Evolutionary Approach for Composing a Thoroughly Optimized Ensemble of Regression Neural Networks

Abstract

The paper presents the GeNNsem (**Ge**netic algorithm A**NN**s en**sem**ble) software framework for the simultaneous optimization of individual neural networks and building their optimal ensemble. The proposed framework employs a genetic algorithm to search for suitable architectures and hyper-parameters of the individual neural networks to maximize the weighted sum of accuracy and diversity in their predictions. The optimal ensemble consists of networks with low errors but diverse predictions, resulting in a more generalized model. The scalability of the proposed framework is ensured by utilizing micro-services and Kubernetes batching orchestration. GeNNsem has been evaluated on two regression benchmark problems and compared with related machine learning techniques. The proposed approach exhibited supremacy over other ensemble approaches and individual neural networks in all common regression modeling metrics. Real-world use-case experiments in the domain of hydro-informatics have further demonstrated the main advantages of GeNNsem: requires the least training sessions for individual models when optimizing an ensemble; networks in an ensemble are generally simple due to the regularization provided by a trivial initial population and custom genetic operators; execution times are reduced by two orders of magnitude as a result of parallelization.

Keywords: ensemble modeling, regression, ANN, genetic algorithm, distributed computing

1. Introduction

Building a high-quality artificial neural network (ANN) is a demanding task. The accuracy of the model mainly depends on the network's architecture [1] but obtaining a model with good generalization performance also requires hyperparameters' fine tunings. Labor-intensive and time-consuming tasks of neural architecture search (NAS) and training-related hyperparameter optimization (HPO) can be automated as a part of the AutoML pipeline. Searching for the best combination of ANN's architecture and the training-related parameters for the specific model is a particularly challenging optimization problem. Such complex optimization problems can be successfully solved using the genetic algorithm (GA) [2] which is proven to be robust and capable of avoiding local minima. GA randomly searches the hyperparameter space and utilizes previous results to direct the search, which makes it superior to other non-manual methods: grid search and random search [3]. The quality of a model can be significantly improved by training multiple models and combining their predictions in the form of an model ensemble. Ensemble learning assumes that, by combining multiple base models, the error of a single base model will likely be compensated by other models, and, as a result, the overall prediction performance of the ensemble will be better than that of a single base model [4]. A neural network ensemble (NNE) is constructed by training multiple individual neural networks and combining their outputs into an ensemble output. Ever since Hansen and Salamon [5] showed that NNE outperforms the generalization ability of a single network, researchers have proposed numerous methods for creating ensembles of neural networks. When choosing networks to be incorporated into an ensemble, two criteria must be considered: the accuracy of the trained ANNs and the diversity among them. With diversity in predictions, networks make errors in different parts of the input space, allowing the ensemble to reduce the overall error rate, as there is a lower likelihood of an error by the majority decision rule [5]. The GA-based NAS procedure generates numerous architectures, offering the potential to combine these networks into a NNE. However, this approach does not ensure optimal ensemble construction, as networks are optimized independently. Additionally, training parameters significantly influence the performance of individual models, further affecting the overall quality of the ensemble. To build an optimal ensemble model, simultaneous optimization of NAS and HPO is essential, guided by the performance of the ensemble rather than individual model performance. To the best of our knowledge, a research gap exists in this field: no GA-based methods currently optimize NAS, HPO, and model selection of ANNs in NNEs simultaneously, with a focus on ensemble performance during the optimization process. This gap underscores the need for a fully automated framework that addresses these aspects, minimizing development costs and enhancing overall performance.

This paper presents a novel evolutionary approach for composing an optimal ensemble of ANNs for regression problems. GA is used to search for the optimal architectures and training-related hyperparameters of individual networks, as well as to select the best networks that will constitute an ensemble based on their weighted sum of training accuracy and diversity. The architectures are optimized for the number of layers, the number of neurons in each layer, and activation functions. Training-related hyperparameters' optimization includes finding the best dropout rate and network training algorithm. It should be emphasized that the search for the best architectures and hyperparameters of the individual networks is guided not only by their performances but also by the ensemble's performance.

A special distributed software framework called GeNNsem was developed to completely automate the process of ensemble composing. To build an optimal ensemble model within an acceptable time frame, GeNNsem parallelizes GA using the WoBinGO framework for distributed optimization [6, 7, 8]. To ensure the scalability of the proposed solution, WoBinGO utilizes a manager-worker parallelization model and a cloud-native microservice architecture with Kubernetes [9] orchestration engine.

The proposed approach was evaluated by solving three different regression problems: the synthetic benchmark problem proposed by Friedman [10], the Boston housing problem [11], and the real-world use-case from the field of hydro-informatics - modeling the displacement of a point inside the dam structure [12]. The experimental study was performed to compare the performance of the proposed solution with other ensemble approaches: averaging, bagging, boosting, Classic Star [13], Random Regression Machines (RRM) [14], TPOT [15], Ranking Prediction Strategy assisted Automatic Model Selection (RPS-AMS) [16], and AutoSL-GA [17] as well as with an individual ANN optimized using various techniques. The results show that GeNNsem requires the least training sessions for individual models when optimizing an ensemble. Networks in the GeNNsem-generated ensembles are generally simple due to the regularization provided by a trivial initial population and custom genetic operator. With GeNNsem, the execution time is reduced by two orders of magnitude as a result of parallelization.

The key contribution of this paper is the development of the GeNNsem framework for the automated construction of NNEs using a GA for regression tasks. The primary advantages include simultaneous optimization of the NAS and HPO of individual networks, considering both their performance and the overall performance of the ensemble. The novel chromosome encoding and custom GA operators enable efficient optimization of key network features by guiding the search from simpler to more complex architectures. Additionally, the distributed execution and scalability are ensured through a cloud-native micro-service architecture and Kubernetes orchestration.

The obtained results for Friedman and Boston housing datasets demonstrate the remarkable dominance of the proposed solution over all other approaches. When dealing with the real-world dataset, techniques rooted in decision trees proved to be more effective in capturing the intricacies of point displacement within the dam structure, compared to ensembles reliant solely on artificial neural networks (ANNs). However, even in this context, GeNNsem demonstrates comparably promising results within a notably more reasonable time frame.

The rest of the paper is organized as follows: In Chapter 2, the related work is introduced followed by the description of methodology in Chapter 3. Chapter 4 contains the explanation of the proposed microservice architecture for running optimization tasks in the cloud environment. The experimental results and discussion are given in Chapter 5 followed by the concluding remarks in Chapter 6.

2. Related work

Many research papers have focused on using evolutionary algorithms to optimize neural networks [18]. Various methods of encoding ANN hyperparameters in GA chromosome have been developed [12, 19, 20, 21], as well as GA operators for crossover and mutation of individuals [20, 21]. The GA-based NAS framework Genetic-GNN [22] can simultaneously optimize Graph Neural Network (GNN) structures and hyperparameters. The improved multi-objective evolutionary algorithm, MMEAPSL [23], uses a GNG [24] network to enhance its performance, making it highly effective for multi-objective neural architecture search (MoNASP) in convolutional neural networks (CNNs). The complex multi-objective NAS task can be simplified by transforming it into a classification problem, where a classifier is trained to predict the dominance relationships between candidate and reference architectures [25]. In the latest research, the Single-Domain Generalized Predictor (SDGP) [26] is trained on a single source search space while effectively performing in unseen target search spaces by utilizing a feature extractor to learn domain-invariant features and a neural predictor to map architectures to their accuracy in the target domain, supplemented by a multi-head attention-driven regularizer for improved generalization.

Much of the research has also been devoted to devising techniques for creating ensemble models that can be categorized into averaging, bagging, boosting, and stacking. By taking a set of models and simply averaging the outputs of all models, the ensemble can produce more accurate predictions than any specific model in the set [27, 28, 29]. Bagging [27] is a method for training models on different subsets taken from the base training set. The stacking [30, 28] method defines different weights for each model in the ensemble model. Boosting [31] can potentially produce a more diverse set of models in the ensemble than the bagging method. By determining the weights of the parts of the training set, the boosting forces new models to be trained on the parts of the training set that are harder to predict.

Random Machines (RM) and Regression Random Machines (RRM) methods do not belong to the group of traditional ensemble learning methods. These methods can be interpreted as a mixing of bagging and boosting focused on support vector models [32, 14]. Regularized Boosting (RBOOST) introduces a novel approach to stacking ensembles, utilizing boosting as its meta-learner and integrating a non-parametric stopping criterion specifically designed for HPO [33]. EvoBagging employs GA to select diverse set of bags for its base learners. This method evolves the contents of each bag across generations, prioritizing bags that enhance the performance of the base models created within them [34]. The forward selection method for adding models to an ensemble is fast and efficient but can lead to overfitting. Forward selection has worse performance than selection with replacement, Sorted Ensemble Initialization, and Bagged Ensemble Selection [35]. Increasing the size of the training set has a significant effect on improving the quality of the ensemble model. On the other hand, the increasing number of hidden layers in ANN base learners does not guarantee improvements in the ensemble model [36]. Considering the conflicting nature between the accuracy and diversity of base learners, researchers used the multi-objective optimization approach to evolve ANNs in an ensemble [37, 38], but base ANN learners had a fixed architecture. Classic Star [13] is the state-of-the-art algorithm for ANN ensembles with a fixed architecture. In the first step, weights of the constituent blocks (ANNs) are fine-tuned independently. Following this, in the second step, a new block is introduced, establishing connections among all blocks through a convex layer. This iterative process systematically explores all potential simplices, optimizing the weights of the added block, along with the convex weights. The NEAS [39] pipeline efficiently searches for lightweight ensemble models based on one-shot NAS, employing a novel diversity score to optimize the search space and a layer-sharing strategy to reduce model complexity. The authors also introduce a new search dimension, called the split point, to manage the trade-off between diversity and complexity constraints. Soares et al. [40] proposed GA and simulated annealing-based methods (GA-NNE and SA-NNE) to select a subset of ANN models that exhibit a high degree of diversity. This was achieved by employing various weight initialization methods, distinct training datasets, and models featuring different learning parameters and architectures. Sun et al. introduced the two-step BOP-Stacking [41] ensemble, incorporating K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Decision Tree (DT) algorithms as base learners, and Random Forest (RF) as the second-level learner. They used Bayesian optimization to enhance prediction accuracy by independently optimizing the hyperparameters of these models.

More recently, with a focus on classification problems, the authors of [42] proposed a stacked ensemble model that integrates multiple classic machine learning classifiers and deep learning algorithms, including SVM, LR, CNN, BiLSTM, and BiGRU. Based on the probability values generated by each model, an ANN meta-classifier determines the extent to which each model contributes to the final result. Ren et al. in [43] proposed a Multi-objective Iterative Model Selection (MoItMS) strategy to simultaneously maximize the ensemble model diversity and the accuracy of meta-learners resulting the optimized stacking-based ensemble created of three distinct types of learners KNN, SVM, and ANN, as its basic learners. The Addemup-GA algorithm [44] uses GA to optimize the choices of ANN models that make up an ensemble model. The usage of GA yields better results than bagging, Adaboosting, and the simulated annealing versions of Addemup. In contrast to our approach, the MoItMS uses Bayesian optimization for hyperparameter search adding limitations in ANN base learners: choice between only three fixed architectures, identical activation in each hidden layer, and not considering the dropout rate; while the Addemup does not search for the optimal training algorithms for ANNs constituting an ensemble. Authors of [45] present the results of combining networks produced by the NAS method called Multi-node Evolutionary Neural Networks for Deep Learning (MENNDL) [3, 46] which produces a variety of convolutional neural networks that perform well on the given dataset. The ensemble is created using the subset of best networks that were produced by MENNDL. The quality assessment of individual networks is performed independently of the ensemble quality assessment, and thereby the performance of the ensemble does not affect the search for the optimal architecture of the individual networks. Zhou et al [47] introduce the GASEN approach for creating a neural network ensemble by training several individual networks, assigning them random weights, and using GA to evolve those weights to produce the optimal ensemble. NAS and hyperparameters' search were not employed, and, in contrast to our solution, the architectures of the individual networks are not optimized with respect to the performance of the ensemble.

The Tree-based pipeline optimization tool (TPOT) [15] is an automated machine learning tool for optimizing machine learning pipelines. TPOT uses the GA for the hyperparameters optimization. By stacking and optimizing feature selection, pre-processing, and construction with diverse types of base learners it produces an optimized machine-learning pipeline. Mohand and Badra in [17] presented the optimized versions of SuperLearner (SL). They applied Bayesian optimization (AutoSL-BO) and genetic algorithms (AutoSL-GA) for hyperparameter optimizations of base learners, showing the superiority of AutoSL-GA over other methods. SL consists of six types of base learners: ANN, SVM, Elastic Net Regularization (ENR), Kernel Ridge Regression (KRR), LightGBM Regressor (LGB), and CatBoost Regressor (CBR), allowing significant diversity in base learners' predictions. In contrast to our approach, TPOT and AutoSL-GA utilize more distinct types of base learners, resulting in significantly larger hyperparameters search space. It is also fair to say that, at the present state, AutoSL-GA does not consider any parallelization strategy. Since TPOT and AutoSL-GA are both based on genetic algorithms, we benchmark GeNNsem against these methods. In our study, we also conducted a comparative analysis involving GeNNsem and the RPS-AMS. The RPS-AMS method improves the warm-start procedure of AutoML by creating an integration of evolutionary algorithms and a feature-based driven model selection strategy. Initially, the training data undergoes transformation into a meta-features dataset. Subsequently, the pretrained XGBoost regression model, utilizing these meta-features, is employed to estimate the potential performance of the twelve distinct types of candidate models in the surrogate model. Finally, the surrogate model is constructed by assigning specific weights to each of the top four models in a Sequential Least Squares Quadratic Programming (SLSQP) minimization process.

To summarize, none of the previous solutions for dealing with the evolutionary-based creation of NNEs proposes an ensemble performance-driven NAS and HPO implemented in a software framework that enables a fully automated distributed process of constructing an optimal NNE.

3. Evolving the ensemble model

In this section, we first introduce the chromosome encoding scheme, which represents each ANN within the ensemble. The structure of the chromosome is shown in Fig. 1. Following this, we provide a detailed description of each step of the algorithm.

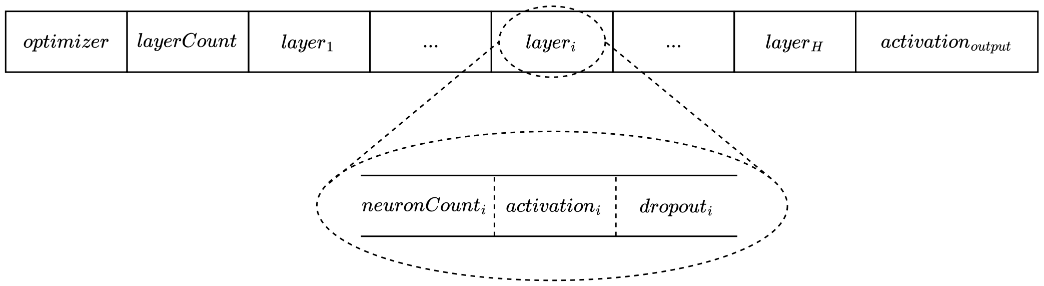

Fig. 1: The structure of the chromosome

The encoding scheme. Chromosome encoding in GAs is commonly performed using real, integer, or binary encoding. In GeNNsem, some genes are encoded with integer values, while others must be considered as a group consisting of both integer and real encodings. To solve this problem, we defined a complex chromosome with different types of genes, which led to the creation of specialized GA operators. The gene named *optimizer* represents an algorithm that will be used for training an ANN encoded in the chromosome. The *optimizer* takes a value from the set {*RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam*}. The number of hidden layers is encoded in *layerCount* gene, where the restriction $1 \leq layerCount \leq H$ applies. Each hidden layer i is encoded in a block of three genes: $neuronCount_i$, $activation_i$ and $dropout_i$ where $neuronCount_i$ represents the number of neurons, $activation_i$ is one of the activation functions from the set {*elu, selu, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, linear*} and $dropout_i$ is a dropout rate. Since all chromosomes must have the same fixed length, the genes for a prescribed maximum number of hidden layers are allocated. If the particular network has fewer hidden layers than the maximum allowed, only the first *layerCount* layers will be used, while the rest are ignored. The very last gene ($activation_{output}$) represents the activation function of the output layer. It uses the same activation function set as hidden layers. All genes are encoded as integers, except for the $dropout_i$ genes, which are encoded as real numbers.

The process of evolving an ensemble model using the proposed approach and GeNNsem framework is outlined in Fig. 2. GA is utilized to search for the best architectures and training-related hyperparameters of individual networks and the best networks to constitute an ensemble according to their training accuracy and diversity. The following subsections will provide a more detailed explanation of each enumerated step presented in Fig. 2.

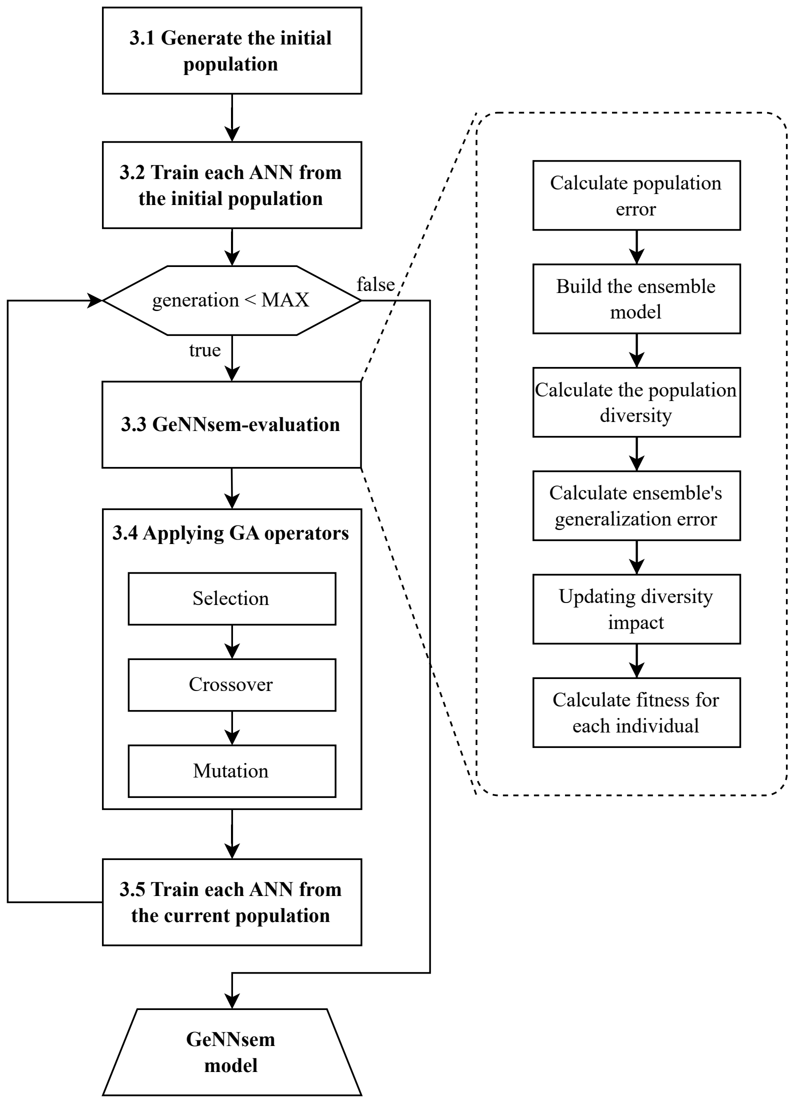
3.1 Generate the initial population
3.2 Train each ANN from the initial population
generation < MAX
false
true
3.3 GeNNsem-evaluation
3.4 Applying GA operators
Selection
Crossover
Mutation
3.5 Train each ANN from the current population
GeNNsem model
Calculate population error
Build the ensemble model
Calculate the population diversity
Calculate ensemble's generalization error
Updating diversity impact
Calculate fitness for each individual

Fig. 2: Algorithm for evolving an ensemble model

3.1. Generate the initial population

As Fig. 2 depicts, the process starts by generating the initial population of ANNs. Each ANN is encoded in a single chromosome with the structure shown in Fig. 1.

The number of hidden layers is set to 1 for all individuals in the initial population. This approach ensures that search favors simpler over complex networks [20, 21]. Simpler networks require less training time, which speeds up the algorithm convergence. The proposed chromosome structure is based on the NAS work of Vidnerová and Neruda [20], with the difference that they encode hidden layers to a variable-length chromosome, and do not use the *optimizer* gene since their algorithm searches only for the best neural architecture. The values of genes *optimizer*, $neuronCount_1$, $activation_1$, $dropout_1$, and $activation_{output}$ are chosen randomly for each ANN model within the defined gene ranges.

3.2. Training the initial population

After the initial population is generated, each ANN in the population is trained. Firstly, based on the data about the architecture, activation functions, and dropout rate extracted from the chromosome, an individual ANN model is built using the *Sequential* model from *Keras* library [48]. Model training is performed according to the algorithm indicated in the *optimizer* gene. The dataset is divided into a training and validation set. *The early Stopping* algorithm is applied to prevent a model from overfitting. The individuals that belong to the initial population use a Mean Squared Error (MSE) as a cost function.

3.3. GeNNsem evaluation phase

The next step in the optimization process described at Fig. 2, is the GeNNsem evaluation phase. It starts as soon as all the models in the population have been trained.

Calculating population error. The first step of the evaluation phase is the calculation of the population error $\bar{E}$ (equation (1)) which represents a weighted average of individual ANN errors:

$$\bar{E} = \sum_{i=1}^{k} w_i E_i, \tag{1}$$

where k is a number of networks in an ensemble, and E_i is the average of squared errors (equation (2)) over input data distribution of i-th model, $i = \overline{1..k}$. The squared error ϵ_i of i-th base model for input x can be obtained as:

$$\epsilon_i(x) \equiv [o_i(x) - f(x)]^2, \tag{2}$$

where $o_i(x)$ represents the i-th ANN model output for the input x, and $f(x)$ represents the expected output value for the input x. A weight assigned to each network based on its error on the validation set w_i is defined by the equation (3).

$$w_i = \frac{1 - E_i}{\sum_{j=1}^{k}(1 - E_j)} \tag{3}$$

Build the ensemble model. Next, an ensemble model is built as depicted in Fig. 3. The ensemble input is conveyed to each ANN in the ensemble and the outputs of the individual ANNs are combined to form the unique ensemble output. Combining the individual ANN model outputs into a single output is performed by summing the weighted outputs of ANN models. In equation (4) we define $\hat{o}(x)$ as the ensemble output model for the input x:

$$\hat{o}(x) = \sum_{i}^{k} w_i o_i(x), \tag{4}$$

where $o_i(x)$ represents the output of the i-th ANN model for the input x.
Calculating population diversity. Optimization of the ensemble model implies optimization of the individual networks that compose the ensemble. The individual networks should be as precise as possible, but their predictions should significantly differ from each other to obtain a more general model [5, 43, 44]. The disagreement of individual ANN predictions is represented by their diversity which is calculated in the next step of GeNNsem evaluation phase. The diversity d_i of i-th model for input x is defined by the equation (5):

$$d_i(x) \equiv [o_i(x) - \hat{o}(x)]^2, \tag{5}$$

where $o_i(x)$ and $\hat{o}(x)$ represent the outputs i-th model and the ensemble model for the input x respectively.
Calculating ensemble's generalization error. After calculating the diversity of the base models, GeNNsem proceeds to calculate the ensemble generalization error. The ensemble generalization error $\hat{E}$ can be expressed

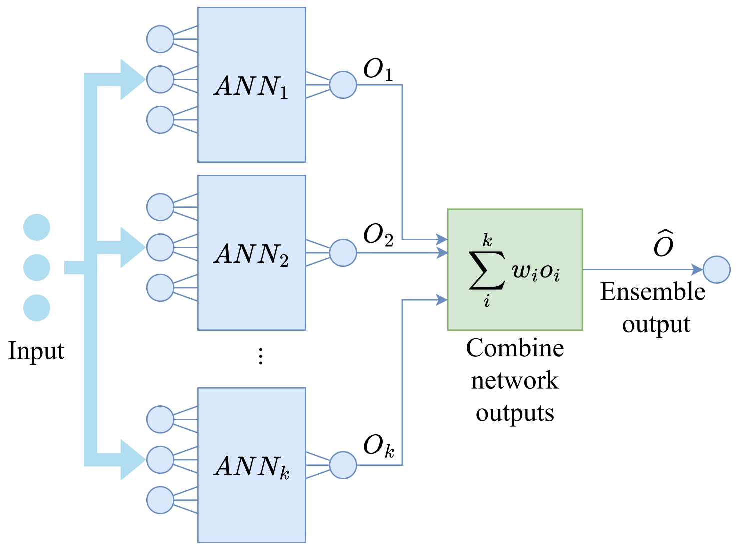

Fig. 3: GeNNsem ensemble model scheme

by the average errors of the individual models and the diversity as in equation (6):

$$\hat{E} = \bar{E} - \bar{D}. \tag{6}$$

Here, $\bar{E} = \sum_i w_i E_i$ represents a weighted average of individual ANN errors, and $\bar{D} = \sum_i w_i D_i$ represents a weighted average of the diversity among ANN models, where E_i and D_i are the average values of $\epsilon_i(x)$ and $d_i(x)$ over input data distribution, respectively. The equation (6) implies that an optimal ensemble model consists of networks with low errors, but that their predictions are as diverse as possible.

Updating diversity impact and calculating fitness. Fitness (equation (7)) is defined as a combination of model accuracy and diversity over other networks:

$$\begin{aligned} Fitness_i &= Accuracy_i + \lambda Diversity_i \\ &= (1 - E_i) + \lambda D_i. \end{aligned} \tag{7}$$

The variable parameter λ defines the diversity impact on the fitness of an individual network. To ensure that E_i and D_i have a proportional impact on fitness, their values are normalized to a range $[0-1]$. Since there is no strict rule for choosing λ, we apply heuristics that automatically adjust λ value during the optimization process in the "updating diversity impact" step of the evaluation phase. When the ensemble error of the model E_i decreases, the value of λ remains unchanged. In contrast, if the population error $\bar{E}$ does not increase and the diversity $\bar{D}$ decreases, the impact of diversity is insufficient, and we increase λ; if $\bar{E}$ increases and $\bar{D}$ stays approximately constant, the impact of diversity is too high, and we decrease λ. The variation of λ remains within 10% of its current value. The initial value in all our numerical experiments was set to 0.1.

3.4. Applying GA operators

Following the determination of fitness values for all individuals in the population, as outlined in Section 3.3, the subsequent generation is created by applying a sequence of GA operators. As illustrated in Fig. 2, we apply operators in the following order: selection, crossover, and mutation.

Selection. Binary tournament selection applies. Two random individuals are selected to participate in the tournament. The winner of the tournament is the individual with a higher fitness value [49].

Crossover. Crossover is a binary operator, that combines genes from parents and creates two children individuals as shown in Fig. 4. This operator applies only when the participating individuals have at least two hidden layers. The intersection happens at the point positioned between two adjacent hidden layers, dividing the chromosome into head and tail. The lengths of the parents' tails may vary, so it is also necessary to cross the $layerCount$ gene to determine the number of hidden layers.

Mutation. The mutation is a unary operator which randomly changes a chromosome. The mutation can be carried out in one of the following forms:

- *Optimizer mutation* - the current optimizer in a chromosome is replaced by a randomly selected optimizer
- *Hidden layer mutation* - the mutation is applied to a randomly chosen hidden layer. One of the following is performed:
  - The number of neurons in the hidden layer is replaced with a randomly selected number of neurons from a given range.
  - The activation function is replaced with a randomly selected function from an activation function set.
  - The dropout rate is set to a randomly selected value within predefined limits.

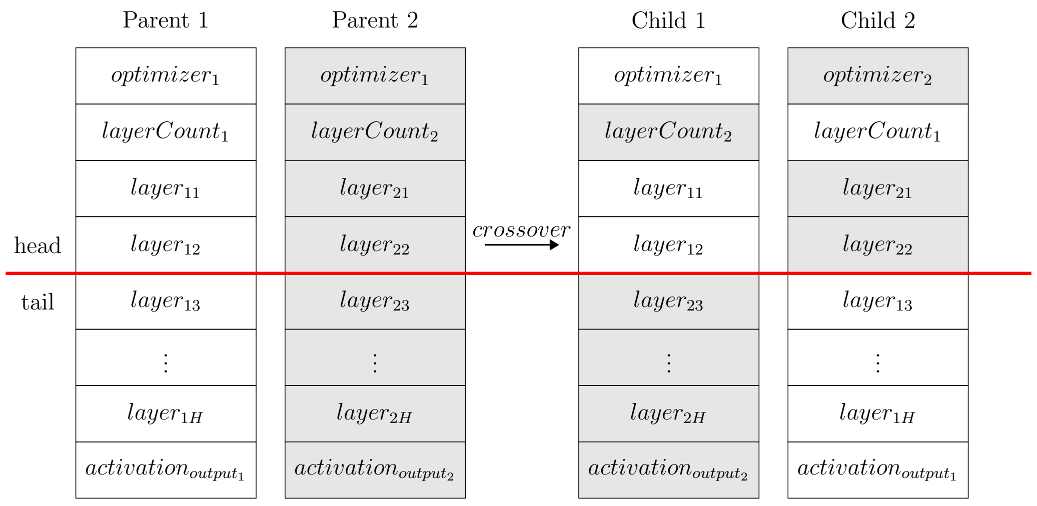

Fig. 4: Crossover operation in GeNNsem

- *Adding a hidden layer* - the number of hidden layers is increased and a new block with randomly generated values is inserted. This mutation can be applied only if the number of hidden layers is less than the maximum allowed number of hidden layers, H.

- *Removing a hidden layer* - a hidden layer is randomly selected and removed. There should be at least two hidden layers for this mutation to be applicable.

3.5. Training current population

The prediction of the ensemble model generated in one generation influences the training of ANNs in the next generation. The goal of an ANN within the ensemble model is to minimize errors across most samples, while allowing for larger errors only on those samples for which other models in the ensemble exhibit strong predictive performance. This is achieved by utilizing the cost function as in equation (8):

$$Cost = \sum_{x \in D} \left| \frac{f(x) - \hat{o}(x)}{\hat{E}} \right|^{\frac{\lambda}{\lambda+1}} [f(x) - o(x)]^2, \tag{8}$$

which introduces the additional expression that multiplies the standard error function. Here, $f(x)$ is the output value for input x from the dataset D,

$\hat{o}(x)$ is the prediction of the ensemble from the previous generation for input x, $\hat{E}$ is the ensemble generalization error, λ is the diversity impact, and $o(x)$ is ANN model output for input x. The values of $\hat{o}(x)$ and $\hat{E}$ do not depend on the model being trained, but on the ensemble from the previous generation. These values do not change during the model training. We normalize the difference $f(x) - \hat{o}(x)$ by the value $\hat{E}$ which keeps an average value of approximately 1. This is especially important when it comes to high-quality models in the population. The difference $f(x) - \hat{o}(x)$ will be close to 0 for most samples and the model will be trained on only a few samples. If λ is near zero, then the diversity is not significant, and the network is trained with the usual loss function. In contrast, when λ is large, the diversity is significant and there is a significant impact on the loss function.

After training all the ANNs in the current population, the algorithm proceeds with the GeNNsem evaluation phase and repeats the selection, crossover, mutation, training, and evaluation of individual networks and the comprising ensemble through generations, until it reaches the stopping criterion defined by the maximum number of generations.

4. GeNNsem framework architecture

The proposed optimization process is highly computationally intensive since it requires multiple training of all individual ANN models and the subsequent construction of the ensemble of those models.

To create an ensemble model in an acceptable time frame, we employed a modified WoBinGO framework [6, 7, 8] (Fig. 5). The job distribution component of WoBinGO remained unchanged, while the main optimization loop was completely altered by introducing a new component named *GeNNsem evaluation service* which builds an ensemble model from pre-trained neural networks and determines the fitness values of individual ANNs in each generation as was previously described.

JARE optimization service [6, 7, 8] executes the main loop of the genetic algorithm. In each generation, JARE submits parallel batch jobs to the Kubernetes cluster, one for each individual in the population. Each job is responsible for training one individual ANN model. Kubernetes cluster is made up of multiple nodes, and the Kubernetes scheduler finds the node with sufficient resources to assign a job to it. By specifying the appropriate resource requirements for individuals' training, it is possible to achieve optimal CPU and memory utilization of the entire cluster. With sufficient resources available, the training of individual ANNs can be fully parallelized and the training time of all ANN models in the generation becomes the training time of the most complex model in that generation. However, if the number of ANNs is greater than the number of available CPUs, certain training jobs stay in a queue for some time waiting for the resources. Just after an ANN model completes the training, the resources become available for the next job.

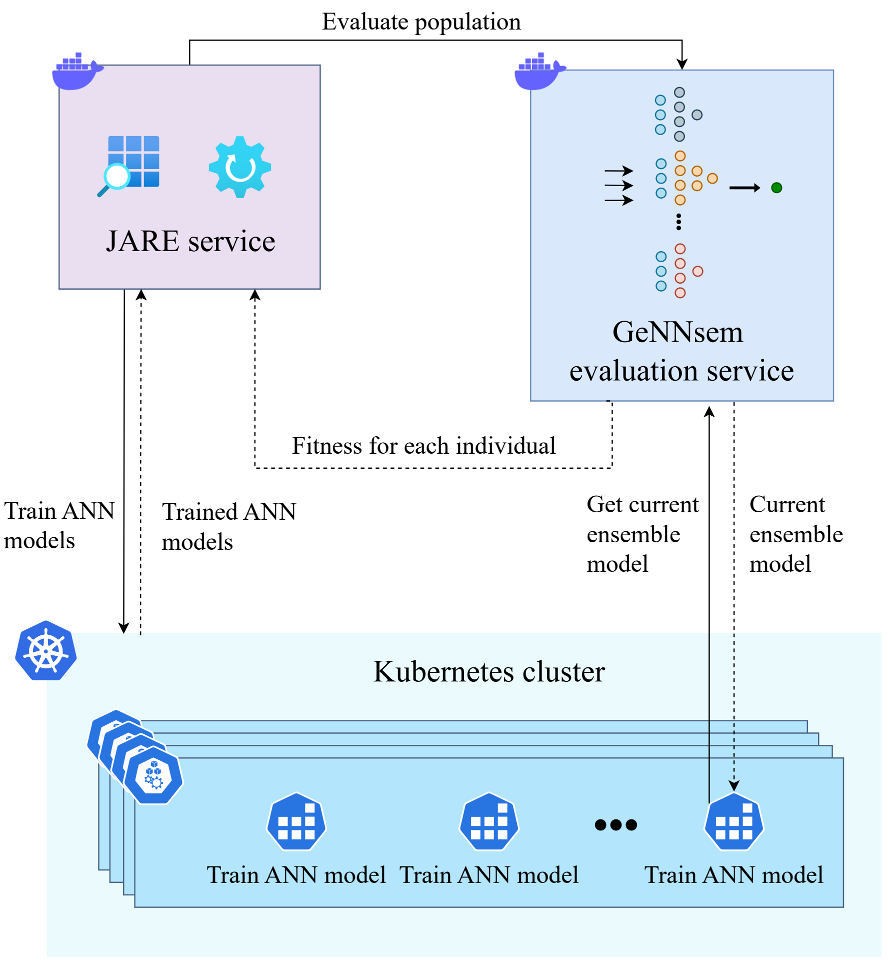

Fig. 5: GeNNsem optimization framework architecture

The number of Kubernetes nodes can also be scalable and should be determined by considering the number of individuals in the GA population. That way, by utilizing Kubernetes-based cloud offers, like Amazon EKS [50] with auto-scaling capabilities, the scaling process could be fully automated, resulting in even higher resource utilization.

After the batch job completes ANN training, JARE gathers the trained ANN model from the batch job output. JARE waits for all batch jobs in generation to finish, and then sends all ANN models to GeNNsem evaluation service. The GeNNsem evaluation service is a stateful Python API service whose main responsibility is to create a GeNNsem ensemble model from a group of pre-trained ANNs. The GeNNsem evaluation service loads all the ANNs received from JARE and calculates the population error (equation 1). The next step is building the GeNNsem ensemble model. Then, based on predictions of ANN models and the GeNNsem model, the service determines diversity among the ANNs and calculates the ensemble's generalization error (equation (6)). GeNNsem evaluation service monitors how the ensemble's generalization error changes through generations and updates the influence of diversity λ. The service determines the fitness value (equation (7)) for every individual in the population based on ANN models' accuracy and diversity.

ANN model training uses a custom training function that requires predictions of the GeNNsem model from a previous generation. At the beginning of each batch job, the GeNNsem model from the previous generation is embedded in the cost function.

The described processes repeat until the optimization reaches the maximum number of generations. The result is the optimized GeNNsem ensemble model.

5. Experimental study

This section presents the evaluation of the proposed approach for the automated evolution of neural network ensemble. A thorough investigation was conducted to assess the performance of the proposed solution. Firstly, we compared the proposed solution with individual ANNs optimized using various techniques. Secondly, we put it against various ensemble approaches, including averaging, bagging, boosting and Classic Star. Finally, we compared the GeNNsem performance with some of the most recent AutoML techniques. We evaluated the proposed solution on three regression problems: the *Friedman #1* synthetic benchmark problem, the Boston housing dataset, and the real-world use case from the field of hydro-informatics.

5.1. The Experimental Setup

We compared the GeNNsem ensemble against several models which can be organized into the following groups (Table 1):

I **Individual ANN models.**

II **Simple averaging ensemble of ANNs** takes average predictions from the set of individual ANN models. These models were obtained by taking a single network with optimized architecture and hyperparameters (using random search or GA) and training it 50 times with different initial weights.

III **Bagging ensemble of ANNs** applies the bootstrapping, training ANN models on diverse subsets of the training set, and averaging prediction results. All ANNs in the ensemble have the same, optimal architecture and hyperparameters obtained through the random search or GA.

IV **Boosting ensemble** is an iterative technique for combining weak learners into strong learner.

V **Classic Star ensemble** method involves training multiple ANNs with identical hyperparameters independently. Subsequently, an additional ANN is incorporated, connecting all outputs to a convex layer. This process optimizes the weights of the latest ANN and the convex weights simultaneously. We are using optimal architecture and hyperparameters obtained through the random search or GA. Additional benchmarks with the proposed architecture by the authors of [13] are provided in Appendix A.2.

VI **AutoML powered ensembles** group presents a compilation of contemporary ensemble approaches, encompassing a diverse array of methods beyond those confined solely to ANN ensembles.

Three different regression problems were used: the synthetic *Friedman #1* dataset, the Boston housing dataset, and the real-world problem of modeling the radial displacement of a point inside the dam structure.

Friedman #1 synthetic dataset contains five continuous features and one target variable determined by the equation (9):

$$y = 10\sin(\pi x_1 x_2) + 20\cdot(x_3-0.5)^2 + 10x_4 + 5x_5 + N(0,1) \quad x_i \sim U(0,1), \quad (9)$$

where $N(0,1)$ represents random values with the standard normal distribution and $U(0,1)$ represents random values with the uniform distribution over the interval $(0,1)$. The dataset consists of 1000 instances generated by the

Table 1: The list of models used for comparison with the GeNNsem model.

	Estimator type	**Method**	**Description**
I	Individual ANN (DNN)	a) Manual Search	Manually built ANN models are created by the authors, with the hyperparameters set based on their own experience.
		b) Random Search	Random-search optimized ANN uses a random-search algorithm for finding the right combination of model hyperparameters.
		c) GA optimized	GA optimized ANN models are created considering the methodology introduced in [20].
II	Simple averaging ensemble of ANNs	a) Ensemble of Random Search models	We repeated training of ANNs optimized through a random-search 1.b. for 50 times and cratered ensemble by averaging their predictions.
		b) Ensemble of GA optimized models	We conducted 50 training of ANNs optimized through GA 1.c. and formed an ensemble by averaging their predictions.
III	Bagging ensemble of ANNs	a) Ensemble of Random Search models	We trained multiple ANNs on diverse training subsets using hyperparameters from random-search optimized ANNs 1.b and averaged their predictions.
		b) Ensemble of GA optimized models	Ensemble model was created applying bagging strategy with a group of 50 GA optimized ANNs from 1.c.
IV	Boosting ensemble	a) XGB Regressor	Extreme Gradient Boosting (XGB) is a widely recognized boosting technique based on decision trees. It combines weak learners to obtain a more general model.
V	Classic Star ensemble	a) Ensemble of Random Search models	We applied the Classic Star [13] methodology to randomly-search-optimized ANNs from 1.b. We adopted Classic Star parameters listed in Table A.1 based on original authors' recommendations.
		b) Ensemble of GA optimized models	Classic Star ensemble [13] was created from ANNs optimized through GA in section 1.c. Again, we used the recommended parameters listed in Table A.1.
VI	AutoML powered ensembles	a) RRM	Regression Random Machines [14] use the benefits of both bagging and boosting techniques. The specific parameters for the proposed RRM can be found in Table B.1.
		b) TPOT	Tree-based pipeline optimization tool [15] utilizes GA to generate optimal ML pipelines. For detailed information on the parameters employed in TPOT, please refer to Table B.2.
		c) RPS-AMS	Ranking Prediction Strategy assisted Automatic Model Selection [16] is using a proposed ranking predictor to estimate the modeling accuracy by analyzing the meta-features before formal modeling.
		d) AutoSL-GA	A SuperLearner optimized by GA [17], AutoSL-GA employs the hyperparameter search space outlined in Table B.3 and it adopts the GA parameters as presented in Table B.4, consistent with the original authors' recommendations.

equation (9). Randomly chosen 80% of the data were used for training, while the remaining 20% were left for model testing.

The Boston housing dataset (available at StatLib Archive Repository [51]) shows the median house price in Boston's suburbs determined by the house condition, but also economic and social factors. The dataset comprises 506 instances, with 12 continuous features and 1 binary feature. The data were divided into two sets; the training set, which contained 404 observations, and the test set, which contained 102 observations.

The last problem was the real-world use case from the field of hydro-informatics. Managing water resources is a crucial task, especially in the context of dam safety. The case study relates to the Grancarevo dam at Trebisnjica River in Bosnia and Herzegovina. Our task was to predict the displacement of a point inside the dam structure, more specifically, to model the radial displacements of point P1 at the dam crest, block 17, as explained in [12]. There were four independent numeric features relevant to modeling radial displacement. The first was head-water (water level) H which is a measure of the hydro-static pressure inducing dam displacement. The thermal effect was considered using the average daily air temperature. The thermal effects have a strong influence on the deformation pattern during the year, so we considered variable d as the number of days elapsed from the beginning of the year. Because the dam has a phase offset regarding temperature changes, we introduced a dummy variable $d_{50} = d + 50$. Since the dam was built in 1967, aging has been an important factor. It shows the degradation of material properties during the years, and it is defined as the number of days elapsed from the dam construction divided by 1000. The data is a time series, obtained by the measurements of the dam sensors. We considered the period from January 1984 to the end of August 2011. We used only half of the available data, selecting every second row in that period. The final dataset consisted of 5042 instances.

We used four performance metrics for assessing the prediction error of the regression models. Mean Absolute Error (MAE) defined by the equation (10) evaluates the absolute average distance between the real values and the predicted values:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{10}$$

where n is the number of observations in the test set, the $\hat{y}_i$ is the predicted value for i-th instance of the test set and the y_i is the real output value for the corresponding input. Since this metric is insufficiently sensitive to outliers, we also used Root Mean Squared Error (RMSE) (equation (11)):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{11}$$

Additionally, we calculated Percent Root Mean Squared Error – PRMSE (equation (12)) as the relative RMSE value to the range of real values:

$$PRMSE = \frac{RMSE}{y_{max} - y_{min}} \cdot 100\%, \tag{12}$$

where y_{max} and y_{min} are the maximum and the minimum target value in the test set. The fourth metric we used is the coefficient of determination, R-squared (R^2). R^2 represents the proportion of the variance in the dependent variable explained by the independent variables in the model. The equation (13) defines R^2:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}, \tag{13}$$

where SS_{res} is the residual sum of squares defined as $SS_{res} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, and SS_{tot} is the total sum of squares, $SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y})^2$, and $\bar{y}$ is the mean of the observed data, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. An R^2 score of 1 indicates a perfect fit, while an R^2 score of 0 denotes that the model predicts the mean of the target variable for any given input. All metrics values in this paper were obtained from the test sets.

We used identical testing configurations for all our experiments. The ensemble models were built upon 50 individual neural networks. Datasets were split into training, validation, and test sets. First, we set aside 20% of a dataset for model testing, and then from the rest of the dataset we randomly selected 80% for the model training and 20% for model validation. The search ranges for NAS and HPO are listed in Table 2. The maximum number of hidden layers was 15, with a maximum of 20 neurons per layer. For each layer, an activation function used by all neurons in that layer was chosen from the following set: *SoftMax, Elu, Selu, SoftPlus, SoftSign, ReLU, TanH, Sigmoid, HardSigmoid*, or *Linear* activation function. For each individual network, the optimal training algorithm was searched among the *SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax*, and *Nadam* algorithm. The dropout rate varied from 0 to 0.25. The batch size was set to 32 based on the model

Table 2: The ranges of the decision variables.

Parameter	Range
Number of hidden layers	[1-15]
Number of neurons in a layer	[1-20]
Dropout rate	[0-0.25]
Training algorithm	SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam
Activation	SoftMax, Elu, Selu, SoftPlus, SoftSign, ReLU, TanH, Sigmoid, HardSigmoid Linear

performance, size of datasets, and computational costs. GA optimization was performed in 30 generations with 50 individuals in the population. The crossover probability was 0.9 and the mutation probability was set to 0.2.

To execute all experiments, we employed an on-premise Kubernetes 1.22 cluster consisting of 7 physical nodes. Each node is equipped with dual Intel Xeon E5-2683 v4 @ 2.1GHz CPU (32 physical cores), 128GB memory, and 10Gbps interconnection, totalling 224 cores and 896GB of RAM. The base OS platform is CentOS 7.7 x86_64.

5.2. Results and discussion

This section presents the results obtained from the previously described experiments. In the first part of the section, we discuss the quality of the GeNNsem ensembles in terms of errors exhibited through generations over the test sets. In the second part of this section, we compare the GeNNsem model performance on *Friedman #1* problem, Boston housing, and the Grancarevo dam problem with the performances of other models. A deeper analysis comparing the performance of the AutoSL-GA model with our proposed approach is presented in the third part of this section. In the final part of this section, we discuss the total time needed for the creation of the GeNNsem model and the advantage of parallel training of all ANN models in one generation.

5.2.1. GeNNsem performance across the generations

To visualize the results in Fig. 6 , we selected the three top-performing models on each problem, evaluated using the RMSE metric on the test set. For the GeNNsem model, performance was recorded after each generation, while for the other models, only their final scores are indicated with horizontal lines. Our model is consistently highlighted in red across all diagrams for ease of identification. It is important to note that this figure is not intended to show how the algorithm converges but rather to highlight how effective the optimization process is in improving model performance. This distinction is crucial because the data presented are from the test set, which was not used during the optimization process.

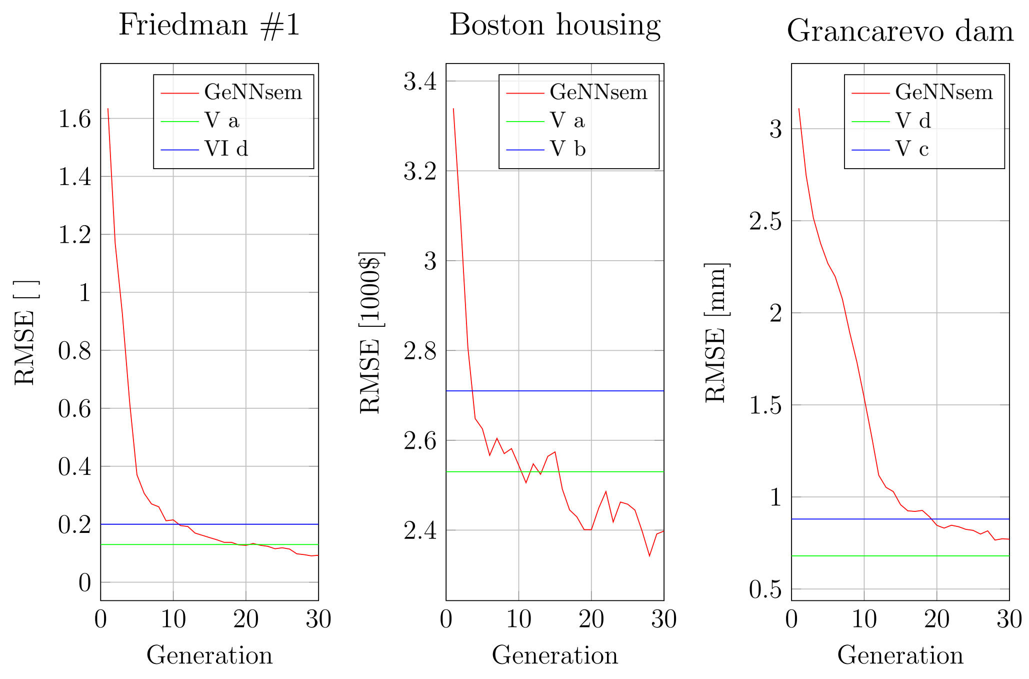

Fig. 6: Test set errors by GeNNsem generations for 3 benchmark problems. Horizontal lines represent errors of 2 other top-performing models.

As we can see from Fig. 6, in all our test problems, there was an apparent trend of lowering errors as the GA process progressed. Ensemble models created in the first few generations did not show sufficient generality on the test dataset. As the GA process was advancing, more precise and more general models were built. If we consider *Friedman #1*, we notice that the model from the 29th generation had the lowest error value on the test set. Despite that, we adopted the ensemble model from the last, 30th generation, because results must be independent of the test set performance. We applied the same rule to the other two benchmarks.

5.2.2. *Comparative performance of the GeNNsem model*

The performance comparison of fourteen models listed in Table 1 and the GeNNsem ensemble model are shown in Table 3 and visualized in Fig. 7.

It is evident that in most test cases the optimized ensemble model obtained by GeNNsem produced significantly lower errors than all other approaches. In *Friedman #1* benchmark, using the optimized ensemble model, we obtained an RMSE of just 0.09, which is a significant improvement over both individual and other ensemble models. Among the models evaluated on the Boston housing dataset, GeNNsem stood out as generalizing the best, exhibiting substantially lower error rates than other benchmark models. This result shows the importance of respecting both accuracy and diversity among the models. It can also be noticed that building ensembles based on a single ANN model with optimized architecture and hyperparameters (as in comparative models II a, II b, III a and III b) can produce ensembles with lower performance than the single model itself. This is a consequence of the non-reproducibility of model training, the small diversity between the models, and the equal weighting of all models within the ensemble. Extreme Gradient Boosting Regressor is made of diverse models where each model has a specific weight. This results in its better performance on real-world problems when compared to averaging and bagging. Classic Star defines different influences of base models in the ensemble while simultaneously fine-tuning the weights of the additional ANN model. As a result, it creates more general predictions than other ANN-based methods, but the GeNNsem method still outperforms them all.

The AutoSL-GA outperforms all other models when solving our real-world problem. However, GeNNsem remains the closest contender to the best-performing model achieving better results than all other AutoML methods. To gain deeper insights, we conducted further investigations into this matter.

5.2.3. *Grancarevo dam: In-depth analysis of AutoSL-GA and GeNNsem*

It shows that AutoSL-GA in the SLSQP minimization process completely rejects ANN, SVM, ENR, and KRR models from the ensemble by setting their weights to zero. Instead, it favored LGB, assigning it a significant weight of 0.983, while CBR had a minor influence with a weight of only 0.017. To assess the quality of discarded models, we created additional experiments by repeating the SLSQP minimization process while excluding one tree-based method at a time. Models with only one tree-based base learner also reject

Table 3: Comparison of models' performance on the test sets.

Friedman #1						
	Estimator type	Method	MAE	RMSE	PRMSE	R^2
I	Individual ANN (DNN)	a) Manual Search	2.02	2.88	12.42%	0.6912
		b) Random Search	0.18	0.24	1.05%	0.9978
		c) GA Optimized	0.20	0.31	1.32%	0.9965
II	Simple averaging ensemble of ANNs	a) Ensemble of Random Search models	2.05	2.85	12.27%	0.6987
		b) Ensemble of GA optimized models	0.57	0.97	4.20%	0.9647
III	Bagging ensemble of ANNs	a) Ensemble of Random Search models	2.04	2.84	12.25%	0.6994
		b) Ensemble of GA optimized models	0.73	1.21	5.23%	0.9453
IV	Boosting ensemble	a) XGB Regressor	1.06	1.36	5.84%	0.9317
V	Classic star	a) Ensemble of Random Search models	0.10	0.13	0.57%	0.9993
		b) Ensemble of GA optimized models	0.21	0.29	1.26%	0.9968
VI	AutoML powered ensembles	a) RRM	0.52	0.81	3.47%	0.9759
		b) TPOT	0.35	0.49	2.11%	0.9911
		c) RPS-AMS	0.66	0.90	3.89%	0.9697
		d) AutoSL-GA	0.14	0.20	0.84%	0.9986
VII	GA optimized ensemble of ANNs	**GeNNsem**	**0.07**	**0.09**	**0.40%**	**0.9997**
Boston Housing						
	Estimator type	Method	MAE	RMSE	PRMSE	R^2
I	Individual ANN (DNN)	a) Manual Search	2.31	3.31	7,57%	0.8892
		b) Random Search	2.18	2.88	6.60%	0.9159
		c) GA Optimized	2.10	2.74	6.27%	0.9241
II	Simple averaging ensemble of ANNs	a) Ensemble of Random Search models	2.38	3.49	7.99%	0.8765
		b) Ensemble of GA optimized models	2.58	3.91	8.94%	0.8454
III	Bagging ensemble of ANNs	a) Ensemble of Random Search models	2.49	3.68	8.41%	0.8633
		b) Ensemble of GA optimized models	2.47	3.81	8.71%	0.8534
IV	Boosting ensemble	a) XGB Regressor	2.27	3.02	6.92%	0.9075
V	Classic star	a) Ensemble of Random Search models	1.96	2.53	5.80%	0.9351
		b) Ensemble of GA optimized models	2.14	2.71	6.21%	0.9254
VI	AutoML powered ensembles	a) RRM	2.55	3.68	8.43%	0.8627
		b) TPOT	2.58	3.40	7.78%	0.8831
		c) RPS-AMS	2.13	2.74	6.27%	0.9240
		d) AutoSL-GA	2.37	3.21	7.34%	0.8960
VII	GA optimized ensemble of ANNs	**GeNNsem**	**1.84**	**2.40**	**5.49%**	**0.9418**
Grancarevo dam						
	Estimator type	Method	MAE	RMSE	PRMSE	R^2
I	Individual ANN (DNN)	a) Manual Search	1.79	2.80	5.63%	0.9455
		b) Random Search	1.61	2.61	5.24%	0.9527
		c) GA Optimized	1.59	2.53	5.08%	0.9556
II	Simple averaging ensemble of ANNs	a) Ensemble of Random Search models	1.70	2.65	5.32%	0.9512
		b) Ensemble of GA optimized models	1.57	2.54	5.10%	0.9551
III	Bagging ensemble of ANNs	a) Ensemble of Random Search models	1.58	2.55	5.12%	0.9548
		b) Ensemble of GA optimized models	1.56	2.54	5.10%	0.9552
IV	Boosting ensemble	a) XGB Regressor	0.75	1.01	2.02%	0.9929
V	Classic star	a) Ensemble of Random Search models	0.89	1.15	2.30%	0.9909
		b) Ensemble of GA optimized models	1.53	2.30	4.62%	0.9632
VI	AutoML powered ensembles	a) RRM	1.08	1.60	3.12%	0.8626
		b) TPOT	1.42	2.17	4.36%	0.9673
		c) RPS-AMS	0.65	0.88	1.77%	0.9946
		d) **AutoSL-GA**	**0.52**	**0.68**	**1.36%**	**0.9968**
VII	GA optimized ensemble of ANNs	GeNNsem	0.59	0.77	1.55%	0.9959

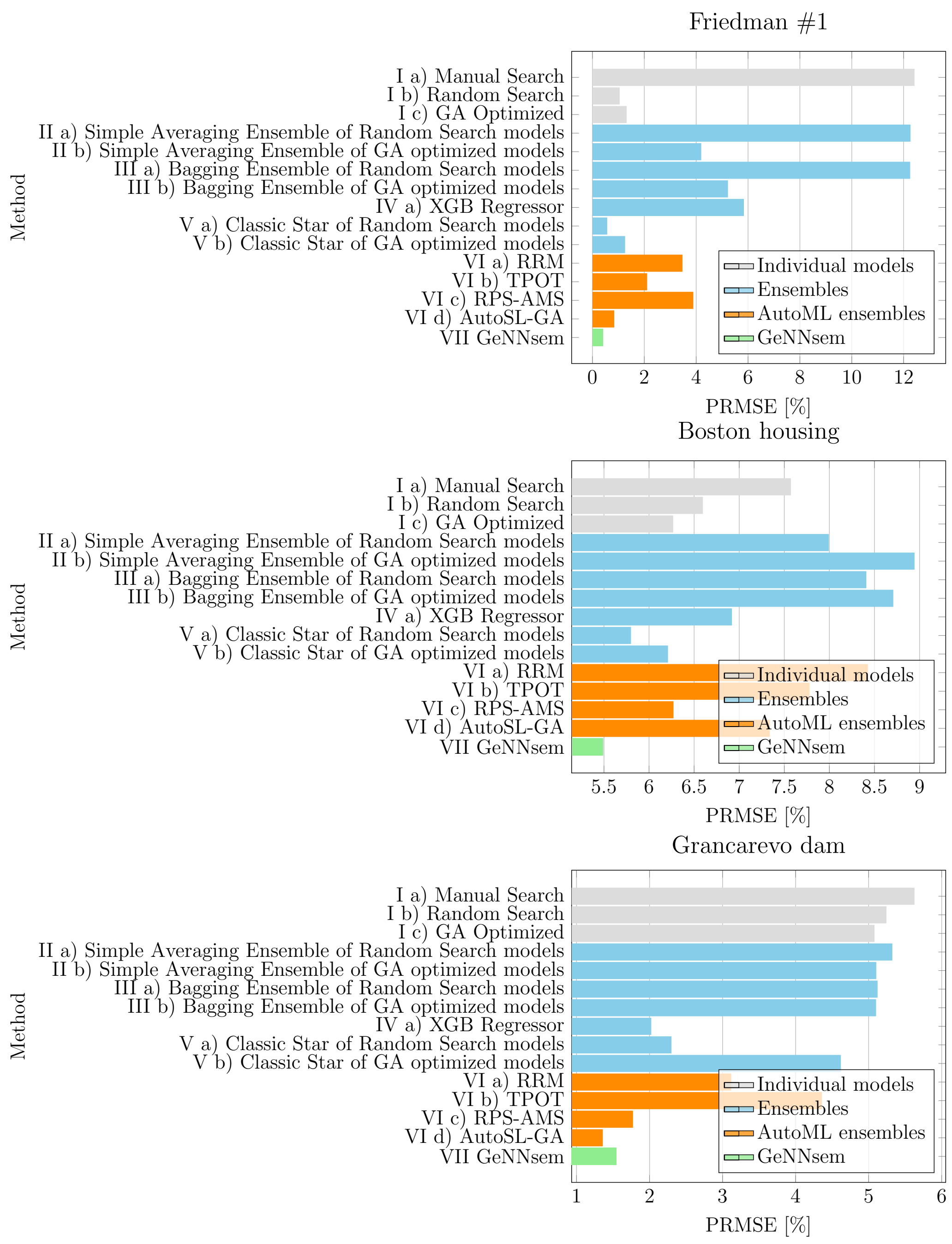

Fig. 7: Comparison of models' relative errors on benchmark test sets

Table 4: Additional experiments on the Grancarevo dam problem - Comparison of models' performance on the test set

Method	Base learners	MAE	RMSE	PRMSE	R^2
AutoSL-GA	ANN, SVM, ENR, KRR, LGB, CBR	**0.52**	**0.68**	**1.36%**	**0.9968**
	ANN, SVM, ENR, KRR, LGB	**0.52**	**0.68**	**1.36%**	**0.9968**
	ANN, SVM, ENR, KRR, CBR	0.57	0.74	1.48%	0.9962
	ANN, SVM, ENR, KRR	1.01	1.42	2.85%	0.9860
GeNNsem	ANN	0.59	0.77	1.55%	0.9959

other types of base learners by setting their weights to 0, while model performance remains stable. The resultant model, comprising only ANN, SVM, ENR, and KRR, exhibits a significant degradation in performance, increasing RMSE to 1.42. An in-depth comparison of models AutoSL-GA, AutoSL-GA without tree-based methods, and GeNNsem across all employed metrics on the Grancarevo dam problem is presented in Table 4. Based on these findings, we concluded that tree-based methods are more suitable for this particular problem, although it's worth noting that the GeNNsem model also shows good potential. As Fig. 6 shows, the GeNNsem optimization stopped by reaching the stopping criterion - 30 generations, despite an ongoing trend of error reduction.

5.2.4. Efficiency of GeNNsem parallelization

An additional point worth noting is that although AutoSL-GA does not include a parallelization strategy, it requires training a significantly larger number of models than GeNNsem. The time complexity of developing the AutoSL-GA model posed challenges, as we had to train 15000 models, including 2500 ANNs for each benchmark problem, and explore a significantly larger hyperparameters' search space than with GeNNsem. GeNNsem creates a comparable model by training 1500 ANNs for each benchmark problem. Additionally, GeNNsem's ANNs on average, less complex due to the regularization method (Section 3), suggesting clear efficiency advantages in terms of training time. To compare the execution speed, we ran the Friedman #1 problem with AutoSL-GA on a single CPU, setting a timeout equal to the total execution time of our algorithm, including the time used for building ensemble models. The results show that AutoSL-GA could evaluate only 107 ANN models within the given time, while the evaluation of the remaining model types had not even started.

In addition to more efficient single-CPU execution, with the proposed parallelization approach, GeNNsem achieved speeding up of the model training process just above 40 times per generation.

The in-depth analysis of the GeNNsem performance is shown in Fig. 8. The blue dashed line represents the total time required for evaluating all individuals in each generation, while the green dashed line shows the obtained speed-up in each generation. The distribution of training times for ANNs in each generation is represented using a box plot. Based on Fig. 8,

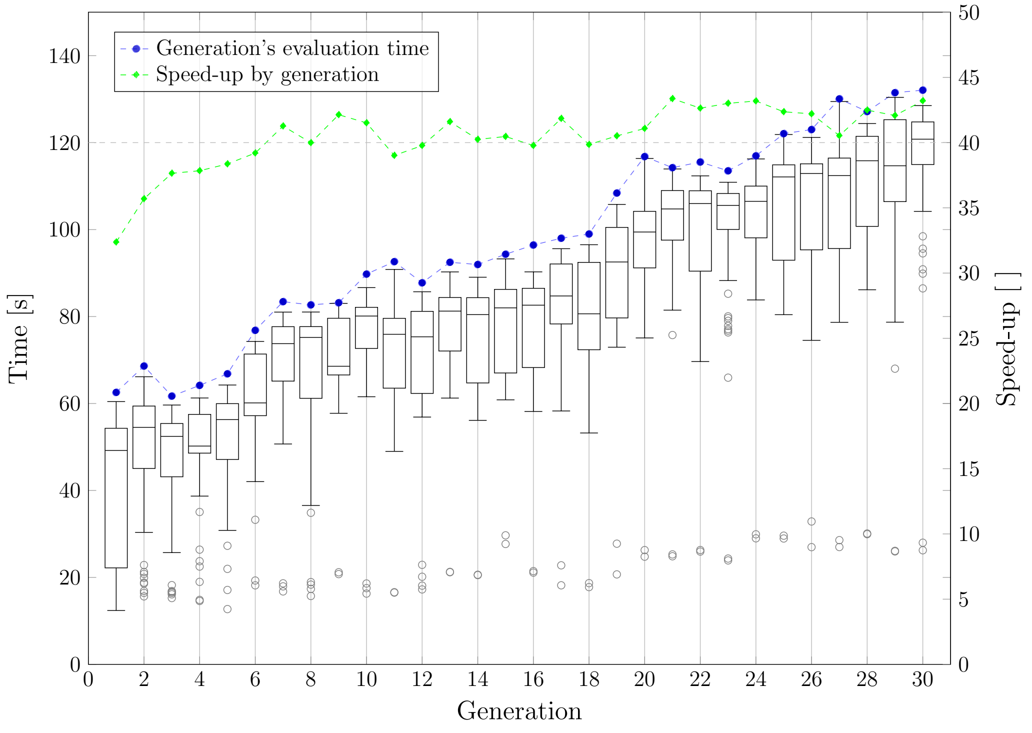

Fig. 8: The evaluation times through generations for Friedman #1. The blue dashed line represents the total time required for evaluating all individuals in each generation. The green dashed line shows the speed-up in each generation. The ideal speed-up corresponds to the number of individuals, in this case 50.

four important conclusions can be drawn:

1. The evaluation of a generation takes just a little more time than the training of the most complex ANN model in the population.
2. ANNs' training times do not have any upper outliers, meaning that the parallelization is sufficiently efficient.
3. As the optimization searches for a more accurate and diverse GeNNsem

model, the ANNs in the population become more and more complex resulting in increased training times.

4. As a consequence of the previous point, thanks to Amdahl's effect [52], the speed-up slightly increases through generations. In simple words, the more complex ANNs, the more benefit from the parallelization.

The maximum achievable speed-up in this configuration is constrained by the population size, set at 50 in our scenario. Achieving a consistent speed-up exceeding 40 represents a remarkable outcome.

6. Conclusion

This paper introduces the GeNNsem framework for the automated GA-based construction of neural network ensembles for regression tasks. The framework efficiently identifies the optimal architectures and hyperparameters of individual ANNs and creates an ensemble based on their training accuracy and diversity. The search process is guided not only by individual network performance, but also by the performance of the ensemble as a whole.

The proposed approach introduces a novel chromosome encoding and customized crossover and mutation operators to simultaneously optimize multiple aspects of neural network architecture and training-related parameters. The ensemble model's predictive performance improves over generations of the genetic algorithm, driven by the fitness of individual networks. By modifying the cost function to incorporate ensemble influence from the previous generation, subsequent models enhance prediction accuracy on samples with previously low precision.

GeNNsem outperformed all other approaches on the *Friedman #1* and Boston housing problems. For the real-world use case, GeNNsem emerged as the most competitive model, closely approaching the best-performing solution, with the notable observation that the problem itself is better suited for tree-based methods. Additionally, GeNNsem improves training efficiency by requiring fewer and less complex models, which enables transparent parallel training of ANNs and results in a training speed improvement of over 40 times. In contrast to other ensemble-building strategies proposed in the literature, GeNNsem provides a fully automated framework, enabling users across various domains to create high-quality ensemble models without needing expertise in ML model optimization.

The next step in improving the GeNNsem approach would be completing the whole AutoML pipeline by adding data preparation and feature engineering phases. Applying data preparation and especially feature engineering should lead to more accurate predictions, which will hopefully result in even better and more efficient ensembles.

Acknowledgement
This paper is funded through the EIT's HEI Initiative DEEPTECH-2M project, supported by EIT Digital and coordinated by EIT RawMaterials, funded by the European Union.

References

[1] L. Prechelt, et al., Proben1: A set of neural network benchmark problems and benchmarking rules (1994).

[2] D. E. Golberg, Genetic algorithms in search, optimization, and machine learning, Addion wesley 1989 (102) (1989) 36.

[3] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, R. M. Patton, Optimizing deep learning hyper-parameters through an evolutionary algorithm, in: Proceedings of the workshop on machine learning in high-performance computing environments, 2015, pp. 1–5.

[4] O. Sagi, L. Rokach, Ensemble learning: A survey, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (4) (2018) e1249.

[5] L. K. Hansen, P. Salamon, Neural network ensembles, IEEE transactions on pattern analysis and machine intelligence 12 (10) (1990) 993–1001.

[6] M. Ivanovic, V. Simic, B. Stojanovic, A. Kaplarevic-Malisic, B. Marovic, Elastic grid resource provisioning with WoBinGO: A parallel framework for genetic algorithm based optimization, Future Generation Computer Systems 42 (2015) 44–54.

[7] V. Simic, B. Stojanovic, M. Ivanovic, Optimizing the performance of optimization in the cloud environment–an intelligent auto-scaling approach, Future Generation Computer Systems 101 (2019) 909–920.

[8] M. Ivanovic, V. Simic, Efficient evolutionary optimization using predictive auto-scaling in containerized environment, Applied Soft Computing 129 (2022) 109610. doi:https://doi.org/10.1016/j.asoc.2022.109610.

[9] Kubernetes: Production-Grade Container Orchestration, accessed: 2023-08-05.
URL `https://kubernetes.io/`

[10] J. H. Friedman, E. Grosse, W. Stuetzle, Multidimensional additive spline approximation, SIAM Journal on Scientific and Statistical Computing 4 (2) (1983) 291–301.

[11] D. Harrison Jr, D. L. Rubinfeld, Hedonic housing prices and the demand for clean air, Journal of environmental economics and management 5 (1) (1978) 81–102.

[12] B. Stojanovic, M. Milivojevic, N. Milivojevic, D. Antonijevic, A self-tuning system for dam behavior modeling based on evolving artificial neural networks, Advances in Engineering Software 97 (2016) 85–95.

[13] S. Zinchenko, D. Lishudi, Star algorithm for neural network ensembling, Neural Networks 170 (2024) 364–375. doi:https://doi.org/10.1016/j.neunet.2023.11.020.

[14] A. Ara, M. Maia, F. Louzada, S. Macêdo, Regression random machines: An ensemble support vector regression model with free kernel choice, Expert Systems with Applications 202 (2022) 117107. doi:https://doi.org/10.1016/j.eswa.2022.117107.

[15] T. T. Le, W. Fu, J. H. Moore, Scaling tree-based automated machine learning to biomedical big data with a feature set selector, Bioinformatics 36 (1) (2020) 250–256.

[16] J. Li, H. Wang, H. Luo, X. Jiang, E. Li, A ranking prediction strategy assisted automatic model selection method, Advanced Engineering Informatics 57 (2023) 102068. doi:https://doi.org/10.1016/j.aei.2023.102068.

[17] B. Mohan, J. Badra, A novel automated superlearner using a genetic algorithm-based hyperparameter optimization, Advances in Engineering Software 175 (2023) 103358. doi:https://doi.org/10.1016/j.advengsoft.2022.103358.

[18] M. Zhang, H. Li, S. Pan, J. Lyu, S. Ling, S. Su, Convolutional neural networks-based lung nodule classification: A surrogate-assisted evolutionary algorithm for hyperparameter optimization, IEEE Transactions on Evolutionary Computation 25 (5) (2021) 869–882. doi:10.1109/TEVC.2021.3060833.

[19] M. Milivojevic, Methods for creating and adaptations regression models based on genetic algorithms, Ph.D. thesis, Faculty of Science, University of Kragujevac (2016).

[20] P. Vidnerová, R. Neruda, Evolution strategies for deep neural network models design., in: ITAT, 2017, pp. 159–166.

[21] A. Kaplarević-Mališić, B. Andrijević, F. Bojović, S. Nikolić, L. Krstić, B. Stojanović, M. Ivanović, Identifying optimal architectures of physics-informed neural networks by evolutionary strategy, Applied Soft Computing 146 (2023) 110646. doi:https://doi.org/10.1016/j.asoc.2023.110646.

[22] M. Shi, Y. Tang, X. Zhu, Y. Huang, D. Wilson, Y. Zhuang, J. Liu, Genetic-GNN: Evolutionary architecture search for graph neural networks, Knowledge-Based Systems 247 (2022) 108752. doi:https://doi.org/10.1016/j.knosys.2022.108752.

[23] F. Ming, W. Gong, Y. Jin, Growing neural gas network-based surrogate-assisted pareto set learning for multimodal multi-objective optimization, Swarm and Evolutionary Computation 87 (2024) 101541. doi:https://doi.org/10.1016/j.swevo.2024.101541.

[24] B. Fritzke, A growing neural gas network learns topologies, in: G. Tesauro, D. Touretzky, T. Leen (Eds.), Advances in Neural Information Processing Systems, Vol. 7, MIT Press, 1994.

[25] L. Ma, N. Li, G. Yu, X. Geng, S. Cheng, X. Wang, M. Huang, Y. Jin, Pareto-wise ranking classifier for multiobjective evolutionary neural architecture search, IEEE Transactions on Evolutionary Computation 28 (3) (2024) 570–581. doi:10.1109/TEVC.2023.3314766.

[26] L. Ma, H. Kang, G. Yu, Q. Li, Q. He, Single-domain generalized predictor for neural architecture search system, IEEE Transactions on Computers 73 (5) (2024) 1400–1413. doi:10.1109/TC.2024.3365949.

[27] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[28] D. H. Wolpert, Stacked generalization, Neural networks 5 (2) (1992) 241–259.

[29] R. T. Clemen, Combining forecasts: A review and annotated bibliography, International journal of forecasting 5 (4) (1989) 559–583.

[30] L. Breiman, Stacked regressions, Machine learning 24 (1) (1996) 49–64.

[31] R. E. Schapire, The boosting approach to machine learning: An overview, Nonlinear estimation and classification (2003) 149–171.

[32] A. Ara, M. Maia, F. Louzada, S. Macêdo, Random machines: A bagged-weighted support vector model with free kernel choice, Journal of Data Science 19 (3) (2021) 409–428. doi:10.6339/21-JDS1014.

[33] L. Fdez-Díaz, J. R. Quevedo, E. Montañés, Regularized boosting with an increasing coefficient magnitude stop criterion as meta-learner in hyperparameter optimization stacking ensemble, Neurocomputing 551 (2023) 126516. doi:https://doi.org/10.1016/j.neucom.2023.126516.

[34] G. Ngo, R. Beard, R. Chandra, Evolutionary bagging for ensemble learning, Neurocomputing 510 (2022) 1–14. doi:https://doi.org/10.1016/j.neucom.2022.08.055.

[35] R. Caruana, A. Niculescu-Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: Proceedings of the twenty-first international conference on Machine learning, 2004, p. 18.

[36] D.-H. Lee, D.-S. Kang, The application of the artificial neural network ensemble model for simulating streamflow, Procedia engineering 154 (2016) 1217–1224.

[37] A. Chandra, X. Yao, Ensemble learning using multi-objective evolutionary algorithms, Journal of Mathematical Modelling and Algorithms 5 (4) (2006) 417–445.

[38] S. Gu, Y. Jin, Generating diverse and accurate classifier ensembles using multi-objective optimization, in: 2014 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), IEEE, 2014, pp. 9–15.

[39] M. Chen, H. Peng, J. Fu, H. Ling, One-shot neural ensemble architecture search by diversity-guided search space shrinking, arXiv preprint arXiv:2104.00597 (2021). doi:https://doi.org/10.48550/arXiv.2104.00597.

[40] S. Soares, C. H. Antunes, R. Araújo, Comparison of a genetic algorithm and simulated annealing for automatic neural network ensemble development, Neurocomputing 121 (2013) 498–511, advances in Artificial Neural Networks and Machine Learning. doi:https://doi.org/10.1016/j.neucom.2013.05.024.

[41] J. Sun, S. Wu, H. Zhang, X. Zhang, T. Wang, Based on multi-algorithm hybrid method to predict the slope safety factor– stacking ensemble learning with bayesian optimization, Journal of Computational Science 59 (2022) 101587. doi:https://doi.org/10.1016/j.jocs.2022.101587.

[42] J. Serrano-Guerrero, B. Alshouha, M. Bani-Doumi, F. Chiclana, F. P. Romero, J. A. Olivas, Combining machine learning algorithms for personality trait prediction, Egyptian Informatics Journal 25 (2024) 100439. doi:https://doi.org/10.1016/j.eij.2024.100439.

[43] L. Ren, H. Zhang, A. Sekhari Seklouli, T. Wang, A. Bouras, Stacking-based multi-objective ensemble framework for prediction of hypertension, Expert Systems with Applications 215 (2023) 119351. doi:https://doi.org/10.1016/j.eswa.2022.119351.

[44] A. J. Sharkey, A genetic algorithm approach for creating neural network ensembles, in: Combining artificial neural nets, Springer, 1999, pp. 79–99.

[45] E. J. Herron, S. R. Young, T. E. Potok, Ensembles of networks produced from neural architecture search, in: International Conference on High Performance Computing, Springer, 2020, pp. 223–234.

[46] S. R. Young, D. C. Rose, T. Johnston, W. T. Heller, T. P. Karnowski, T. E. Potok, R. M. Patton, G. Perdue, J. Miller, Evolving deep networks

using hpc, in: Proceedings of the Machine Learning on HPC Environments, 2017, pp. 1–7.

[47] Z.-H. Zhou, J.-x. Wu, W. Tang, Z.-q. Chen, Combining regression estimators: GA-based selective neural network ensemble, International Journal of Computational Intelligence and Applications 1 (04) (2001) 341–356.

[48] F. Chollet, et al., Keras, `https://keras.io` (2015).

[49] V. Simic, Elastic reservation of distributed computing resources in multicriteria optimization processes based on genetic algorithms, Ph.D. thesis, Faculty of Science, University of Kragujevac (2016).

[50] Amazon Elastic Kubernetes Service (EKS), accessed: 2023-08-05.
URL `https://aws.amazon.com/eks/`

[51] D. Harrison Jr, D. L. Rubinfeld, StatLib archive, Carnegie Mellon University, The Boston Housing Dataset, accessed: 2023-08-05.
URL `http://lib.stat.cmu.edu/datasets/boston`

[52] M. J. Quinn, Parallel programming, TMH CSE 526 (2003) 105.

Appendix A. Parameters and Benchmark Analysis of Classic Star

This section presents the selected parameters for the Classic Star algorithm, informed by recommendations and guidelines from the authors. It includes additional benchmark tests that compare the algorithm's performance with proposed architectures and other models' hyperparameters.

Table A.1: Classic Star Ensemble parameters used in benchmarks.

Parameter	Value
Number of models (d)	5
Epochs	200
Loss	MSELoss
Learning rate (lr)	0.01

Table A.2: Comparison of Classic Star Ensemble models' performance with different ANN architectures on the test sets.

Benchmark problem	Classic Star Ensemble	MAE	RMSE	PRMSE	R^2
Friedman #1	Random Search models (1.b)	**0.10**	**0.13**	**0.57%**	**0.9993**
	GA optimized models (1.c)	0.21	0.29	1.26%	0.9968
	Proposed ANN architecture (128-64-32-16)	0.48	0.63	2.70%	0.9851
Boston Housing	Random Search models (1.b)	**1.96**	**2.53**	**5.80%**	**0.9351**
	GA optimized models (1.c)	2.14	2.71	6.21%	0.9254
	Proposed ANN architecture (128-64-32-16)	2.00	2.78	6.37%	0.9214
Grancarevo dam	Random Search models (1.b)	**0.89**	**1.15**	**2.30%**	**0.9909**
	GA optimized models (1.c)	1.53	2.30	4.62%	0.9632
	Proposed ANN architecture (128-64-32-16)	1.61	2.26	4.55%	0.9643

Appendix B. AutoML powered ensembles - selected parameters

In this section, we present the parameters of AutoML methods used in our benchmark problems based on the proposed instructions of their authors.

Table B.1: Random Regression Machines (RRM) parameters used in benchmarks.

Parameter	Value
Epsilon (ϵ)	0.1
The cost (C)	1
The number of bootstrap samples (B)	25
The degree of polynomial kernel (d)	2
Gamma (γ) of Gaussian kernel function	1
Gamma (γ) of Laplacian kernel function	1
Beta (β)	2
Automatic tuning	True

Table B.2: Parameters of Tree-based pipeline optimization tool (TPOT) used in benchmarks.

Parameter	Value
Generations	25
Population size	100
Offspring	100
Mutation rate	0.9
Crossover rate	0.05
Random state	42
Number of folds in CV	5

Table B.3: Hyperparameters search space in AutoSL-GA optimization.

Model	Hyperparameters	Minimum value	Maximum value
ANN	No. of neurons in hidden layers	10	250
	Alpha	1e-6	1.0
	Tol	1e-6	1e-4
	Max iterations	200	2000
SVM	C	1e-6	100
	Degree	1	10
	Kernel	poly, rbf, sigmoid	
ENR	Alpha	1e-6	100
	L1 ratio	1e-6	1.0
	Fit intercept	True, False	
	Normalize	True, False	
	Max iterations	1e3	1e6
	Tol	1e-8	1e-2
KRR	Alpha	1e-6	1.0
	Gamma	1e-4	1.0
LGB	Boosting type	gbdt, dart, goss	
	No. of leaves	20	100
	No. of estimators	100	1e4
	Learning rate	1e-6	1.0
CBR	Depth	5	10
	Bagging temperature	1	10
	Learning rate	1e-6	1.0
	Iterations	30	1e3

Table B.4: Genetic algorithm parameters used in AutoSL-GA optimization in benchmarks.

Parameter	Value
Population size	100
Generations	25
Elite percentage	0.1
Crossover probability	0.5
Mutation probability	0.1