concave down in $\mathbb{D}_e$.

Magman. The magnetic manipulation system consists of an iron ball moving on a rail and an electromagnet placed at a fixed position under the rail. The goal is to find a nonlinear model of the magnetic force, $f(x)$, affecting the ball as a function of the horizontal distance, x, between the iron ball and the electromagnet, given a constant current i through the coil. The whole operating region of `magman` is the interval $\mathbb{D}_o = [-0.075, 0.075]$ m out of which only a small part $\mathbb{D}_t = \mathbb{D}_i = [-0.027, 0.027]$ m is covered by training and validation data sets $|D_t| = 400$ and $|D_v| = 201$. These data samples were measured on the real system. The extrapolation data are sampled from the domain $\mathbb{D}_e = \mathbb{D}_o \setminus \mathbb{D}_i$ using an empirical model $\tilde{f}(x) = -ic_1 x/(x^2 + c_2)^3$ proposed in [14], see Figure 2(c). Five types of prior knowledge were defined: the function is odd; it is monotonically increasing on the intervals $[-0.075, -0.008]$ m and $[0.008, 0.075]$ m and monotonically decreasing on the interval $[-0.008, 0.008]$ m; it passes through the origin, i.e., $f(0) = 0$; it quickly decays to zero as x tends to infinity.

5.3 Experiment set up

We used the following two variants of the master topology, `master-A` and `master-B`, both with three hidden layers and a single identity unit (ident) in the output layer. The ident unit calculates a weighted sum of its inputs to realize addition and subtraction. The first two hidden layers in `master-A` and `master-B` contain four elementary functions {sin, tanh, ident, ∗} and {sin, arctan, ident, ∗}, each with two copies. The third hidden layer contains, in addition to that, one division unit. Topology `master-A` was used for `resistors` and `magman`, while `master-B` was used for `magic` problem.

We set the parameters of `N4SR-ACYE`, `EQL`$^{\div}$, and `mSNGP-LS` the same as in [23]. Algorithm `EN4SR` was tested with the following parameter setting: $POPSIZE = 10$, $R = 3$, $G_R = 20$, $N_n = 10$, $N_t = 100$, $N_f = 50$.

Note that the maximum number of backprop iterations was set to 90000 for a fair comparison with `N4SR`. All tested methods used the same parameter setting on all test problems. No parameter tuning was carried out.

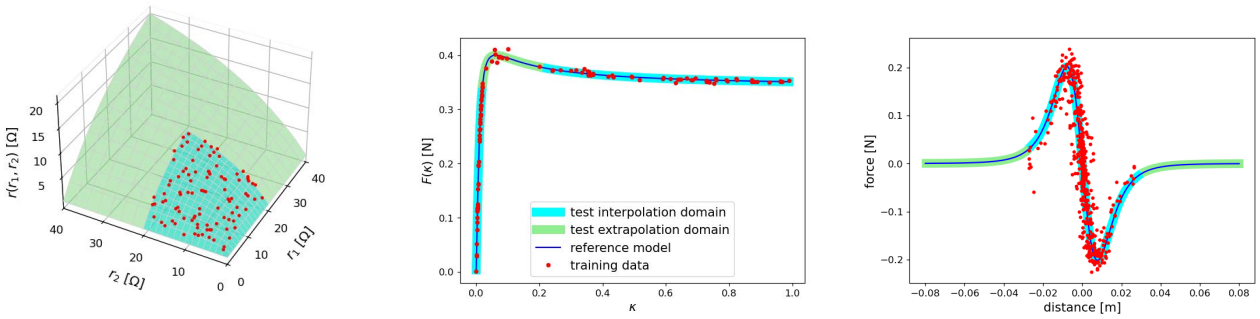
Thirty independent runs were performed with each method on every test problem. The best model is returned at the end of each run, yielding a set of thirty models on which we calculate the following performance measures:

- complexity – model complexity defined as the number of active units and active units,
- $RMSE_{int+ext}$ – RMSE calculated on test data sets sampling both the interpolation and extrapolation input domain of each test problem.

To assess the statistical significance of the observed differences in the $RMSE_{int+ext}$ performance of the compared algorithms, we use the Wilcoxon rank-sum test.

5.4 Results

Table 1 presents the results obtained with the considered methods. We can see that `EN4SR` outperforms the NN- based `N4SR-ACYE` on



(a) `resistors` (b) `magic` (c) `magman`

Figure 2: Reference models, training data D_t, interpolation domain $\mathbb{D}_i$, and extrapolation domain $\mathbb{D}_e$ of the three test problems.

all problems in terms of $RMSE_{int+ext}$. This shows that the evolutionary component introduced in EN4SR helps to achieve better, though slightly larger, models. EN4SR outperforms EQL$^{\div}$ on all problems that can be attributed to the fact that EQL$^{\div}$ does not use prior knowledge. EN4SR also achieves significantly better results than the GP-based mSNGP-LS on `resistors`, and performs equally well on the other two problems.

Table 1: Results of EQL$^{\div}$, mSNGP-LS, N4SR-ACYE, EN4SR-base, and EN4SR on the `resistors`, `magic`, and `magman` problem. The presented values are medians over 30 runs. The p-values are calculated on the sets of $RMSE_{int+ext}$ values.

Method	complexity	$RMSE_{int+ext}$	p-value
`resistors`			
EN4SR	5 / 15.5	$7.24 \cdot 10^{-3}$	–
EN4SR-base	5 / 16	$\mathbf{6.52 \cdot 10^{-3}}$	0.52
N4SR-ACYE	3 / 10	$4.7 \cdot 10^{-2}$	$1.02 \cdot 10^{-5}$
EQL$^{\div}$	18 / 30	6.01	$9.4 \cdot 10^{-14}$
mSNGP-LS	NA	$2.9 \cdot 10^{-2}$	$2.7 \cdot 10^{-4}$
`magic`			
EN4SR	6 / 15.5	$5.8 \cdot 10^{-3}$	–
EN4SR-base	7.5 / 19	$7.0 \cdot 10^{-3}$	0.92
N4SR-ACYE	4 / 8.5	$9.37 \cdot 10^{-2}$	$1.9 \cdot 10^{-8}$
EQL$^{\div}$	17 / 29	$5.26 \cdot 10^{-2}$	$1.22 \cdot 10^{-9}$
mSNGP-LS	NA	$\mathbf{2.01 \cdot 10^{-3}}$	0.21
`magman`			
EN4SR	6 / 10.5	$5.54 \cdot 10^{-2}$	–
EN4SR-base	4 / 5.5	$7.52 \cdot 10^{-2}$	0.015
N4SR-ACYE	5 / 9	$7.52 \cdot 10^{-2}$	$7.58 \cdot 10^{-3}$
EQL$^{\div}$	18 / 30	$5.86 \cdot 10^{-2}$	0.039
mSNGP-LS	NA	$\mathbf{5.18 \cdot 10^{-2}}$	0.17

Figures 3(a) and 3(b) illustrate different convergence properties of EN4SR and EN4SR-base on `magic` problem where the algorithms exhibit otherwise very similar performance (p-value = 0.92). The figures show convergence trajectories of models with 6 and 7 active units observed in 30 independent runs. For each run an evolution of the best $RMSE_{int+ext}$ value of a model with the given complexity is shown if such a model appears in the run (trajectories are plotted in light shades of red and blue color). Some of these trajectories are shown in saturated colors if a model of the given complexity with a $RMSE_{int+ext}$ value less than $5 \cdot 10^{-3}$ appeared in the run for the first time before generation 60. The generation number and the quality threshold were chosen to demonstrate the different behavior of the algorithms during the phase in which high-quality models begin to emerge. We can see, though it is hard to quantify, a difference in the number and distribution of these small and high-quality models.

5.5 Dynamic model of Quadcopter

In this section, we use real-world data from the Parrot Bebop 2 quadcopter to demonstrate the method's performance in a dynamic system identification task. The measured variables are the translational velocities v_x, v_y, and v_z in the fixed world frame (acquired in an indoor flying arena with the OptiTrack motion-capture system) and the body angles θ, φ and ψ, denoting pitch, roll, and yaw, respectively. The quadcopter was controlled by a model predictive controller (MPC) using the desired roll, pitch, yaw rate, and vertical velocity as control input, with the sampling period $T_s = 0.05$ s. The data set contains 498 samples acquired in an experiment with the quadcopter following an 8-shape trajectory in the horizontal plane (at a constant altitude). The data collected were divided into the training set (360 samples), the validation set (90 samples), and the test set (48 samples). The master topology had two hidden layers, each with one sin and cos unit, and two ident and multiplier units.

We use the proposed method to build a prediction model for the translational velocity v_x in the form:

$$v_x(k+1) = g(v_x(k), \theta(k)),$$

where $k = 0, 1, 2, \ldots$ denotes the discrete time instant and g is the unknown function sought by symbolic regression.

We generated 30 models, of which 24 turned out to be linear $v_x(k+1) = 0.985 v_x(k) + 0.473\theta(k)$, where the coefficients were rounded to three decimal digits. The remaining six models contained a sine or cosine nonlinearity. The 30 models performed

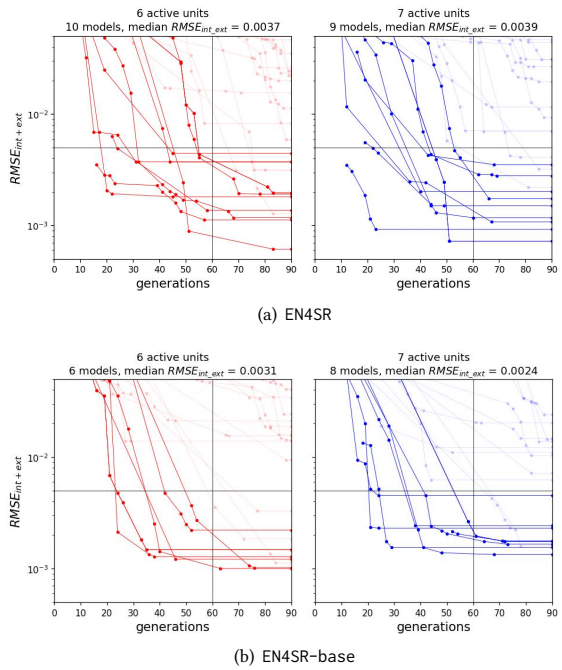


Figure 3: Convergence trajectories of models with 6 and 7 active units observed in 30 independent runs with EN4SR and EN4SR-base on magic problem.

almost equally well in a test simulation run on a fresh data set that was not used in the training. The test data set contains 699 samples acquired in an experiment with the quadcopter following a square trajectory in the horizontal plane. Figure 4 illustrates the performance of the above linear model. This recursive simulation of the model, $\hat{v}_x(k+1) = 0.985\hat{v}_x(k) + 0.473\theta(k)$, demonstrates the ability of the SR method to discover accurate predictive models of the quadcopter dynamics from data. The fact that the models do not contain an affine term (offset) represents the fact that the data were acquired indoors in the absence of wind. In an outdoor situation, the affine term would estimate the wind speed component in the respective direction.

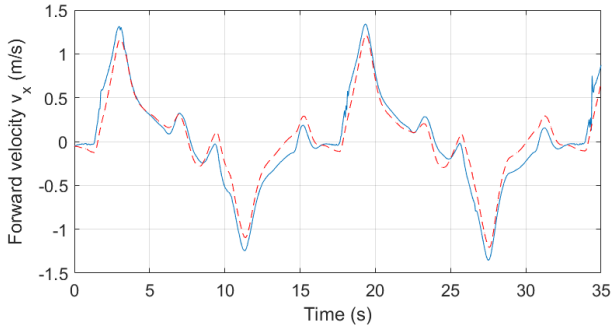


Figure 4: Quadcopter model simulation on unseen test data. Blue solid line: measured data, red dashed line: model.

6 Conclusions and future work

We proposed a new neuro-evolutionary symbolic regression method, EN4SR, which combines an evolutionary search for a suitable neural network topology with gradient-descent learning of the network's coefficients. We experimentally evaluated the proposed method on three tested problems. We also demonstrated the method's performance in a dynamic system identification task using real-world data. With a similar computing budget, this approach outperforms other NN-based approaches. It is also computationally more efficient and less prone to getting stuck in local optima. This clearly indicates the positive impact of the evolutionary component implemented in the method.

Note that the complexity of evolved neural network topologies measured by the number of active units and links relates only approximately to the complexity of the final analytic expression reduced to its minimal form. In our future research, we will address the problem of complexity reduction throughout the learning process via symbolic simplification. We will also analyze the effect of the memory-based strategy for maintaining and reusing useful information acquired during the evolutionary process and investigate new possibilities to realize this feature.

Since the proposed method is based on a multi-objective optimization approach, it inherently produces a set of non-dominated solutions. We have observed that the current final model selection strategy often overlooks models that perform notably better than the one ultimately selected. Therefore, we will also focus on designing a final model selection strategy that mitigates this undesired effect.

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] Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. 2015. Building Predictive Models via Feature Synthesis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 983–990. doi:10.1145/2739480.2754693

[2] Dhananjay Ashok, Joseph Scott, Sebastian Wetzel, Maysum Panju, and Vijay Ganesh. 2020. Logic Guided Genetic Algorithms. arXiv:2010.11328 [cs.NE]

[3] Tommaso Bendinelli, Luca Biggio, and Pierre-Alexandre Kamienny. 2023. Controllable Neural Symbolic Regression. arXiv:2304.10336 [cs.LG] https://arxiv.org/abs/2304.10336

[4] Amanda Bertschinger, James Bagrow, and Joshua Bongard. 2024. Evolving Form and Function: Dual-Objective Optimization in Neural Symbolic Regression Networks. 277–285. doi:10.1145/3638529.3654030

[5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurélien Lucchi, and Giambattista Parascandolo. 2021. Neural Symbolic Regression that Scales. *CoRR* abs/2106.06427 (2021). arXiv:2106.06427 https://arxiv.org/abs/2106.06427

[6] Iwo Bladek 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)*. ACM, New York, NY, USA, 977–984. doi:10.1145/3321707.3321743

[7] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. 2020. Operon C++: an efficient genetic programming framework for symbolic regression. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (2020). https://api.semanticscholar.org/CorpusID:220409306

[8] Allan Costa, Rumen Dangovski, Owen Dugan, Samuel Kim, Pawan Goyal, Marin Soljačić, and Joseph Jacobson. 2021. Fast Neural Models for Symbolic Regression at Scale. doi:10.48550/ARXIV.2007.10784

[9] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. 2022. Deep Symbolic Regression for Recurrent Sequences. *CoRR* abs/2201.04600 (2022). arXiv:2201.04600 https://arxiv.org/abs/2201.04600

[10] Erik Derner, Jirí Kubalík, Nicola Ancona, and Robert Babuska. 2020. Constructing parsimonious analytic models for dynamic systems via symbolic regression. *Appl. Soft Comput.* 94 (2020), 106432. doi:10.1016/j.asoc.2020.106432

[11] Erik Derner, Jirí Kubalík, and Robert Babuska. 2021. Selecting Informative Data Samples for Model Learning Through Symbolic Regression. *IEEE Access* 9 (2021), 14148–14158. doi:10.1109/ACCESS.2021.3052130

[12] C. Haider, F.O. de Franca, B. Burlacu, and G. Kronberger. 2023. Shape-constrained multi-objective genetic programming for symbolic regression. *Applied Soft Computing* 132 (2023), 109855. doi:10.1016/j.asoc.2022.109855

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269. doi:10.1109/CVPR.2017.243

[14] Zdenek Hurak and Jiri Zemanek. 2012. Feedback linearization approach to distributed feedback manipulation. In *American control conference*. Montreal, Canada, 991–996.

[15] Samuel Kim, Peter Y. Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Ceperic, and Marin Soljacic. 2021. Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery. *IEEE Transactions on Neural Networks and Learning Systems* 32, 9 (sep 2021), 4166–4177. doi:10.1109/tnnls.2020.3017010

[16] Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[17] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. 2020. Parameter Identification for Symbolic Regression Using Nonlinear Least Squares. *Genetic Programming and Evolvable Machines* 21, 3 (sep 2020), 471–501. doi:10.1007/s10710-019-09371-3

[18] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[19] Krzysztof Krawiec, Iwo Bladek, and Jerry Swan. 2017. Counterexample-driven Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Berlin, Germany) *(GECCO '17)*. ACM, New York, NY, USA, 953–960. doi:10.1145/3071178.3071224

[20] G. Kronberger, F. O. de Franca, B. Burlacu, C. Haider, and M. Kommenda. 2022. Shape-Constrained Symbolic Regression—Improving Extrapolation with Prior Knowledge. *Evolutionary Computation* 30, 1 (03 2022), 75–98. doi:10.1162/evco_a_00294

[21] Jirí Kubalík, Erik Derner, and Robert Babuska. 2021. Multi-objective symbolic regression for physics-aware dynamic modeling. *Expert Syst. Appl.* 182 (2021), 115210. doi:10.1016/j.eswa.2021.115210

[22] 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

[23] Jiří Kubalík, Erik Derner, and Robert Babuška. 2023. Toward Physically Plausible Data-Driven Models: A Novel Neural Network Approach to Symbolic Regression. *IEEE Access* 11 (2023), 61481–61501. doi:10.1109/ACCESS.2023.3287397

[24] Shilin Li, Gang Wang, Yuelan Di, Liping Wang, Haidou Wang, and Qingjun Zhou. 2023. A physics-informed neural network framework to predict 3D temperature field without labeled data in process of laser metal deposition. *Engineering Applications of Artificial Intelligence* 120 (2023), 105908. doi:10.1016/j.engappai.2023.105908

[25] Viktor Martinek, Julia Reuter, Ophelia Frotscher, Sanaz Mostaghim, Markus Richter, and Roland Herzog. 2024. Shape Constraints in Symbolic Regression using Penalized Least Squares. *CoRR* abs/2405.20800 (2024). doi:10.48550/ARXIV.2405.20800 arXiv:2405.20800

[26] Georg Martius and Christoph H. Lampert. 2016. Extrapolation and learning equations. *CoRR* abs/1610.02995 (2016). arXiv:1610.02995 http://arxiv.org/abs/1610.02995

[27] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.

[28] M. Raissi, P. Perdikaris, and G.E. Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707. doi:10.1016/j.jcp.2018.10.045

[29] Gustavo Richmond-Navarro, Williams R. Calderón-Muñoz, Richard LeBoeuf, and Pablo Castillo. 2017. A Magnus Wind Turbine Power Model Based on Direct Solutions Using the Blade Element Momentum Theory and Symbolic Regression. *IEEE Transactions on Sustainable Energy* 8, 1 (2017), 425–430. doi:10.1109/TSTE.2016.2604082

[30] Subham S. Sahoo, Christoph H. Lampert, and Georg Martius. 2018. Learning Equations for Extrapolation and Control. *CoRR* abs/1806.07259 (2018). arXiv:1806.07259 http://arxiv.org/abs/1806.07259

[31] M. Schmidt and H. Lipson. 2009. Distilling free-form natural laws from experimental data. *Science* 324, 5923 (2009), 81–85.

[32] Nicolas Staelens, Dirk Deschrijver, Ekaterina Vladislavleva, Brecht Vermeulen, Tom Dhaene, and Piet Demeester. 2013. Constructing a No-Reference H.264/AVC Bitstream-Based Video Quality Metric Using Genetic Programming-Based Symbolic Regression. *IEEE Trans. Cir. and Sys. for Video Technol.* 23, 8 (Aug. 2013), 1322–1333. doi:10.1109/TCSVT.2013.2243052

[33] Alexander Topchy, William F Punch, et al. 2001. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, Vol. 155162. Morgan Kaufmann San Francisco, CA.

[34] Leonardo Trujillo, Luis Munoz, Edgar Galvan-Lopez, and Sara Silva. 2016. neat Genetic Programming: Controlling Bloat Naturally. *Information Sciences* 333 (10 March 2016), 21–43. doi:doi:10.1016/j.ins.2015.11.010

[35] Silviu-Marian Udrescu and Max Tegmark. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6 (04 2020), eaay2631. doi:10.1126/sciadv.aay2631

[36] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. 2021. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. *CoRR* abs/2106.14131 (2021). arXiv:2106.14131 https://arxiv.org/abs/2106.14131

[37] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. 2022. SymFormer: End-to-end symbolic regression using transformer-based architecture. doi:10.48550/ARXIV.2205.15764

[38] Cees F. Verdier, Robert Babuška, Barys Shyrokau, and Manuel Mazo. 2019. Near Optimal Control With Reachability and Safety Guarantees. *IFAC-PapersOnLine* 52, 11 (2019), 230–235. doi:10.1016/j.ifacol.2019.09.146 5th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2019.

[39] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Michal Walczak, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. 2023. Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2023), 614–633. doi:10.1109/TKDE.2021.3079836

[40] Matthias Werner, Andrej Junginger, Philipp Hennig, and Georg Martius. 2021. Informed Equation Learning. *CoRR* abs/2105.06331 (2021). arXiv:2105.06331 https://arxiv.org/abs/2105.06331

[41] Jared D. Willard, Xiaowei Jia, Shaoming Xu, Michael S. Steinbach, and Vipin Kumar. 2020. Integrating Physics-Based Modeling with Machine Learning: A Survey. *ArXiv* abs/2003.04919 (2020).

[42] Casper Wilstrup and Jaan Kasak. 2021. Symbolic regression outperforms other models for small data sets. *ArXiv* abs/2103.15147 (2021).

[43] Min Wu, Weijun Li, Lina Yu, Wenqiang Li, Jingyi Liu, Yanjie Li, and Meilan Hao. 2024. PruneSymNet: A Symbolic Neural Network and Pruning Algorithm for Symbolic Regression. arXiv:2401.15103 [cs.LG] https://arxiv.org/abs/2401.15103

[44] Hongpeng Zhou and Wei Pan. 2022. Bayesian Learning to Discover Mathematical Operations in Governing Equations of Dynamic Systems. doi:10.48550/ARXIV.2206.00669

[45] Jan Žegklitz and Petr Pošík. 2019. Symbolic regression in dynamic scenarios with gradually changing targets. *Applied Soft Computing* 83 (2019), 105621. doi:10.1016/j.asoc.2019.105621