AD-NEv: A Scalable Multilevel Neuroevolution Framework for Multivariate Anomaly Detection

Marcin Pietroń, Dominik Żurek, Kamil Faber, and Roberto Corizzo, *Member, IEEE*

***Abstract*— Anomaly detection tools and methods present a key capability in modern cyberphysical and failure prediction systems. Despite the fast-paced development in deep learning architectures for anomaly detection, model optimization for a given dataset is a cumbersome and time-consuming process. Neuroevolution could be an effective and efficient solution to this problem, as a fully automated search method for learning optimal neural networks, supporting both gradient and nongradient fine-tuning. However, existing methods mostly focus on optimizing model architectures without taking into account feature subspaces and model weights. In this work, we propose anomaly detection neuroevolution (AD-NEv)—a scalable multilevel optimized neuroevolution framework for multivariate time-series anomaly detection. The method represents a novel approach to synergically: 1) optimize feature subspaces for an ensemble model based on the bagging technique; 2) optimize the model architecture of single anomaly detection models; and 3) perform nongradient fine-tuning of network weights. An extensive experimental evaluation on widely adopted multivariate anomaly detection benchmark datasets shows that the models extracted by AD-NEv outperform well-known deep learning architectures for anomaly detection. Moreover, results show that AD-NEv can perform the whole process efficiently, presenting high scalability when multiple graphics processing units (GPUs) are available.**

***Index Terms*— Anomaly detection, autoencoders, neural architecture search, neuroevolution.**

Nomenclature

N	Number of samples in the original training dataset.
M	Number of sensors.
σ	Reduction rate.
X	Original training dataset.
x	Data point from X.
X_r	Reduced training dataset.
x_{r_i}	Data point from X_r.
l_w	Window size.
P	Population in a genetic algorithm.
N_g	Iterations (generations) in a genetic process.
N_p	Size of a population.
p_m	Probability of mutation.
G_i	ith subspace.
S_i	ith solution.
F_i	ith solution.
$F[i]_{\text{ic}}$	Input channels of ith layer.
$F[i]_{\text{oc}}$	Output channels of ith layer.
$F[i]_K$	1-D filter size of ith layer.
$F[i]_P$	Padding value of ith layer.
$F[i]_c^B$	Number of channels in batch norm of ith layer.
P_{G_i}	Population of models specific for subspace G_i.
$\delta(F, G(X))$	Loss function.
$\Delta(F, G)$	Fitness function.
$d(F_i, F_j)$	Distance between models F_i and F_j.
L_{F_i}	Layers of model F_i.
$L_{\max}$	Max number of layer in F_i.
$c_{\max}$	Max difference between out and in channels.
$\gamma(l)$	Number of channels in layer l.
$F'_{G_i}(x)$	Model classifying input as normal or anomaly.
$F_e(x)$	Ensemble classification model.
θ	Weight in neural network.
p_c	Crossover probability.
p_m	Mutation probability.
τ	Mutation power.

Manuscript received 15 September 2022; revised 5 October 2023 and 6 May 2024; accepted 29 July 2024. This work was supported in part by the PLGrid Infrastructure, in part by the Polish Ministry of Science and Higher Education allocated to the AGH University, and in part by Program "Excellence Initiative—Research University" for AGH University. *(Corresponding author: Marcin Pietroń.)*

Marcin Pietroń, Dominik Żurek, and Kamil Faber are with the Faculty of Computer Science, AGH University of Krakow, 30-059 Kraków, Poland (e-mail: pietron@agh.edu.pl).

Roberto Corizzo is with the Department of Computer Science, American University, Washington, DC 20016 USA.

Digital Object Identifier 10.1109/TNNLS.2024.3439404

I. Introduction

Modern cyberphysical and failure prediction systems involve sophisticated equipment that records multivariate time-series data from several up to thousands of features. Such systems need to be continuously monitored to prevent expensive failures. In anomaly detection, it is common to have abundant availability of normal data deriving from sensor monitoring and scarcity of labeled anomalies. For this reason, most anomaly detection works focus on semi-supervised learning settings, where model training is conducted exclusively using normal data [1], as well as unsupervised learning settings, where training data are mostly normal but may contain a small number of unknown anomalies. Among recent works on semi-supervised and unsupervised multivariate time-series anomaly detection, deep learning-based methods achieve the best results on well-known benchmarks [2], [3], [4], [5], [6], [7], [8], [9].

Within deep learning methods, a wide spectrum of autoencoder-based approaches were designed to deal with the anomaly detection problem [10], [11], [12], [13], [14]. The most efficient are those based on convolutional, fully connected, and long short-term memory (LSTM) layers, or a combination of them in single model. Alternative methods are based on adversarial techniques [6] as well as variational autoencoders (VAEs) [15], [16]. Other recent and very promising trends include autoencoders based on graph neural networks [17], [18], [19], generative adversarial networks (GAN)-based architectures [3], supervised classification models [20], ensemble autoencoders [21], and autoencoders with attention [22].

In this article, one important problem is the identification of a suitable and optimized architecture for a given dataset, in a fully automated way. Neuroevolution is a form of artificial intelligence that uses evolutionary approaches to find optimal neural networks. The most popular forms of neuroevolution algorithms include NeuroEvolution of Augmenting Topologies (NEAT) [23], HyperNEAT [24], and coDeepNEAT [25], which aim to optimize parameters, model architectures, or both. However, despite its potential, neuroevolution approaches have been focused on the optimization of model architectures without taking into account the joint optimization of model architectures with feature subspaces and model weights.

In this article, we aim to fill this gap proposing anomaly detection neuroevolution (AD-NEv), a multilevel neuroevolution approach that aims to identify robust and optimized autoencoder architectures for anomaly detection. Inspired by the framework formulated in [26], loosely based on the coDeepNEAT algorithm, our novel approach involves the simultaneous evolution of two populations: models and subspaces. The former contains neural network architectures that evolve during the neuroevolution process. The latter consists of subspaces, which define subsets of input features. After the neuroevolution process, the framework sets up a bagging technique-based ensemble model from single optimized architectures. A distinctive feature of our proposed approach stands in the optimization of a single model for each subspace, overcoming the limitation of a single suboptimal solution for all subspaces. Subsequently, a fully scalable nongradient fine-tuning process is performed to iteratively select the best solutions and generate model populations. In addition, fine-tuning is performed on the evolved ensemble model. Another key novelty of our proposed approach stands in the fully automated nature of the optimization process, encompassing subspace evolution, model evolution, and fine-tuning. To the best of the authors' knowledge, this research direction has never been explored before in the context of anomaly detection tasks. Our extensive experimental evaluation shows that our proposed multilevel neuroevolution approach yields deep ensembles of autoencoder models that outperform state-of-the-art methods without requiring any predefined scoring function.

The main contributions of our work can be summarized in the following.

1) A novel multilevel neuroevolution approach with a separate population of models for each subspace of features, which can be evolved independently, leading to a better adaptation of specific models to each subspace.
2) A novel selection process for models evolution based on an adapted distance measure for deep autoencoders that promotes model diversity.
3) A fast nongradient-based fine-tuning approach for the evolved model architecture, leveraging adaptations of neural network weights in the evolution process, which improves results achieved by the previous levels.
4) Automatic induction of a regularized ensemble model with a low number of submodels, which further improves anomaly detection performance.
5) An extensive evaluation with benchmark datasets that are widely adopted for multivariate anomaly detection.

This article is organized as follows. Section II describes related works. Section III presents the proposed neuroevolution approach in detail. Section IV describes and discusses our experiments. Section V summarizes the key results obtained in our study. Section VI focuses on our ablation study. Finally, Section VII concludes this article and outlines directions for future work.

II. Related Works

In this section, we analyze the anomaly detection methods for multivariate time-series data, as well as neuroevolution methods that are most relevant for our research scope. Recent surveys on general anomaly detection [27], deep learning-based anomaly detection [1], [28], [29], [30], [31], and unsupervised time-series anomaly detection [32] present techniques relevant to unsupervised and semi-supervised multivariate time-series anomaly detection. Autoencoder-based methods include fully connected autoencoder (FC AE), unsupervised anomaly detection (USAD) [6], and univariate fully connected autoencoder (UAE) [21]. These methods have become prominent in a number of real-world applications, such as cybersecurity [10], [11], energy [12], physics [13], and medical imaging [14].

LSTM-based methods include National Aeronautics and Space Administration (NASA)-LSTM [20], LSTM-AE [21] (which is based on [2]), and LSTM-VAE [33]. CNN-based methods include temporal convolutional AE (TCN AE) [34]. GAN-based methods include one-class adversarial nets (OCAN) [9] and BeatGAN [3]. Graph neural network-based approaches include [17]. Finally, hybrid approaches include Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) [35], deep autoencoding Gassian mixture model (DAGMM) [36], and OmniAnomaly [5].

Regarding autoencoder-based approaches, the FC AE model introduced in [21] is similar to UAE, but it involves a single model over all the features, where the input sample is a vector resulting from the concatenation of time steps observed for all sensors. The USAD model is an autoencoder with an additional discriminator model and loss extensions to boost the final scores. Garg et al. [21] present comparative studies on multivariate anomaly detection models. They describe UAE as a model consisting of multiple autoencoders, each connected by its input to a separate feature. Each encoder is a multilayer perceptron with a number of nodes corresponding to the number of time steps (window size) and a reduced number of dimensions in the latent space by a factor of 2. The decoder is a mirror image of the encoder with tanh activation. The resulting ensemble model outperforms many other deep learning architectures.

The attention mechanism for anomaly detection is exploited in PAFormer [22], where attention weights are used to learn the global–local distributional differences for each data point, enabling to discriminate anomalies.

Focusing on LSTM-based approaches, NASA-LSTM is a two-layer LSTM model that uses predictability modeling, i.e., forecasting for anomaly detection [20]. The LSTM-AE presented in [21] consists of single-LSTM layer for each encoder and decoder. LSTM-VAE [15] models the data generating process from the latent space to the observed space using variational techniques.

A VAE-based method with evolutionary features is proposed in [16]. However, unlike our proposed framework, the method neither consider weights mutation in the model nor subspace evolution. The model is evolved only for one specialized ECG dataset.

The CNN-based approach of TCN AE described in [21] is an architecture in which the encoder is built from a stack of temporal convolution (TCN) [34] residual blocks. In the decoder, convolutions in the TCN residual blocks are replaced with transpose convolutions. The study shows that the scoring function has a significant impact on the pointwise $F1$-score.

GAN-based methods include OCAN [9], an end-to-end one-class classification method in which the generator is trained to produce examples that are complimentary to normal data patterns, which is used to train a discriminator for anomaly detection using the GAN framework. BeatGAN [3] uses a generative adversarial network framework where reconstructions produced by the generator are regularized by the discriminator instead of fixed reconstruction loss functions.

Graph-based models also represents a viable and powerful alternative for anomaly detection with time-series data. The dense graph neural network approach presented in [17] models the anomaly detection problem as a graph neural network, where each node represents a single feature, and edges allow to represent data exchanged between different nodes. The graph-based modeling capabilities represent a distinctive trait of this method and were shown to yield a significant performance boost other state-of-the-art methods with very well-known multivariate time-series datasets.

Zheng et al. [18] adopts a graph neural network to encode spatial information from complex pairwise dependencies between variables, and a module with dilated convolutional functions allows to capture temporal dependencies. A similar challenge is addressed in [19], where a cross-time spatial graph network with fuzzy embedding is proposed to disentangle latent and mixing temporal states and learn cross-time spatial dependencies.

Hybrid methods such as MSCRED [35] learn to reconstruct signature matrices, i.e., matrices representing cross correlation relationships between channels constructed by pairwise inner product of the channels. It is efficient in the case of long-term anomalies that are significantly out of the normal data distribution. Deep autoencoding Gaussian mixture model (GMM) [36] uses a deep autoencoder to generate a low-dimensional representation and reconstruction error for each input data point. The output of the autoencoder is further fed into a GMM. DAGMM jointly optimizes the parameters of the deep autoencoder and the mixture model in an end-to-end fashion. The joint optimization balances autoencoding reconstruction and density estimation of latent representation. The proposed regularization helps the autoencoder escape from less attractive local optima and further reduce reconstruction errors. OmniAnomaly [5] is a stochastic recurrent neural network for multivariate time series. Its main idea is to capture the normal patterns of multivariate time series by learning their robust representations with key techniques, such as stochastic variable connection and planar normalizing flow. Then, it reconstructs input data and uses the reconstruction probabilities to determine anomalies.

One common drawback of these methods is that they do not perform automatic model optimization, and therefore require a significant manual effort to identify and tune the right architecture for the right domain and dataset. This limitation may be solved by neuroevolution approaches, which have been recently used in many machine learning tasks for improving the accuracy of deep learning models and finding optimal network topologies [37], [38], [39], [40].

Miikkulainen et al. [25] show that a two-level neuroevolution strategy scheme can outperform human-designed models in some specific tasks, e.g., language modeling and image classification. This strategy is based on the co-deep NEAT algorithm with two optimization levels: single sub-block optimization and composition of sub-blocks to form a whole network. In [40], a novel deep reinforcement learning-based framework is proposed for electrocardiogram time-series signal. The framework is optimized by neuroevolution algorithm. Jiau and Huang [39] present the framework of the self-organizing map-based neuroevolution solver by which the self-organizing maps (SOMs)-like network represents the abstract carpool service problem. The SOM network is trained by using neural learning and evolutionary mechanism. In [38], the novel neuroevolution approach is described. The algorithm is incorporated with powerful representation which unifies most of the neural networks into one representation and with new diversity preserving method called spectrum diversity. The combination of spectrum diversity with a unified neuron representation enables the algorithm to either outperform or have similar performance with NEAT on five classes of problems tested. Ablation tests show the importance of new added features in the unified neuron representation. In [41], a novel neuroevolutionary method for optimization the architecture and hyperparameters of convolutional autoencoders. The hypervolume indicator in the context of neural architecture search is introduced. Results show that images can be compressed by a factor of more than 10, while still retaining enough information. In [42], it is shown that genetic algorithm could evolve autoencoders that can reproduce the data better than the manually created autoencoders with more hidden units. The experiments were performed on the MNIST dataset. The first approach of a co-evolutionary neuroevolution-based multivariate anomaly detection system is presented in [26]. However, one substantial limitation is that the optimization of subspaces and models occurs separately, so that one model is optimized for all subspaces. This characteristic limits the capability of the neuroevolution process to optimize the model for each specific subspace, forcing the model to compromise in order to handle all subspaces simultaneously, and potentially resulting in a loss of anomaly detection accuracy. Moreover, the proposed method does not provide fine-tuning capabilities, which would provide the opportunity to further optimize the model and improve anomaly detection performance. Fine-tuning is a quite popular technique for improving the accuracy of the pretrained models. The most popular technique is gradient-based fine-tuning [43], [44]. The nongradient approach is quite rare but can yield significant improvements

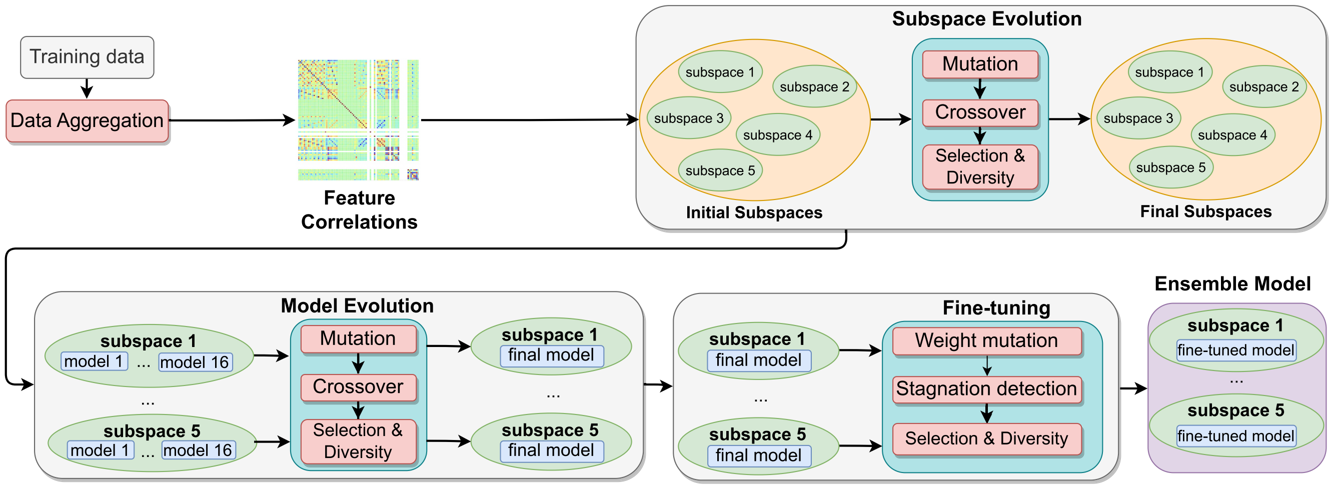


Fig. 1. AD-NEv framework architecture. Time-series data are used to train and evaluate a set of models during neuroevolution through different levels. The framework returns an optimized ensemble model that can be used for inference.

as shown in [45]. The presented method improves the accuracy of pretrained quantized models.

One common drawback of existing neuroevolution methods is that they typically optimize model architectures or model weights in isolation. Moreover, the majority of approaches that optimize model weights focus on shallow neural networks. Finally, to the best of the authors' knowledge, there is no neuroevolution approach that jointly optimizes feature spaces, model architecture, and model weights.

III. Method

In this section, we describe AD-NEv, a scalable multilevel neuroevolution framework that jointly addresses the aforementioned limitations of anomaly detection and neuroevolution methods. The starting point of the framework is data preparation—which consists of downsampling training data used in the following evolution levels, and splitting it into overlapping windows, which reduces the computational cost of the following steps. The next step is finding the optimal partition of input features into subspaces, leading to an effective matching between features and models, as well as a reduced number of models in the final ensemble. After that, model evolution is performed for each subspace. At the next level, the best model for each subspace extracted from the previous level is fine-tuned using the nongradient genetic optimization method. Subsequently, the ensemble model combines all fine-tuned models evaluating them using a voting mechanism. A visual representation of the framework architecture is shown in Fig. 1. In the following, we describe all levels in detail in Sections III-A–III-G.

A. Autoencoders

AD-NEv leverages a deep neural network architecture consisting of autoencoders as base models for the neuroevolution process. Autoencoders are unsupervised learning models that learn a compressed representation of raw input data, which is then used to accurately reconstruct inputs. They consist of two parts: an encoder E and a decoder D. The encoder learns how to efficiently compress and encode the input data X in a new representation with reduced dimensionality—latent variables Z. The decoder learns how to reconstruct the latent variables Z back to their original shape. The model is trained to minimize the *reconstruction loss*, which corresponds to minimizing the difference between the decoder's output and the original input data. It can be expressed as follows:

$$\mathcal{L}(X, \hat{X}) = ||X - \mathrm{AE}(X)||_2 \tag{1}$$

where

$$\mathrm{AE}(X) = D(Z), \quad Z = E(X). \tag{2}$$

The autoencoder in our method can consist of various layers, e.g., fully connected layers and convolutional layers. Several variants of autoencoder architecture exist, including adversarial autoencoder (AAE) [46]), VAE [47]), or denoising autoencoder (DAE) [48]) but the general idea of reconstruction loss minimization is similar in all cases, (1). When dealing with time-series data, the error is predicted at each time point (norm between input and multivariate reconstructed vector), as in the following equation:

$$Er_t = ||X_t - \mathrm{AE}(x_t)||_2. \tag{3}$$

B. Data Reduction

The algorithm starts by reducing the training data for the evolution process. Specifically, a consecutive number of data points σ are aggregated by averaging values for each feature, leading to a coarser time granularity. As a result, we are able to provide a fast evolution while retaining the most important information. The original training dataset X contains N samples, and each sample contains data from M features, as shown in the following equations:

$$X = \{x_t, t \in 1, 2, \dots, N\} \tag{4}$$
$$x_t = \{x_{t_i}, i \in \{1, 2, \dots, M\}\}. \tag{5}$$

The reduced dataset is annotated as X_r, whereas σ is the reduction ratio parameter

$$X_r = \left\{x_{r_t}, t \in 1, 2, \dots, \frac{N}{\sigma}\right\}. \tag{6}$$

Single data points from X_r are obtained according to the following equation, where med is the median function:

$$x_{r_t} = \text{med}(\{x_{t*\sigma}, x_{t*\sigma+1}, \dots, x_{(t+1)*\sigma}\}). \quad (7)$$

Although the averaging step may lead to information loss, our experiments with and without this step highlighted that the ranking of models remains fundamentally unaltered, while a significant increase in the efficiency of the model selection step can be obtained. Moreover, it should be noted that, in the proposed approach, averaging is just used as a reference to compare model efficiency. The final model architecture is trained using the whole training dataset in its original granularity. We have done tests and compared the ranking of the models after training on the reduced dataset and nonreduced. The final best architecture in all cases achieved the best $F1$-score in both cases.

During this step, data points are also grouped into overlapping windows used in the following steps. The rationale is based on the fact that working with time-series data, an aggregated context can be more beneficial for the algorithm than a single data point. The window size l_w can change during the evolution process since it is one of the parameters subject to mutation.

C. General Evolution Algorithm

In our multilevel neuroevolution framework, we apply a genetic algorithm in three levels: subspaces, models, and nongradient fine-tuning. The genetic process starts with the generation of the initial population P_0. Each population contains N_p solutions. After that, the single iteration is repeated N_g times. Each iteration starts with creating offspring from the parents by the crossover operator. Next, the mutation operator mutates each solution from the offspring with the probability p_m. The genetic process is formally described in Algorithm 1. Genetic operators such as crossover and mutation are different for subspaces (see Algorithms 2–5) and models (see Algorithms 6 and 7). The models' fine-tuning has a unique selection and mutation process (see Algorithm 9). Nevertheless, the structure of the genetic algorithm is the same for all levels. Details about each specific step of the process for all levels are provided later on in this article.

Subsequently, the fitness of each solution is calculated. In our approach, the generation process works on the model architecture in the evolution of a single model, on subspaces in the subspace optimization, and on weight values during fine-tuning. The combination of these three levels in our framework allows us to explore and exploit a larger search space during model optimization. A single genetic iteration finishes with the selection of solutions that form a new population. The result of the genetic process is the final population P_{N_g}.

D. Subspaces Evolution

The goal of this level of the algorithm is to find an optimal partitioning of input features into subspaces. We define a subspace G_i as a subset of the input features that is specific for each dataset, as shown in the following equation:

$$G_i(X) \subseteq \{X_0, X_1, X_2, \dots, X_M\} \quad (8)$$

where X_i denotes data from the ith input feature.

Algorithm 1 General Genetic Algorithm Workflow

```
Result: $P_{N_g}$ – Final population
Input: $N_g$ – Number of iterations
Input: $N_p$ – Size of the given population
Input: $p_m$ – Probability of mutation
1 $i \leftarrow 1$;
2 Generate initial population $P_0$ ;
3 while $i \leq N_g$ do
4     Create offspring (crossover) for subspaces (Alg. 2) or model (Alg. 7);
5     Mutate offspring for subspaces (Alg. 3, 4, 5) or model (Alg. 6);
6     Compute fitness (loss) of all solutions (Alg. 8) ;
7     Select best solutions for a new population $P_i$ using fitness and distance as in Eq. 14;
8     $i \leftarrow i + 1$;
9 return Final population $P_{N_g}$
```

Algorithm 2 Subspaces Crossover Algorithm

```
Result: $S'$— solution created by crossover
Input: $S_1$—parent solution with $K$ subspaces
Input: $S_2$—parent solution with $K$ subspaces
1 $S' = \{\}$;
2 for $i \in [0, 1, \dots, K]$ do
3     $g_1 = S_{1_i}$ ;                  // Subspace from $S_1$
4     $g_2 = S_{2_i}$ ;                  // Subspace from $S_2$
5     $g_{min} = \min(\min(g_1), \min(g_2))$;
6     $g_{max} = \max(\max(g_1), \max(g_2))$;
7     $\gamma = \text{randint}(g_{min}, g_{max})$ ;     // Split point
8     $g' = \{\}$;
9     for $\kappa \in g_1$ do
10        if $\kappa < \gamma$ then
11            $g' \leftarrow g' \cup \kappa$;
12    for $\kappa \in g_2$ do
13        if $\kappa > \gamma$ then
14            $g' \leftarrow g' \cup \kappa$;
15    $S' \leftarrow S' \cup g'$ ;
16 return $S'$
```

The partition S of input features contains K subspaces. There is no restriction on the frequency of the presence for a single input feature in subspaces, which means that it can be used in zero, one, or more subspaces. Our method leverages the genetic algorithm to find the optimal partition of the input features into subspaces. A single gene provides information about a given feature being present in a subspace G. To perform subspace evolution we adopt the genetic operators defined in [26]: crossover (Algorithm 2), moving mutation (Algorithm 3), vanishing mutation (Algorithm 4), and adding mutation (Algorithm 5).

To improve the convergence speed of the genetic algorithm, we form the initial population based on the correlation between features, instead of using a randomly generated population. Features are clustered performing agglomerative clustering with a degree of randomness to achieve a diverse population.

Algorithm 3 Subspaces Moving Mutation

```
Result: $S'$— solution created by mutation
Input: $S$—solution containing $K$ subspaces
Input: $P_m$—probability of a mutation
1  $S' \leftarrow \emptyset$ ;
2  for $i \in [0, 1, \ldots, K]$ do
3      sample $r$ from $\mathcal{N}(0, 1)$;
4      if $r < P_m$ then
5          sample $\kappa$ from $S_i$ ;  // Feature to move
6          $j = (i + 1) \bmod K$ ;  // id of the next
            subspace in solution
7          $S'_j \leftarrow S_j \cup \kappa$;
8          $S' \leftarrow S' \cup S'_j$
9  return $S$
```

Algorithm 4 Subspaces Vanishing Mutation

```
Result: $S'$— solution created by mutation
Input: $S$—solution containing $K$ subspaces
1  $S' \leftarrow \emptyset$ ;
2  for $S_i \in S$ do
3      $S'_i = S_i$ ;
4      for $\kappa \in S_i$ do
5          $C_\kappa = \sum_{S_i \in S} \begin{cases} 1, & \text{if } \kappa \in S_i \\ 0, & \text{otherwise} \end{cases}$ ;
6          sample $r$ from $\mathcal{N}(0, 1)$;
7          if $r > \frac{1}{C_\kappa}$ then
8              $S'_i \leftarrow S'_i \setminus \{\kappa\}$ ;
9      $S' \leftarrow S' \cup S'_i$ ;
10 return $S'$
```

This process has the effect of recursively merging pairs of clusters leveraging the linkage distance [49]. In our approach, we leverage correlations between features as an intuitive and automatic way to estimate their similarity and drive the clustering process.

E. Models Evolution

This level follows subspace evolution, and its aim is to find optimal models that can be later parts of the ensemble model. Models for each subspace are evaluated independently, since a model that is optimal for one subspace may not be efficient in another subspace. Therefore, we create and evaluate a single population of models for each subspace. We denote P_{G_i} as a population of models specific for subspace G_i

$$P_{G_i} = \left\{ F^j_\Theta(G_i), j = 0, \ldots, N_P \right\}. \tag{9}$$

Each model is represented by an encoder and a decoder, as in the following equations, where the layers in encoder and decoder appear in the reversed order

$$z = E\left(F^j_{\Theta^E}(G_i)\right) = f^j_{\theta^E_N}\left(f^j_{\theta^E_{N-1}}, \ldots, \left(f^j_{\theta^E_0}(G_i)\right)\right) \tag{10}$$

$$G^r_i = D\left(F^j_{\Theta^D}(z)\right) = f^j_{\theta^D_0}, \ldots, \left(f^j_{\theta^D_{N-1}}\left(f^j_{\theta^D_N}(z)\right)\right). \tag{11}$$

Algorithm 5 Subspaces Adding Feature Mutation

```
Result: $S'$— solution created by mutation
Input: $S$—solution containing $K$ subspaces
Input: $F$— set of all features
1  $S' \leftarrow \{S'_0, S'_1, \ldots, S'_K\}$, where $S'_i = S_i$ ;
2  for $\kappa \in F$ do
3      if $\kappa \notin S$ then
4          for $S_i \in S$ do
5              sample $r$ from $\mathcal{N}(0, 1)$;
6              if $r > \frac{1}{K}$ then
7                  $S'_i \leftarrow S'_i \cup \kappa$;
8  return $S'$
```

Performing $D(E(F^j_\Theta(G_i)))$, we obtain the reconstructed subspace—G^r_i.

Each population P_{G_i} is evolved independently from the others in order to find the best solution for each subspace G_i by means of a genetic algorithm. The genetic operators follow the specifications in [26] and include crossover and mutation of the following parameters: number of layers, number of input and output channels for each layer i ($F[i]_{\text{ic}}$, $F[i]_{\text{oc}}$), and window size (l_w) (Algorithms 6 and 7).[1]

If a mutation of the number of layers takes place, the L_F parameter is modified by removing the last Λ_l layers in the encoder ($f^j_{\theta^E_N}$, $f^j_{\theta^E_{N-1}}, \ldots$) (lines 10–12, Algorithm 6) and the first Λ_l layers in the decoder ($f^j_{\theta^D_N}$, $f^j_{\theta^D_{N-1}}, \ldots$) or adding new ones to the end of the encoder after the $f^j_{\theta^E_N}$ layer, and to the decoder before $f^j_{\theta^D_N}$ (lines 14–17, Algorithm 6). The mutation of the output channels in the specific layer is described in lines 3–7, Algorithm 6, and the window size mutation is presented between lines 18 and 20, Algorithm 6. In the case of a crossover, the chosen lth layer of the encoder $f_{\theta^E_l}$ and the decoder $f_{\theta^D_l}$ are exchanged between two child models F'_1 and F'_2 in a subgroup population (lines 4–8, Algorithm 7). The next option of crossover is to exchange the length between the two parent autoencoders (lines 10–18, Algorithm 7).

A key aspect of our method is that it does not require the selection of a single value for the mutation stage. To simplify this task, users can provide a range to define the search space for the mutation stage, delegating the responsibility of choosing a proper window size value to the framework, reducing the user's margin for error.

As the loss function δ for model F and subspace G on dataset X, we use the mean squared error defined as follows:

$$\delta(F, G(X)) = \frac{1}{|G(X)|} \sum_{x \in G(X)} (F(x) - x)^2. \tag{12}$$

The genetic algorithm needs to calculate the fitness Δ for each single model F and subspace G. To achieve this goal, the methods relies on a windowed training dataset X which is split into consecutive parts according to the timestamp of data points: a training part X_t (80%) and validation part X_v (20%).

[1]Without loss of generality, Algorithms 6 and 7 operate on any layer of the neural network. For conciseness, we do not report the processing of the encoder and decoder parts of the model.

Algorithm 6 Single Model Mutation

```
Input: F - a model
Input: $w_{max}$ - maximum window size, $L_{max}$ - maximum number of conv layers
1  $F' \leftarrow copy(F)$;
2  sample $m$ from $\{0, 1, 2\}$ ;    // mutation types
3  if $m = 0$ then
       // mutate the number of channels in a layer
4      sample $l$ from $\{0, \dots, L_F - 1\}$;
5      $c' = randint(F[l]_{ic}, F[l+1]_{ic})$;
6      $F'[l]_{oc} \leftarrow c'$;
7      $F'[l+1]_{ic} \leftarrow c'$;
8  if $m = 1$ then
       // reduce the length of the model
9      sample $l$ from $\{0, \dots, L_{max}\}$;
10     if $l < L_F$ then
11         for $k \in [l+1, \dots, L_F]$ do
12             $F'[k] \leftarrow \emptyset$ ;
13     else
14         for $k \in [L_F, \dots, l-1]$ do
15             $c' = randint(F[k]_{oc}, F[k]_{oc} + c_{max})$;
16             $F'[k+1]_{ic} \leftarrow F[k]_{oc}$;
17             $F'[k+1]_{oc} \leftarrow c'$;
18 if $m = 2$ then
       // mutate window size
19     sample $w$ from $\{1, \dots, w_{max}\}$;
20     $F'[0]_{ic} \leftarrow w$;
21 return $F'$
```

Algorithm 7 Models Crossover

```
Input: $F_1$, $F_2$
1  $F'_1 \leftarrow copy(F_1)$;
2  $F'_2 \leftarrow copy(F_2)$;
3  sample $m$ from $\{0, 1\}$ ;   // type of crossover
4  if $m = 0$ ;               // exchange the layers
5  then
6      sample $l$ from $\{0, 1, \dots, \min(L_{F_1}, L_{F_2})\}$ ;
7      $F'_1[l] \leftarrow F_2[l]$;
8      $F'_2[l] \leftarrow F_1[l]$;
9  if $m = 1$ ;       // exchange the lengths of models
10 then
11     if $L_{F_1} > L_{F_2}$ then
12         $F'_1[L_{F_2} - 1]_{oc} \leftarrow F_2[L_{F_2} - 1]_{oc}$;
13         for $k \in [L_{F_2}, \dots, L_{F_1}]$ do
14             $F'_1[k] \leftarrow F_2[k]$;
15             $F'_2[k] \leftarrow \emptyset$ ;
16     else
17         $F'_2[L_{F_1} - 1]_{oc} \leftarrow F_1[L_{F_1} - 1]_{oc}$;
18         for $k \in [L_{F_1}, \dots, L_{F_2}]$ do
19             $F'_2[k] \leftarrow F_1[k]$;
20             $F'_1[k] \leftarrow \emptyset$ ;
21 return $F'_1$, $F'_2$
```

During the evolution, the model is trained on the training part X_t. After each evolution iteration, we calculate the fitness as the weighted loss from the validation datasets. The value is negated because the goal of the genetic algorithm is to maximize the fitness, whereas we want to minimize loss values. The whole calculation is expressed in the following equation:

$$\Delta(F, G) = -\frac{|X_t| * \delta(F, G(X_t)) + |X_v| * \delta(F, G(X_v))}{|X_t| + |X_v|}. \quad (13)$$

Our method introduces a novel selection process in the evolution of the models. Its goal is to avoid convergence to a local optimum by keeping diversity in the models' population. To achieve this goal, we modify the selection process. Instead of choosing only the best models in each generation, we also keep a few of the most different models. We calculate the distance d_F between models F_i and F_j. The value is based on the models' hyperparameters, such as the number of layers L and the number of channels γ in the convolutional or fully connected layer. The distance calculation is performed as follows:

$$d_F(F_i, F_j) = L_{F_i} - L_{F_j} + \sum_{l_a, l_b \in L_{F_i}, L_{F_j}} \frac{\text{abs}(\gamma(l_a) - \gamma(l_b))}{\min(\gamma(l_a), \gamma(l_b))} \quad (14)$$

where abs denotes the absolute value.

Algorithm 8 Model Fitness Calculation

```
Result: Fitness value for solution $S$
Input: $S$ - solution containing $K$ subspaces
Input: $X_t$ - train dataset
Input: $X_v$ - validation dataset
Input: $N_{ep}$ - number of epochs to train while calculating fitness
1  $\mathcal{L} \leftarrow 0$ ;
2  for $S_i \in S$ do
3      $\mathcal{F} = \text{train\_model}(X_t, S_i, N_{ep})$;
4      $\mathcal{L}_t = \delta(F, S_i(X_t))$ ; // Rec. loss (Eq. 12)
5      $\mathcal{L}_v = \delta(F, S_i(X_v))$ ;
6      $\mathcal{L}_X = \frac{|X_t|}{|X_t|+|X_v|} * \mathcal{L}_t + \frac{|X_v|}{|X_t|+|X_v|} * \mathcal{L}_v$;
7      $\mathcal{L}_{S_i} = \frac{\mathcal{L}_X}{|S_i|}$;
8      $\mathcal{L} = \mathcal{L} + \mathcal{L}_{S_i}$
9  return $-\mathcal{L}$
```

F. Nongradient Fine-Tuning

Changing the weights values of pretrained models with gradient-based methods can result in suboptimal models that could be further optimized. This phenomenon was noticed and presented in [45], which shows that performing nongradient fine-tuning after gradient-based optimization can yield more accurate models. Inspired by this work, we perform nongradient fine-tuning on the best models extracted from the previous level. As the optimization step, we choose the evolutionary approach in which the genetic operators modify weights values to improve the performance of the models. The mutation operator modifies the chosen weights by the mutation power τ with mutation probability p_m as in

$$\theta' = \theta * (1 \pm p_m * \tau) \quad (15)$$

where θ is a single weight.

Algorithm 9 Fine-Tuning

```
Result: $R$ - A set of fine-tuned models
Input: $F$ - The best model from previous level
Input: $N_P$ - Size of fine-tuning population in every generation
Input: $N_g$ - Number of generations in the fine-tuning
Input: $p_m$ - Mutation probability
Input: $\tau$ - Mutation power
1  $R \leftarrow \emptyset$ ;
2  $g \leftarrow 0$ ;                          // Generation number
3  $F_g \leftarrow F$ ;
4  while $g < N_g$ do
5      $P_g \leftarrow \{F_{\Theta'_1}, F_{\Theta'_2}, \ldots, F_{\Theta'_{N_P}}\}$ where $\theta'$ is created by
       mutating weights from $F_g$ using eq. 15 with $p_m$
       and $\tau$;
6      $F_g \leftarrow \arg\min_{F_i \in P_g} FP(F_i)$ ;
7      if $FP(F_g) = 0 \vee$
8        $\forall_{k \in [g-5, g-4, \ldots, g-1]} FP(F_k) = FP(F_g)$ ;   // is
         stagnated
9      then
10         $R \leftarrow R \cup F_g$ ;
11         $F_g \leftarrow \arg\max_{F' \in P} \; d_F(F, F')$ ; // $d_F(\cdot,\cdot)$ from
           eq. 16
12     $g \leftarrow g + 1$
13 return $R$
```

During the fine-tuning process formalized in Algorithm 9, the algorithm randomly mutates a percentage of the weights[2] (line 5, Algorithm 9). The fitness for each solution is calculated in the population as the number of false positives (FPs) (line 6, Algorithm 9). During the fine-tuning process, a sample is marked as an anomaly (FP) if its reconstruction error is higher than the average value of all reconstruction errors, multiplied by a constant factor that determines the allowed deviation from the mean. The best solutions are selected to be included in the new population (lines 6–11, Algorithm 9). In the following iteration, a complete new set of models is generated based on the best models selected from the previous iteration.

If a fitness value of zero FPs is achieved (line 7, Algorithm 9), or when a model does not further improve for a given number of iterations, i.e., stagnation condition (line 8, Algorithm 9), no further improvement is possible. We recall that this assumption holds since model training takes place in a semi-supervised setting, in which training data contains no anomalies. If zero FPs are achieved or the stagnation condition occurs, the best model is removed from the population and is used for the inference phase, during which it is evaluated on testing data.

In all cases, after removing the best model, another model is selected (line 11, Algorithm 9) and further improved in the following iterations. The new model is selected based on the highest distance from the best model, as shown in the following equation:

$$d_\theta(F_i, F_j) = \sum_{k=0}^{\text{Layers}(F_i)} \sqrt{|\theta_{F_{ik}} - \theta_{F_{jk}}|^2}. \tag{16}$$

The distance represents the difference between the values of the weights of the two models. The process of selecting the most diverse model aims at achieving the exploration of a larger space of models, while avoiding getting stuck in a local minima.

The fine-tuning process does not involve the crossover operation, since its adoption would provide the same effect as a very high value of the p_m parameter. In that case, model weights would likely drift to a wrong direction in a single iteration, leading to a drastic loss in model's accuracy (as further showcased in Section VI-B).

G. Ensemble Model

The outcome of the previous steps can be summarized as follows.

1) An optimized partition of input features into subspaces: $\{G_i, i \in \{0, \ldots, K\}\}$.
2) An optimized and fine-tuned model for each subspace G_i: $\{F_{G_i}, i \in \{0, \ldots, K\}\}$.

Our next goal is to build an ensemble model that is capable of classifying input data as normal or as an anomaly. To accomplish this goal, we add a threshold to each model to return a binary prediction: either 0 (normal) or 1 (anomaly). Therefore, for each subspace G_i, we create a model F'_{G_i} that compares the output of F_{G_i} to the threshold η_i, specifically defined for each model

$$F'_{G_i}(x) = \begin{cases} 0, & F_{G_i}(x) < \eta_i \\ 1, & F_{G_i}(x) \geq \eta_i. \end{cases} \tag{17}$$

As a result of this process, we compose base models into the ensemble model F_e adopting a crisp voting strategy

$$F_e(x) = \begin{cases} 0, & \sum_{i=0}^{K} F'_{G_i}(x) = 0 \\ 1, & \sum_{i=0}^{K} F'_{G_i}(x) > 0. \end{cases} \tag{18}$$

$$F_e(x) \iff (F_{G_1}(x_{G_1}), F_{G_2}(x_{G_2}), \ldots, F_{G_K}(x_{G_K})). \tag{19}$$

This voting mechanism was empirically determined and selected since it gives better results than standard approaches such as majority voting. As a result, the ensemble model F_e can classify input data as normal or an anomaly based on the output of single models.

IV. Experiments

Our experiments focus on multivariate anomaly detection with time-series data, which represents a challenging task that is crucially important in many real-life applications.

More specifically, our experiments aim to address the following research questions.

1) *RQ1:* Does AD-NEv yield a higher anomaly detection performance than state-of-the-art anomaly detection methods for multivariate time series?

[2]To change weights, the plus/minus sign is randomly chosen with a probability of 50%.

2) *RQ2:* Are all modeling levels in AD-NEv contributing to the achievement of the final anomaly detection performance of the method?
3) *RQ3:* Does AD-NEv model training present a reasonable time complexity in terms of execution time for all its levels (training, subspaces, models, and fine-tuning)?
4) *RQ4:* Does the AD-NEv framework present satisfactory speedup values, allowing to efficiently scale model optimization when using multiple general purpose graphics processing units (GPGPU)?
5) *RQ5:* Does AD-NEv present a stable and efficient model evolution and fine-tuning convergence process considering different values for the population size, τ, and mutation probability parameters?

To answer these questions, we perform a set of quantitative and qualitative experiments that assess the performance of AD-NEv from different perspectives. In the following subsections, we describe the evaluation protocol followed in our study, the datasets analyzed, the metrics used to assess the anomaly detection performance, as well as competitor methods considered and their configuration. Results are discussed in Section V.

A. Datasets

We selected relevant time-series datasets, including the Secure Water Treatment (SWAT) Dataset (SWAT) [50], the Water Distribution (WADI) dataset [51], the Mars Science Laboratory (MSL) Rover dataset, and the Soil Moisture Active Passive (SMAP) Satellite dataset [20]. All datasets either involve data gathered in real environments (MSL and SMAP) or carefully prepared testbeds reflecting existing systems (SWAT and WADI). The datasets contain anomalies resulting from either device malfunctions or external malicious activity. The anomalies were labeled by the domain experts who created the datasets. It is noteworthy that the chosen datasets are popular benchmarks used in many recent studies.[3]

For all datasets, we leveraged splits provided by the authors of each dataset, which properly considers the continuity of time-series data. Training data contains clean conditions of the process without anomalies, whereas testing data contains both normal data and anomalies (class 0: normal/benign and class 1: anomaly).

Accordingly, we adopt a semi-supervised anomaly detection setting, where normal/benign data are exclusively used for model training (one-class learning). As a result, our models are exclusively trained on normal data, without anomalies. Both training and testing data are processed in windows, according to the value of window size (in channels in the Conv1D layer) chosen during model optimization. Statistics about datasets characteristics are shown in Table I.

While the SWAT and WADI datasets contain one continuous data stream (SWAT: 11 days of continuous operation: 7 days under normal operation and four days with attack scenarios—WADI: 16 days of continuous operation: 14 days under normal operation and 2 days with attack scenarios), SMAP and MSL are more complex, and they are split into smaller subsets called entities. All data samples between 5 and 3 days before the first observed anomaly are used for model training, and all data points between 3 days before and 2 days after the first observed anomaly are used for model evaluation. We concatenate data from these subsets for the evolution of subspaces and models. Following this idea, to better fit specific data characteristics of entities, we build subspaces separately for each entity, and then train and fine-tune separate models. The final result is obtained by concatenating results from each entity. During prediction time, we choose the model corresponding to the particular entity of the source that generated the data instance.

[3]See supplementary material in our external appendix: https://github.com/neuroevolution-agh/AD-NEv.

TABLE I
STATISTICS OF THE USED DATASETS

Datasets	Features	Training size	Test size	Anomalies
SWAT	51	49,668	44,981	11.97%
WADI-2017	123	1,048,571	172,801	5.99%
WADI-2019	123	784,571	172,801	5.77%
SMAP	55	138,004	435,826	12.82%
MSL	27	58,317	73,729	10.48%

B. Metrics

The $F1$-score is a standard metric for anomaly detection tasks, as it is more resistant to class imbalance than other metrics such as accuracy (which yield high values when the normal class in testing data is larger than the anomaly class). We adopt $F1$ as a pointwise metric, i.e., we evaluate every data point independently. While sequence-based evaluation (point-adjusted $F1$) also represents a feasible choice, and it is largely adopted in the literature, it usually results in higher values for performance metrics (since predictions of anomalies in a sequence are considered correct if at least one anomaly in the sequence is identified correctly), without providing precise information about how well the algorithm can decide whether a single data point is an anomaly [52].

V. RESULTS AND DISCUSSION

A. Anomaly Detection Performance

Table II presents the results for AD-NEv in comparison to the results of well-known anomaly detection methods. Some models are evaluated using an optimized Gauss-D scoring function [21], which features a data transformation step on previously observed data points, aiming at increasing F_1 values. However, in our experiments, this scoring function did not provide significant improvements to the model's accuracy and was therefore discarded in the final experiments. For the largest dataset (WADI), AD-NEv outperforms the second-best method [graph neural network (GNN)] by 0.05, whereas F_1 of the third-best model (LSTM-VAE) is significantly lower, with a margin of 0.12. Most of the other methods obtain even worse results. The differences are less impressive with the SWAT dataset, where AD-NEv outperforms GNN and USAD by a value of 0.01 and 0.02, respectively. The smaller difference is probably related to the fact that SWAT is smaller and less complex than WADI, which results in methods being more aligned in terms of predictive performance. In the case of MSL, all the best models provide results that are very close to each other. However, AD-NEv improves the results of NASA-LSTM and TCN AE by 0.02. However, AD-NEv

TABLE II

POINTWISE F_1 SCORE. *—MODELS WERE EVALUATED USING GAUSS-D SCORING FUNCTION; THE BEST RESULT FOR EVERY DATASET IS IN **BOLD**

	SWAT	WADI	MSL	SMAP	Mean
MAD-GAN	0.77	0.37	0.20	0.24	0.40
GNN	0.81	0.57	0.30	0.33	0.50
USAD	0.79	0.43	0.48	0.49	0.55
CNN 1D	0.78	0.27	0.44	0.52	0.50
NASA LSTM*	0.13	0.20	0.55	0.59	0.37
UAE*	0.58	0.47	0.54	0.58	0.54
LSTM-AE*	0.45	0.33	0.54	0.53	0.46
LSTM-VAE*	0.42	0.50	0.49	0.49	0.48
TCN AE*	0.43	0.43	0.55	0.55	0.49
BeatGAN	0.48	0.46	0.53	0.57	0.51
OCAN	0.15	0.0	0.30	0.28	0.18
DAGMM	0.0	0.13	0.14	0.17	0.11
OmniAnomaly	0.15	0.24	0.41	0.38	0.29
AD-NEv	**0.82**	**0.62**	**0.57**	**0.77**	**0.70**

TABLE III

STATISTICAL ANALYSIS WITH WILCOXON SIGNED RANK TESTS COMPARING ALL PAIRWISE COMBINATIONS OF METHODS. BOLD: COMPARISON IS STATISTICALLY SIGNIFICANT (p-VALUE < 0.01)

Methods	p-value	Winner
AD-Nev vs. MAD-GAN	**9.99E-05**	AD-Nev
AD-Nev vs. GNN	**1.17E-03**	AD-Nev
AD-Nev vs. USAD	**3.61E-04**	AD-Nev
AD-Nev vs. CNN 1D	**3.61E-04**	AD-Nev
AD-Nev vs. NASA LSTM	**1.10E-06**	AD-Nev
AD-Nev vs. UAE	**2.48E-05**	AD-Nev
AD-Nev vs. LSTM-AE	**3.15E-08**	AD-Nev
AD-Nev vs. LSTM-VAE	**3.15E-08**	AD-Nev
AD-Nev vs. TCN AE	**3.15E-08**	AD-Nev
AD-Nev vs. BeatGAN	**1.97E-07**	AD-Nev
AD-Nev vs. OCAN	**3.15E-08**	AD-Nev
AD-Nev vs. DAGMM	**3.15E-08**	AD-Nev
AD-Nev vs. OmniAnomaly	**3.15E-08**	AD-Nev

provides a radical improvement with the SMAP dataset, where it outperforms the second-best model (NASA-LSTM) by 0.18.

Overall, it is noteworthy that the models extracted by AD-NEv achieve the best results across all datasets when compared with all the other methods. For some datasets, the difference is very significant, while for others it is relatively smaller. However, the other methods considered in our experiments usually achieve high results with only a few of the analyzed datasets. For example, NASA-LSTM is the second-best method for MSL and SMAP datasets, but it does not handle SWAT and WADI very well (in which F_1 is 0.13 and 0.20, respectively). The opposite situation is in the case of GNN. It is the second-best model on WADI and SWAT but is significantly worse on SMAP and MSL datasets. The reason behind this fact is that in the SMAP and MSL datasets, there is a small number of dominant sensors (most of them have constant values and can be treated as noise signals). It shows that AD-NEv subspace evolution can be the crucial stage that helps to overcome this specific time-series multisensor data problem. On the other hand, AD-NEv showcases the capability to adjust the evolving model to specific dataset requirements thanks to the neuroevolution approach. This result can be clearly observed by looking at the mean value of F_1 across all datasets in Table II. To validate the statistical significance of our results, we apply Wilcoxon signed rank tests to all pairwise combinations of methods across multiple executions with all datasets. Results in Table III highlight that our method significantly outperforms all considered baslines (*RQ1*).

B. Ablation Study: Detection Performance

Table IV presents results for an ablation study consisting of three models: 1) a baseline model [26]; 2) after our subspaces and models evolution; and 3) after the fine-tuning level. This table reveals the improvements introduced by our newly introduced approaches for subspaces optimization, model evolution, and fine-tuning. Comparing the results from [26] with the results following our improvements in subspaces and model evolution, we see improvements in terms of F_1 on SWAT by 0.01, WADI by 0.05, MSL by 0.05, and SMAP by 0.15. These results show that the introduced novelties, such as the independent population for each subspace, improve the results with all datasets (*RQ2*). Fine-tuning also positively impacts all datasets: SWAT by 0.01, WADI by 0.03, MSL by 0.07, and SMAP by 0.10. The gains are lower for SWAT and WADI, as the results for these datasets are already higher. Those results confirm the positive contributions introduced by the new components proposed in this article, and highlight that nongradient fine-tuning can be an efficient method to boost the anomaly detection performance of pretrained models.

TABLE IV

ABLATION STUDY SHOWING POINTWISE F_1 SCORE USING A BASELINE MODEL WITH BASIC SUBSPACE AND MODEL EVOLUTION AND WITHOUT FINE-TUNING (A), USING AD-NEV WITH ENHANCED SUBSPACE AND MODEL EVOLUTION (B), AND WITH THE FULL VERSION OF AD-NEV THAT INCLUDES FINE-TUNING (C). THE BEST RESULT FOR EACH DATASET IS REPORTED IN **BOLD**

Method	SWAT	WADI	MSL	SMAP
A: Baseline model [26]	0.79	0.54	0.45	0.52
B: AD-NEv: Evolution	0.81	0.59	0.50	0.67
C: AD-NEv: Evolution + Fine-tuning	**0.82**	**0.62**	**0.57**	**0.77**

C. Time Complexity

The framework was implemented with Python and PyTorch.[4] All presented calculations were executed leveraging Nvidia Tesla V100-SXM2-32 GB graphics processing units (GPUs). Since AD-NEv supports parallel model optimization, experiments were performed using a pool of eight GPGPUs (each subspace/model was processed on a separate GPGPU).

We provide the values of the parameters of the neuroevolution process for each level in Tables V and VI. We also present the execution times in Fig. 2. It is important to note that SMAP and MSL datasets are processed in full, without downsampling, since most features are binary, and this operation would result in information loss. Considering all datasets, the evolution of the subspace level took a minimum of 0.8 and a maximum of 18 h. The best results in terms of accuracy were achieved with a value of the maximum

[4]https://github.com/neuroevolution-agh/AD-NEv

TABLE V
HYPERPARAMETERS OF THE GENETIC PROCESS DURING EACH LEVEL

Parameter	Subspaces	Models	Fine-tuning
Population size	16	24	24
Mutation probability	0.1	0.5	0.02
Crossover probability	0.1	0.5	not applicable
Number of generations	10	16	64

TABLE VI
OTHER HYPERPARAMETERS OF THE NEUROEVOLUTION PROCESS

Parameter	Value
Reduction ratio σ (WADI, SWAT)	5
Reduction ratio σ (MSL, SMAP)	1
Mutation power τ	$\frac{1}{256}$
Number of layers range (WADI, SWAT)	[2-7]
Number of layers range (MSL, SMAP)	[2-12]
Number of output channels range	[16-6144]
Filter size range	[1-7]
Window size range	[1-12]
Learning rate range	[0.000001-0.1]

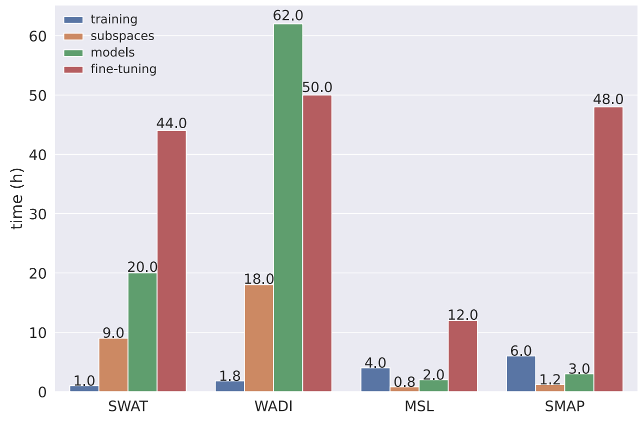


Fig. 2. Execution times in hours using eight GPGPUs with all datasets (SWAT, WADI, MSL, and SMAP).

number of possible subspaces (parameter K of the evolution process) set to 5. The rationale for this choice was to find the highest number yielding a reasonable execution time. We experimentally observed that a higher value for K exceeds the time limits of the evolution process (>140 h). Considering all datasets, the model evolution process took a minimum of 2 and a maximum of 62 h. Fine-tuning was on average more time-consuming, taking between 12 and 50 h. It is noteworthy that model training and fine-tuning were performed separately for each entity in the SMAP and MSL datasets. Therefore, the whole neuroevolution process, including the final training and evaluation, lasted the following execution times: SWAT—74 h; WADI—131.8 h; MSL—18.8 h; and SMAP—58.2 h. Overall, all the executions for each single dataset required less than 132 h, which we consider to be a reasonable time frame considering the size of the datasets and the satisfactory accuracy achieved by the resulting models (*RQ3*).

D. Scalability

One aspect worth discussion is the scalability of the proposed method. It is noteworthy that the execution time can be easily reduced by adding more GPGPUs, since each model can

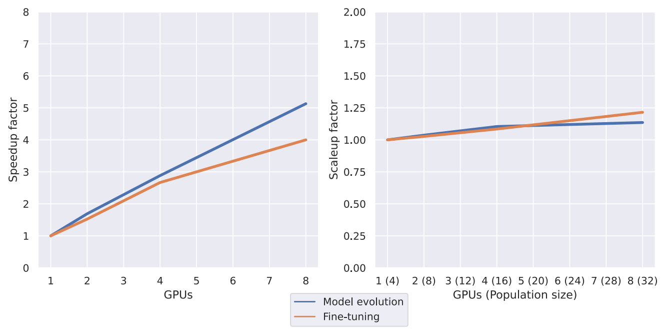


Fig. 3. Scalability results showing speedup (left) and scaleup (right) factors for both model evolution and fine-tuning levels with an increasing number of GPUs and population size (for scaleup only) on the x-axis.

be trained and evaluated independently on a separate GPGPU. Results in Fig. 3 show that a speedup of up to 4× can be obtained for the fine-tuning level and up to 5× for the model evolution level.[5] The results also highlight that the model evolution level of the AD-NEv framework exhibits a linear speedup with an increasing number of GPUs. The fine-tuning process also benefits from the execution on multiple GPUs, achieving comparable performances with respect to the model evolution level. Fig. 3 also shows results in terms of scaleup, where problem complexity (population size) and computational resources (GPUs) increase at the same time, i.e., from a population size of 4 with one GPU, to a population size of 32 with eight GPUs. Results show a remarkable performance which positions very close to the ideal curve, i.e., a constant scaleup of 1 across all configurations. Both speedup and scaleup results support our assumption that multiple GPGPUs can significantly reduce the execution time of the AD-NEv framework, effectively distributing the training and evaluation of multiple models on different GPUs, thus resulting in a significant speedup. Overall, the AD-NEv framework can be effectively scaled, supporting the efficient optimization of a large number of models (*RQ4*).

The models extracted after the neuroevolution process differ depending on the specific dataset and subspace. We present the complete architecture of the best model for the best subspace in every dataset in the supplementary material. The final models for the SWAT and WADI datasets consist of three convolutional layers in the encoder and the decoder parts of the model. However, the SWAT model presents a higher number of channels. The models for MSL and SMAP consist of six convolutional layers in the encoder and decoder. After the first two levels of the framework and before fine-tuning, the final training was run with 120 epochs in the case of the WADI dataset, 90 epochs for SWAT, and 250 for SMAP and MSL.

It is noteworthy that the presented AD-NEv framework requires a smaller number of computational units that the state-of-the-art ensemble method in [21], which achieves the best anomaly detection performance on multivariate time-series benchmarks. While the model in [21] requires a number of submodels equal to the number of features, $S = M$, AD-NEv produces an ensemble model with K submodels working on K subspaces, $K \ll M$. The complexity and memory reduction

[5]The hyperparameter values specification for the scalability experiment are reported in Tables V and VI.

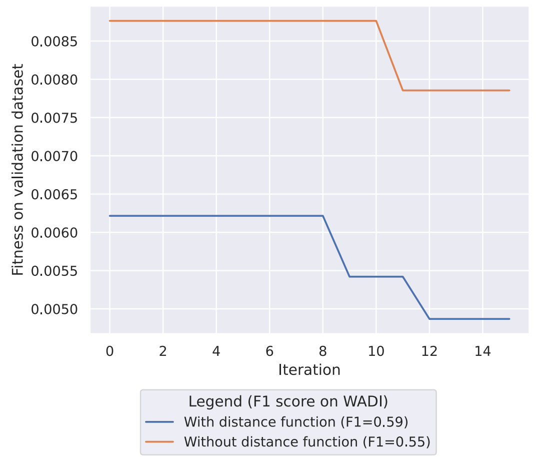


Fig. 4. Convergence plot for the WADI dataset during the model's evolution for population size 24. The diversity in the selection process introduced by the distance function improves not only convergence but also $F1$ score on the test dataset ($F1$ 0.59 in comparison to 0.55 without distance function).

of our solution is $\simeq(K/M)$. (The impact of the input size is negligible.)

VI. Ablation Study

This section presents the impact of neuroevolution parameters on the model's accuracy. A detailed analysis of the most complex dataset (WADI) is presented in the following, focusing on a selection of relevant model parameters. Regarding the SMAP and MSL datasets, a separate model was generated and tuned for each entity (see Section IV-A). For this reason, we do not present partial results for those datasets. It should be noted that, for all datasets, the behavior is consistent with that observed with WADI.

A. Convergence of the Model Evolution Process

Fig. 4 presents the fitness on the validation dataset (not containing anomalies) of the best model in each iteration during model evolution process for version: 1) with the distance function for model evolution $d_F(F_i, F_j)$ introduced in Section III-E—*blue* and 2) without the distance function—*orange*. The distance function aims at achieving model diversity during the model evolution process. It could be observed that, in the scenario where the distance function is adopted, the value of the loss function was persisting on the same level over the first eight iterations. During the following eight iterations, the fitness value was improved twice. Finally, the model with the lowest value of loss function was evaluated on the test dataset achieving F_1 score equal to 0.59. In the case of the scenario without diversity introduced by the distance function, the stagnation was persistent over ten iterations, and it improved only once in the following iterations. Moreover, in this case, the loss value was always higher than the scenario with distance function. The best model evolved without distance function achieved the final F_1 score of 0.54. We can clearly see that introducing the distance function $d_F(F_i, F_j)$ ensuring diversity to the population during evolution improved not only the convergence but also the anomaly detection performance (*RQ5*). Table VII presents F_1 scores for WADI

TABLE VII
$F1$ Score on WADI and SWAT Datasets After Evolution Model Process Depending on Population Size

Population size	**F_1 on WADI**	**F_1 on SWAT**
8	0.54	0.79
16	0.56	0.80
24	0.59	0.81

TABLE VIII
Ablation Study: Final Window Size (Best-Selected Model) and Window Size Evolution at Different Landmarks (Min–Max Range)

Dataset	**Final**	**Iter. 1-4**	**Iter. 5-8**	**Iter. 9-12**	**Iter. 13-16**
SWAT	5	[2-7]	[3-7]	[3-6]	[2-6]
WADI	6	[1-6]	[1-7]	[2-7]	[3-7]
MSL	4	[4-8]	[4-7]	[4-6]	[4-6]
SMAP	5	[3-6]	[3-6]	[4-6]	[4-6]

and SWAT on the test dataset, with different values of the population size parameter. As expected, the most effective models were extracted with the highest value for this parameter (24), showing the contribution of a large population of models during the evolution process. The largest population size was set to 24. Regarding the evolution of the window size parameter, we present a convergence and ablation study showing the ranges of values and the final window size for the best-selected model in Table VIII. Our experiments were conducted with a simple selection of window sizes in a range of [1–12]. We observe that the behavior of the algorithm is rather stable, converging to window size values that lie in a subrange of the initially defined search space. Additional ablation studies include layer size during evolution (see Table IX), and the number of layers of the best selected model (see Table X). For the number of output channels, we defined our search space in the range [16–6144]. We observe that the last encoder layer size lies in a range between 205 and 1041. Interestingly, a smaller capacity is used in SWAT and WADI datasets, when compared with the MSL and SMAP datasets. We also observe that, as the iterations increase, the model converges to smaller subranges of layer size, confirming the stability of the algorithm and the proper selection of the initial search space. The same phenomenon is observed for the number of layers in the final encoder, as shown in Table X. The filter size range used in our experiments is defined as: [1–7]. For the sake of brevity, we omit showing the full model architecture. More detailed model specifications are provided in an external appendix for the interested reader.[6]

B. Convergence of the Nongradient Fine-Tuning Process

Results in Table XI show the average convergence and the F_1 obtained on the WADI dataset using different configurations for N_p, p_m, and τ parameters. Results show that with an increasing value of the population size, the algorithm required a reduced number of iterations to achieve a zero value of FP (eight iterations on average for $N_P = 8$ and 4.92 on average iterations for $N_P = 24$). Moreover, a higher number of N_P also translates to an improved anomaly detection performance, resulting in an increased F_1 value.

[6]https://github.com/neuroevolution-agh/AD-NEv

TABLE IX

ABLATION STUDY: FINAL LAYERS CHANNELS FOR THE BEST-SELECTED MODEL (LEFT) AND MIN–MAX RANGE OF OUTPUT CHANNELS DURING EVOLUTION AT DIFFERENT ITERATIONS (RIGHT). THE FINAL MODELS ARE REPRESENTED BY THE SEQUENCE OF INPUT AND OUTPUT DEPTHS OF THE ENCODER LAYERS

Dataset	**Final**	**Iter. 1-4**	**Iter. 5-8**	**Iter. 9-12**	**Iter. 13-16**
SWAT	[5,84][84,123][123,205]	[53-713]	[53-461]	[53-284]	[53-284]
WADI	[6,91][91,153][153,155]	[40-669]	[40-669]	[40-460]	[40-377]
MSL	[4,95][95,126][126,138][138,261][261,685][685,927]	[14-1038]	[14-957]	[32-930]	[32-1038]
SMAP	[5,75][75,70][70,273][273,437][437,694][694,1041]	[20-934]	[20-681]	[20-1041]	[20-1041]

TABLE X

ABLATION STUDY: FINAL NUMBER OF LAYERS (BEST-SELECTED MODEL) AND EVOLUTION AT DIFFERENT LANDMARKS (MIN–MAX RANGE)

Dataset	**Final**	**Iter. 1-4**	**Iter. 5-8**	**Iter. 9-12**	**Iter. 13-16**
SWAT	3	[3-6]	[3-6]	[3-6]	[2-5]
WADI	3	[4-6]	[4-6]	[2-6]	[2-5]
MSL	6	[3-12]	[2-9]	[2-9]	[2-9]
SMAP	6	[3-8]	[3-8]	[2-9]	[2-10]

TABLE XI

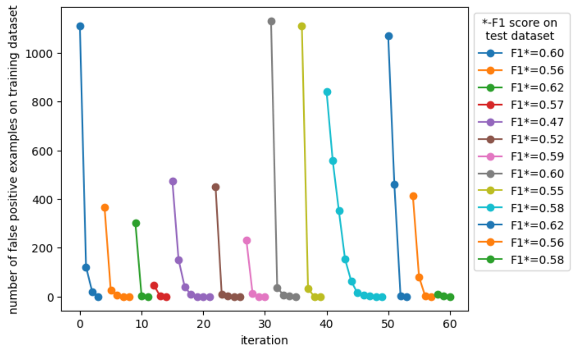
FINE-TUNING CONVERGENCE (NUMBER OF ITERATIONS) AND F_1 OBTAINED WITH VARYING VALUES OF N_P, p_m, AND τ

N_P	p_m	τ	F_1 **on WADI**	**Avg convergence** (iterations)
8	0.02	$\frac{1}{256}$	0.60	8
16	0.02	$\frac{1}{256}$	0.61	5.82
24	0.02	$\frac{1}{256}$	0.62	4.92
24	0.02	$\frac{1}{128}$	0.59	2.67
24	0.02	$\frac{1}{512}$	0.62	12.8
24	0.05	$\frac{1}{256}$	0.53	21.33

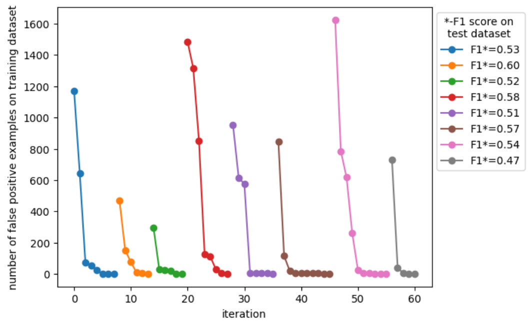
Fig. 5(a) represents a detailed example for row 3 in Table XI (best configuration), showing the values of the fitness function. The best $F1$ score observed is 0.62. Decreasing the population size from 24 to 16 leads to a minor decrease of the highest $F1$ performance (0.61). Decreasing the population size to 8 leads to another slight decrease in $F1$ (0.60), as shown in Fig. 5(b). Increasing the value of mutation power from $\tau = (1/256)$ to $\tau = (1/128)$ as shown in Fig. 5(c) leads to a minimal drop in $F1$ performance drop (0.59). Additional experiments reveal that increasing the mutation probability from 0.02 to 0.05 leads to a more significant drop in $F1$ (0.53).

Another perspective of results in Table XI is how fast the model was able to converge with different values of the mutation power τ. Results (rows 3–5) shows that the optimal model was obtained faster when this parameter had a higher value (1/128), but it does not have an impact on the final F_1 score. Moreover, using a large value for τ allows the process to converge faster. It is worth noting that reducing the value of the τ parameter to $\tau = (1/512)$ allowed us to obtain the same F_1 score as $\tau = (1/128)$ but the algorithm required a higher number of iterations to converge. We argue that generating a compact set of models may be preferred over a large number of models to reduce the time impact of model evaluation on testing data.

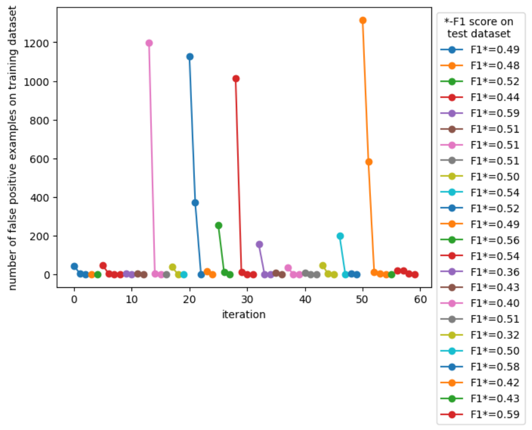
Finally, one interesting aspect worth analysis is how increasing the value of mutation probability p_m from 2% ($p_m = 0.02$) to 5% ($p_m = 0.05$) resulted in a decrease of efficiency for the algorithm. In this case, the best achieved F_1 score was 0.53,



(a)



(b)



(c)

Fig. 5. Fine-tuning convergence in terms of a different number of FPs for the best-generated model in each iteration. The experiments were conducted on the WADI dataset with 64 iterations and different configurations for population size (N_p), mutation power (τ), and mutation probability (p_m). (a) $N_p = 24$, $\tau = (1/256)$, and $p_m = 0.02$. (b) $N_p = 8$, $\tau = (1/256)$, and $p_m = 0.02$. (c) $N_p = 24$, $\tau = (1/128)$, and $p_m = 0.02$. We also report the $F1$ performance for each converged fine-tuned model.

which is worse than the base F_1 value of 0.59. Comparing rows 3 and 5, it could be observed that when too many weights are modified during a single iteration, the convergence process is slower. Overall, the convergence process for the fine-tuning level is stable and efficient with respect to selected ranges of values of population size, τ, and mutation probability parameters (*RQ5*).

VII. Conclusions and Future Work

In this article, we proposed AD-NEv—a novel multilevel framework for evolving ensemble deep learning autoencoder architectures for multivariate anomaly detection. Its novelty consists of an efficient multilevel optimization that includes the evolution of input features subspaces, the model architecture for each subspace, and non-gradient fine-tuning. To further boost the anomaly detection performance, AD-NEv builds an ensemble model with voting to combine the outputs of single optimized models. We show that each introduced optimization component contributes to the achievement of the final model accuracy. An extensive experimental evaluation results show that the final ensemble model outperforms state-of-the-art anomaly detection models for multivariate time-series data, while presenting a reasonable execution time, even with large datasets. AD-NEv's execution can be seamlessly scaled up by adding computational resources. Directions for future work include the integration of the framework with redefined genetic operators for graph neural networks. Another possibility worth investigating is the introduction of modified operators that enhance neural network architectures, such as different types of layers in a model or adding dense connections to each layer. Finally, we will further explore possible data distillation strategies to further reduce training data and reduce the execution time of the evolution level.

Acknowledgment

The authors also acknowledge the support of the Polish Ministry of Education and Science assigned to AGH University of Krakow, Kraków, Poland.

References

[1] G. Pang, C. Shen, L. Cao, and A. Van Den Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–38, Mar. 2019.

[2] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder–decoder for multi-sensor anomaly detection," *CoRR*, pp. 1–5, Jul. 2016.

[3] B. Zhou, S. Liu, B. Hooi, X. Cheng, and J. Ye, "BeatGAN: Anomalous rhythm detection using adversarially generated time series," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4433–4439.

[4] Y. Zhang, Z. Y. Dong, W. Kong, and K. Meng, "A composite anomaly detection system for data-driven power plant condition monitoring," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4390–4402, Jul. 2020.

[5] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 2828–2837.

[6] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: Unsupervised anomaly detection on multivariate time series," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 3395–3404, doi: 10.1145/3394486.3403392.

[7] S. Tariq et al., "Detecting anomalies in space using multivariate convolutional LSTM with mixtures of probabilistic PCA," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2123–2133.

[8] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," in *Proc. Workshops 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–9.

[9] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, "One-class adversarial nets for fraud detection," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, no. 1, pp. 1286–1293.

[10] H. Torabi, S. L. Mirtaheri, and S. Greco, "Practical autoencoder based anomaly detection by using vector reconstruction error," *Cybersecurity*, vol. 6, no. 1, pp. 1–13, Jan. 2023.

[11] K. Faber, R. Corizzo, B. Sniezynski, and N. Japkowicz, "VLAD: Task-agnostic VAE-based lifelong anomaly detection," *Neural Netw.*, vol. 165, pp. 248–273, Aug. 2023.

[12] R. Corizzo, M. Ceci, G. Pio, P. Mignone, and N. Japkowicz, "Spatially-aware autoencoders for detecting contextual anomalies in geo-distributed data," in *Proc. Int. Conf. Discovery Sci.* Halifax, NS, Canada: Springer, 2021, pp. 461–471.

[13] T. Finke, M. Krämer, A. Morandini, A. Mück, and I. Oleksiyuk, "Autoencoders for unsupervised anomaly detection in high energy physics," *J. High Energy Phys.*, vol. 2021, no. 6, pp. 1–32, Jun. 2021.

[14] N. Shvetsova, B. Bakker, I. Fedulova, H. Schulz, and D. V. Dylov, "Anomaly detection in medical imaging with deep perceptual autoencoders," *IEEE Access*, vol. 9, pp. 118571–118583, 2021.

[15] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1544–1551, Jul. 2018.

[16] A. J. Hashim, M. A. Balafar, J. Tanha, and A. Baradarani, "AEVAE: Adaptive evolutionary autoencoder for anomaly detection in time series," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Dec. 6, 2023, doi: 10.1109/TNNLS.2023.3337243.

[17] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *Proc. AAAI Int. Conf. Artif. Intell.*, 2021, pp. 4027–4035.

[18] Y. Zheng et al., "Correlation-aware spatial–temporal graph learning for multivariate time-series anomaly detection," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Nov. 14, 2023, doi: 10.1109/TNNLS.2023.3325667.

[19] K. Zhu, P. Song, and C. Zhao, "Fuzzy state-driven cross-time spatial dependence learning for multivariate time-series anomaly detection," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 8, 2024, doi: 10.1109/TNNLS.2024.3371109.

[20] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Söderström, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 387–395.

[21] A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo, "An evaluation of anomaly detection and diagnosis in multivariate time series," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 6, pp. 2508–2517, Jun. 2022.

[22] N. Bai, X. Wang, R. Han, Q. Wang, and Z. Liu, "PAFormer: Anomaly detection of time series with parallel-attention transformer," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Dec. 11, 2023, doi: 10.1109/TNNLS.2023.3337876.

[23] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[24] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.

[25] R. Miikkulainen et al., "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, pp. 1–8, Mar. 2017.

[26] K. Faber, M. Pietron, and D. Zurek, "Ensemble neuroevolution-based approach for multivariate time series anomaly detection," *Entropy*, vol. 23, no. 11, p. 1466, Nov. 2021.

[27] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009.

[28] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv:1901.03407*.

[29] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines," *IEEE Access*, vol. 9, pp. 120043–120065, 2021.

[30] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Mach. Learn. Appl.*, vol. 12, Jun. 2023, Art. no. 100470.

[31] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, "Deep learning for medical anomaly detection—A survey," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–37, 2021.

[32] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–33, Apr. 2022.

[33] R.-Q. Chen, G.-H. Shi, W.-L. Zhao, and C.-H. Liang, "A joint model for IT operation series prediction and anomaly detection," *Neurocomputing*, vol. 448, pp. 130–139, Aug. 2021.
[34] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, *arXiv:1803.01271*.
[35] C. Zhang et al., "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 1409–1416.
[36] B. Zong et al., "Deep autoencoding Gaussian mixture model for unsupervised anomaly detection," in *Proc. ICLR*, 2018, pp. 1–19.
[37] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Trans. Artif. Intell.*, vol. 2, no. 6, pp. 476–493, Dec. 2021.
[38] D. V. Vargas and J. Murata, "Spectrum-diverse neuroevolution with unified neural models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 8, pp. 1759–1773, Aug. 2017.
[39] M.-K. Jiau and S.-C. Huang, "Self-organizing neuroevolution for solving carpool service problem with dynamic capacity to alternate matches," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 4, pp. 1048–1060, Apr. 2019.
[40] Y. Huang, G. G. Yen, and V. S. Tseng, "Snippet policy network v2: Knee-guided neuroevolution for multi-lead ECG early classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 2, pp. 2167–2181, Feb. 2024.
[41] D. Dimanov, E. Balaguer-Ballester, C. Singleton, and S. Rostami, "MONCAE: Multi-objective neuroevolution of convolutional autoencoders," in *Proc. ICLR*, 2021, pp. 1–8.
[42] H. Okada, "Neuroevolution of autoencoders by genetic algorithm," *Int. J. Sci. Eng. Invest.*, vol. 6, no. 6, pp. 127–131, 2017.
[43] Y. Zhou, L. Liao, Y. Gao, R. Wang, and H. Huang, "TopicBERT: A topic-enhanced neural language model fine-tuned for sentiment classification," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 6, 2021, doi: 10.1109/TNNLS.2021.3094987.
[44] Y. Ro, J. Choi, B. Heo, and J. Y. Choi, "Rollback ensemble with multiple local minima in fine-tuning deep learning networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 3, 2021, doi: 10.1109/TNNLS.2021.3059669.
[45] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. ICML*, 2020, pp. 7197–7206.
[46] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–10.
[47] H. Xu et al., "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in *Proc. World Wide Web Conf. World Wide Web*, 2018, pp. 187–196.
[48] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 1–9.
[49] M. R. Ackermann, J. Blömer, D. Kuntze, and C. Sohler, "Analysis of agglomerative clustering," *Algorithmica*, vol. 69, no. 1, pp. 184–215, 2014.
[50] A. P. Mathur and N. O. Tippenhauer, "SWaT: A water treatment testbed for research and training on ICS security," in *Proc. Int. Workshop Cyber-Phys. Syst. Smart Water Netw. (CySWater)*, Apr. 2016, pp. 31–36.
[51] C. Ahmed, V. Palleti, and A. Mathur, "WADI: A water distribution testbed for research in the design of secure cyber physical systems," in *Proc. 3rd Int. Workshop Cyber-Phys. Syst. Smart Water Netw.*, Apr. 2017, pp. 25–28.
[52] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *Proc. AAAI Conf.*, 2022, pp. 7194–7201.

Marcin Pietroń received the Ph.D. degree in computer science from the AGH University of Krakow, Kraków, Poland, in 2014.

He is currently an Assistant Professor with the AGH University of Krakow. He works as an AI Scientific Consultant for many high-tech companies. He has authored about 80 research articles. He was involved as a reviewer for several international journals and conferences. His research concentrates on artificial intelligence and parallel computing.

Dominik Żurek received the Ph.D. degree from the AGH University of Krakow, Kraków, Poland, in 2021.

He is currently a Research Specialist at the AGH University of Krakow. He cooperates with many companies as an AI expert. His research interests include deep learning, computational intelligence, and general purpose graphics processing units (GPGPU) programming.

Kamil Faber received the B.S. and M.S. degrees from the AGH University of Krakow, Kraków, Poland, in 2018 and 2019, respectively, where he is currently pursuing the Ph.D. degree in computer science.

He was a Research Intern and later a Research Assistant in a DARPA-funded Project at American University, Washington, DC, USA, from 2021 to 2022. His research interests cover lifelong learning, anomaly detection, and machine learning applications in cybersecurity.

Roberto Corizzo (Member, IEEE) received the Ph.D. degree from the University of Bari Aldo Moro, Bari, Italy, in 2018.

He is currently an Assistant Professor with the Department of Computer Science, American University, Washington, DC, USA. He has co-authored over 60 articles, including 20 publications in journals such as IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, *Neural Networks*, and *Machine Learning*. He was involved in the scientific committee of international conferences and served as a reviewer for several international journals.