MDPI

Article

Introducing a parallel Particle Swarm Method for neural network training

Ioannis G. Tsoulos[1,*], Vasileios Charilogis[2]

[1] Department of Informatics and Telecommunications, University of Ioannina, Greece;itsoulos@uoi.gr
[2] Department of Informatics and Telecommunications, University of Ioannina, Greece; v.charilog@uoi.gr
* Correspondence: itsoulos@uoi.gr

Abstract

Artificial neural networks constitute a machine learning tool that has been used across a wide range of scientific as well as commercial applications over the past decades, delivering excellent results in most cases. For the effective training of the parameters of these machine learning models, optimization techniques are employed; however, in most cases these techniques tend to require significant computational time, which is also determined by the complexity and the size of the data to which the artificial neural network is applied. In the present work, a computational method is presented that exploits modern parallel computing architectures and is based on a distributed variant of the Particle Swarm Optimization technique. This method also employs distributed initialization of the artificial neural network parameters to enable more effective exploration of the parameter value space of the machine learning model. Furthermore, experimental results indicate that the proposed method becomes more effective as the number of parallel computational units increases, thereby enhancing the overall efficiency of the technique.In this study, the technique was applied to a large dataset originating from various scientific domains, and the experimental results were more than promising.

Keywords: Neural networks; machine learning; particle swarm optimization; parallel programming

1. Introduction

Artificial neural networks (ANNs) are parametric machine learning models [1,2] that have been extensively employed over the past decades in a wide range of real-world applications. These applications span multiple scientific domains, including physics-related problems [3–5], chemistry [6–8], medicine [9,10], and economics [11–13], among others. Moreover, in the recent years there have been appeared applications of the neural networks to problems such as solutions of differential equations [14,15], problems appeared in agricultural [16,17], image processing [18], wind speed prediction [19], consumption issues [20], network security [21] etc. Typically, a neural network is defined as a function $N(\vec{x}, \vec{w})$, where the vector $\vec{x}$ denotes the input pattern and the vector $\vec{w}$ represents the weight vector, that should be estimated by some optimization procedure. The estimation of this vector is commonly achieved by minimizing the so-called training error, defined mathematically as :

$$E(\vec{w}) = \sum_{i=1}^{M} \left(N(\vec{x}_i, \vec{w}) - y_i\right)^2 \quad (1)$$

Received:
Revised:
Accepted:
Published:

The set $(\overrightarrow{x_i}, y_i)$, $i = 1, ..., M$ represents the associated training set of the current problem. The values y_i denote the desired output for each pattern $\overrightarrow{x_i}$. As proposed in [22], a neural network can be formulated as the following function:

$$N(\overrightarrow{x}, \overrightarrow{w}) = \sum_{i=1}^{H} w_{(d+2)i-(d+1)} \sigma \left(\sum_{j=1}^{d} x_j w_{(d+2)i-(d+1)+j} + w_{(d+2)i} \right) \quad (2)$$

In this equation, the constant number H denotes the number of used processing units. Moreover, the function $\sigma(x)$ is the sigmoid function, expressed as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (3)$$

From equation 2 is deduced that the total number of parameters for neural network is $n = (d + 2)H$. Also, a a series of alternative activation functions has been proposed, such as the tanh function, which is formulated as:

$$\tanh(x) = \frac{e^{2x} + 1}{e^{2x} - 1} \quad (4)$$

Also, Guarnieri et al introduced a new activation function under the name adaptive spline function in a recent publication [23]. Similarly, Ertuğrul introduced a new trained activation function [24] to be used in neural networks. A recent review on activation functions for neural networks is proposed in the work of Rasamoelina et al [25].

The minimization of equation 1 has been tackled in the past by various techniques, such as the Back Propagation method [26,27], the RPROP optimization method [28–30], Quasi Newton methods [32,33], the Simulated Annealing approach [34,35], Genetic Algorithms [36,37], Particle Swarm Optimization [38,39], Differential Optimization methods [40], Evolutionary Computation [41], the Whale optimization algorithm [42], the Butterfly optimization algorithm [43], etc. In a similar vein, Sexton et al. introduced the application of the tabu search algorithm to the training of neural networks [44]. Furthermore, Zhang et al. proposed a hybrid approach that combines the Particle Swarm Optimization (PSO) method with the Backpropagation algorithm [45]. Zhao et al. presented a Cascaded Forward Algorithm aimed at the optimal training of artificial neural networks [46]. In addition, the widespread use of parallel computing techniques in recent years has motivated numerous studies investigating the training of artificial neural networks through such approaches [47,48]. Likewise, a substantial body of research has focused on the development of effective strategies for initializing the parameters of artificial neural networks. Representative methods include initialization schemes based on decision trees [49], approaches derived from Cauchy's inequality [50], the use of genetic algorithms [51], and techniques founded on discriminant learning principles [52], among others.

A major problem encountered in artificial neural network training techniques is the increased completion time of these methods, which directly depends on the number of available patterns in the training set and the number of features in each data set. A series of publications have appeared in the recent literature that propose parallel techniques for optimizing the parameters of the artificial neural network. For example, Lotrič et al proposed a parallel implementation of neural networks using the MPI library [53] in their work [54]. Likewise, Gonzalez et al proposed a parallel implementation of an evolutionary approach that utilizes the MPI method, for the evolution of artificial neural networks [55]. Additionally, Gu et all introduced the cNeural [56], a parallel computing platform to train neural networks using the Back Propagation method. A survey on the parallelization

of neural network using various parallel techniques can be found in the recent work of Chanthini et al [57].

In this paper, the use of a parallel optimization technique for the parameters of the artificial neural network is proposed, which is based on the widely used Particle Swarm Optimization method [58]. In the proposed optimization method, the initial population of particles is divided into small populations and in each subpopulation a modified version of the PSO technique is executed. Each population is evolved on a separate processing thread. In this modified version, a stochastic technique for calculating the inertia coefficient ω is used, as well as a termination technique based on stochastic observations. The individual populations exchange the best individuals with each other after a predetermined number of iterations to accelerate the evolutionary process. The Particle Swarm Optimization method was chosen in this work due to the small number of parameters required but also for its efficiency and speed in finding the global minimum of functions.

The rest of this manuscript is organized as follows: the used dataset and the incorporated methods used in the conducted experiments are outlined in section 2, the experimental results are shown and discussed in section 3 and finally a detailed discussion is provided in section 4.

2. Materials and Methods

This section provides a detailed description of the Particle Swarm Optimization method and the proposed modifications. The overall parallel algorithm is also presented.

2.1. The PSO method

The Particle Swarm Optimization (PSO) method originates from the observations made by Eberhart and Kennedy during the 1990s. By closely examining the collective behavior of birds searching for food, they developed a global optimization technique that draws on insights gained from these natural phenomena. In the proposed framework, particles move within the search space of the problem with the objective of locating the minimum of a given objective function. Each particle is characterized by two fundamental attributes: its current position and its velocity of movement. The current position is denoted by $\overrightarrow{x}$, while the velocity is represented by $\overrightarrow{u}$. Moreover, each particle retains in memory the best position it has previously encountered—corresponding to the lowest value of the objective function—as well as the best position identified by the entire swarm. The optimization process proceeds iteratively, with particle positions at each iteration being updated based on their current positions, their personal best positions, and the global best position of the population.

Owing to its conceptual simplicity and the limited number of parameters that need to be tuned, this method has been successfully applied to a wide range of challenging problems across various scientific disciplines. Such problems encompass applications arising in physics [59,60], chemistry [61,62], medicine [63,64], and economics [65], among others. In addition, the PSO method has recently been successfully employed in a variety of practical applications, including flow shop scheduling [66], the development of effective electric vehicle charging strategies [67], emotion recognition [68], and robotics [69]. A comprehensive tutorial on PSO methodologies is provided by Marini and Walczak [70].

Furthermore, the above global optimization technique has been applied with various modifications to the training of artificial neural networks, such as for example the work of Zhou et al, who applied the PSO method to train neural networks in boring machining [71]. Meissner et al proposed the Optimized Particle Swarm Optimization (OPSO) [72]method for the training of artificial neural networks. The PSO method was also applied to train neural networks to predict the outcome of construction analysis [73]. The PSO was also

used to model the global solar radiation by training artificial neural networks in the work of Mohandes [74]. Moreover, Chen at al suggested a hybrid approach that combines the PSO method the Cuckoo Search technique [75] for the optimal training of neural networks [76].

The PSO method that is executed in every processing thread is outlined as a series of steps in Algorithm 1.

Algorithm 1 The base PSO algorithm which is implemented in every processing unit.

1. **Input:**
   (a) N_m, the number of used particles.
   (b) N_g, the number of maximum allowed iterations.
2. **Initialization Step** .
   (a) **Set** $k = 0$.
   (b) **Initialize** as sets of random double numbers, the positions $p_1, p_2, ..., p_{N_m}$ for each particle. Each particle has dimension n, which is the number of parameters of the artificial neural network, and constitutes a possible combination of values for the parameters of the artificial neural network.
   (c) **Initialize** randomly the corresponding velocities $u_1, u_2, ..., u_{N_m}$.
   (d) **For** $i = 1..N_m$ do $b_i = x_i$, where each vector b_i denotes the best located position for particle i.
   (e) **Set** $p_{\text{best}} = \arg\min_{i \in 1..N_m} f(p_i)$, which represents the best particle of the population. The function $f(p)$ is the training error of the corresponding neural network $N(\overrightarrow{x}, \overrightarrow{p})$.
3. **Termination Check Step** . The termination condition of the serial algorithm is evaluated, and if it is satisfied, the algorithm concludes.
4. **For** $i = 1 \ldots N_m$ **Do**
   (a) **Compute** the corresponding velocity u_i as

$$u_i = \omega u_i + r_1 c_1 (b_i - p_i) + r_2 c_2 (p_{\text{best}} - p_i) \tag{5}$$

   where
      i. The values r_1, r_2 represent random numbers in $[0, 1]$.
      ii. The constants c_1, c_2 are usually defined in $[1, 2]$.
      iii. The variable ω represents the so - called inertia and it was initially proposed by Shi and Eberhart [58].
   (b) **Update** the current position of each particle as: $p_i = p_i + u_i$
   (c) **Calculate** the fitness $f(p_i)$ for particle p_i using the formula of the corresponding training error:

$$f(p_i) = \sum_{j=1}^{M} \left(N(\overrightarrow{x}_j, \overrightarrow{p_i}) - y_j \right)^2 \tag{6}$$

   (d) **If** $f(p_i) \leq f(b_i)$ then $b_i = x_i$
5. **End For**
6. **Set** $p_{\text{best}} = \arg\min_{i \in 1..N_m} f(p_i)$
7. **Set** $k = k + 1$.
8. **Goto** Step 3

Also, the flowchart for this algorithm is outlined graphically in Figure 1.

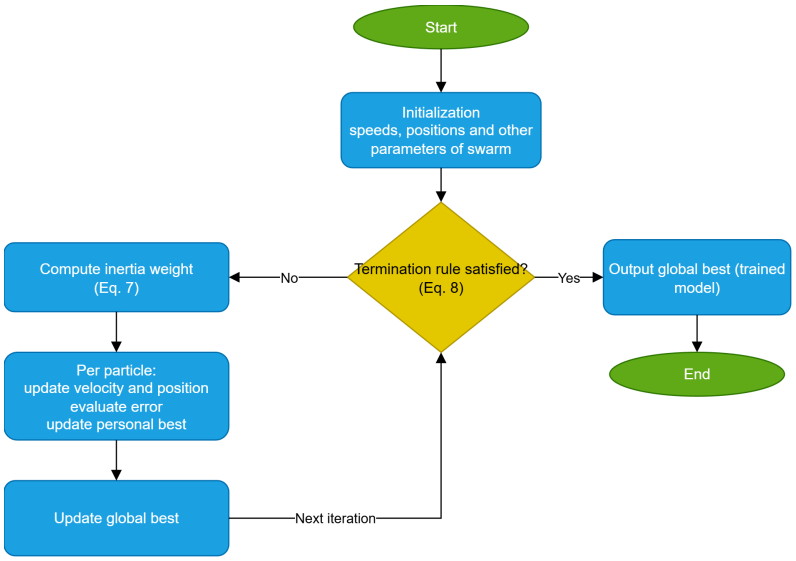

Figure 1. The flowchart of the proposed PSO variant.

2.2. Proposed modifications

The original Particle Swarm Optimization technique was extended using two modifications recently proposed by Charilogis et al [77] and aimed at more efficiently finding the global minimum of a function and at early termination of the technique. The first modification proposes a stochastic calculation for the inertia using the following formula:

$$\omega_{\mathrm{k}} = 0.5 + \frac{r}{2} \tag{7}$$

In this equation the variable k denotes the iteration number of the PSO method and the value r represents a random number in $r \in [0, 1]$. Additionally, a termination rule is proposed where the algorithm computes at every iteration k the quantity:

$$\delta^{(k)} = \left| f_{\min}^{(k)} - f_{\min}^{(k-1)} \right| \tag{8}$$

The value $f_{\min}^{(k)}$ represents the best function value obtained by the algorithm at at iteration k. If the aforementioned quantity remains below a predefined threshold ϵ for N_E consecutive iterations, the execution of the algorithm is terminated.

2.3. The parallel algorithm

The overall parallel algorithm, which is executed on T independent processing units, is outlined in algorithm 2. The proposed parallel algorithm divides the problem into a series of algorithms that are executed on different computing units. At regular intervals, these individual algorithms share the particles with the best value found with the rest, in order to accelerate the evolutionary process but also to keep the individual algorithms informed about the best solutions found by others.

Algorithm 2 The used parallel algorithm.

1. **Input:**
   (a) T as the number of parallel processing units.
   (b) N_R denotes the number of iterations after which each processing unit transmits its best particles to the other processing units.
   (c) N_P specifies the number of particles that are migrated among the parallel processing units.
2. **Set** $k = 0$.
3. **For** $j = 1, .., T$ do in parallel
   (a) **Execute** an iteration of serial PSO algorithm, that was described previously in algorithm 1 on the processing unit j.
   (b) **If** k **mod** $N_R = 0$,**then**
      i. **Obtain** the best N_P particles from processing unit j.
      ii. **Propagate** these N_P particles to the remaining processing units.
   (c) **EndIf**
4. **End For**
5. **Set** $k = k + 1$
6. **Check** the proposed termination rule on the processing units. If the termination rule holds on each processing unit, the goto step 7 else goto step 3.
7. **Local search step**..
   (a) **Obtain** the best particle from all processing units and denote it as p_a
   (b) **Optimize** the corresponding neural network $N(\vec{x}, \vec{p_a})$ using some local optimization method.
   (c) **Apply** the final neural network to the test set of the objective problem and report the associated test error.

The flowchart for the proposed parallel algorithm is shown in Figure 2.

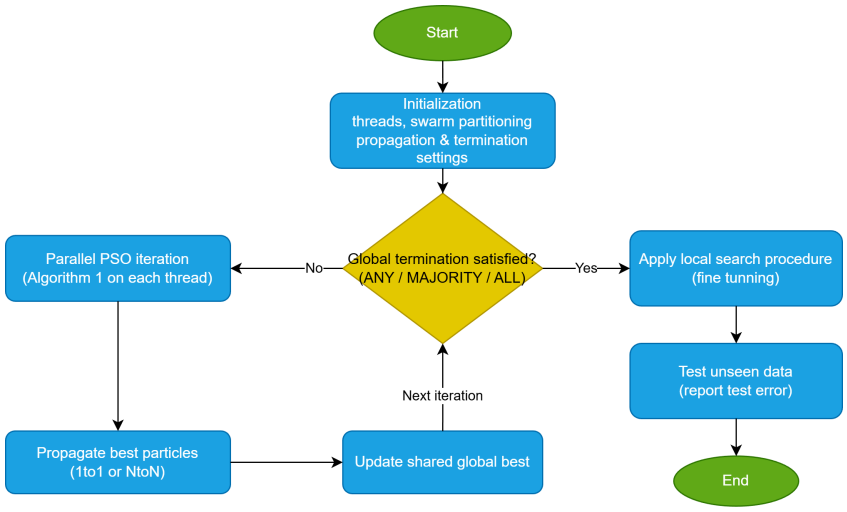

Figure 2. The flowchart of the proposed parallel algorithm.

3. Results

The proposed approach was evaluated on a collection of well-known datasets drawn from recent literature, and its performance was compared with that of other optimization methods used for training neural networks. The datasets employed in this study are publicly available and can be accessed through the following websites:

1. The UCI dataset repository, https://archive.ics.uci.edu/ml/index.php(accessed on 18 January 2026) [78].

2. The Keel repository, https://sci2s.ugr.es/keel/datasets.php(accessed on 18 January 2026) [79].

3.1. Used datasets

The following datasets were used in the conducted experiments:

1. **Alcohol** dataset, which consists of experimental data related to alcohol consumption [80].
2. **Australian** dataset [81], which is related to credit card transactions.
3. **Balance** dataset [82], used in a series of psychological experiments.
4. **Beed** dataset, that was initially presented in the work of Banu [83] and contains various EEG measurements.
5. **Cleveland** dataset, which is considered as a medical dataset [84,85].
6. **Dermatology** dataset [86], which is also medical dataset regarding dermatological deceases.
7. **Ecoli** dataset, which is related to proteins[87].
8. **Heart** dataset [88], which is a medical dataset used in the detection of heart diseases.
9. **HeartAttack** dataset, a medical dataset related to the prediction of heart attacks.
10. **HouseVotes** dataset [89], that contains data from various votes in the U.S. House of Representatives.
11. **Liverdisorder** dataset [90], which is a medical dataset related to liver issues.
12. **Lymography** dataset [91]
13. **Magic** dataset, related to various physics experiments [92].
14. **OptDigits** dataset, which is related to bitmaps of handwritten digits [93].
15. **Parkinsons** dataset, which is related to the detection of Parkinson's disease (PD)[94].
16. **Pima** dataset [95], which is medical dataset used to detect the presence of diabetes.
17. **PirVision** dataset, which is a dataset that contains occupancy detection data [96].
18. **Popfailures** dataset [97], which contains data from climate measurements.
19. **Regions2** dataset, which is a medical dataset related to the presence of hepatitis C [98].
20. **Saheart** dataset [99], which is a medical dataset related to heart diseases.
21. **Satimage** dataset, that contains multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image.
22. **Segment** dataset [100], which is a dataset used in images processing tasks.
23. **Sonar** dataset [101], used in sonar signals.
24. **Spambase** dataset, used to detect spam emails [102].
25. **Spiral** dataset, which is an artificial dataset with two distinct classes.
26. **Statheart** dataset, which is another medical dataset related to heart diseases.
27. **Wdbc** dataset [103], which is a medical dataset related to cancer..
28. **Wine** dataset, which was used to determine the quality of wines. [104,105].
29. **Eeg** datasets, that contains various EEG measurements [106] and the following cases were utilized from this dataset: Z_F_S, ZO_NF_S and ZONF_S.
30. **Zoo** dataset [107], which was used to classify animals.

3.2. Experimental results

The code for the present study was implemented in C++, utilizing the freely available Optimus environment [108]. All experiments were conducted 30 times, each with a different random seed. To validate the experimental results, the standard ten-fold cross-validation procedure was employed. The experimental parameter values are presented in Table 1. These values were chosen to achieve a balance between the efficiency and computational

speed of the methods employed during the experiments. The following notation is used in the experimental tables:

1. The column DATASET denotes the name of the objective problem.
2. The column BFGS is used to depict the results obtained by the application of the BFGS variant of Powell [109] to train the neural network.
3. The column GENETIC is used to represent the obtained results by the application of a Genetic Algorithm with the same parameter set as provided in Table 1 to train a neural network with $H = 10$ processing nodes.
4. The column RBF denotes the results obtained by the the incorporation of a Radial Basis Function (RBF) network [110,111] with $H = 10$ hidden nodes.
5. The column PPSO $T = 1$ presents the results from the application of the proposed method with one processing thread.
6. The column PPSO $T = 2$ shows the results from the application of the proposed method with two processing threads.
7. The column PPSO $T = 4$ outlines the results for the proposed method and 4 processing threads.
8. The column PPSO $T = 10$ shows the results from the usage of the proposed method with 10 processing threads.
9. The last row AVERAGE denotes the average classification error for all datasets in the corresponding table.

Table 1. The values for the used parameters.

PARAMETER	MEANING	VALUE
N_c	Chromosomes/Particles	500
N_g	Maximum number of allowed generations	500
N_E	Value used in the termination rule	15
N_R	Propagation iterations	5
N_P	Propagated particles	3
c_1	Parameter for PSO	1.0
c_2	Parameter for PSO	1.0
H	Number of weights	10

The Table 2 reports classification error rates (%) per dataset and per method. Each cell represents the empirical frequency of misclassification for a given dataset under the adopted experimental protocol; within each row, lower values indicate better generalization. The AVERAGE row provides an aggregate view across heterogeneous datasets, effectively summarizing the expected error when selecting a dataset from this benchmark set and applying the corresponding method.

A clear global trend is that PPSO improves as the number of threads increases: 27.45% ($T = 1$) $\rightarrow$ 24.30% ($T = 2$) $\rightarrow$ 23.10% ($T = 4$) $\rightarrow$ 22.19% ($T = 10$). This is consistent with the behavior of parallel stochastic optimization, where additional threads typically enhance exploration diversity (e.g., multiple initializations and search trajectories) and reduce the probability of converging to poor local minima. The gains also exhibit diminishing returns: the step from $T = 1$ to $T = 2$ yields the largest improvement, whereas the additional benefit from $T = 4$ to $T = 10$ is smaller, suggesting that beyond a point the search either saturates the solution space adequately or becomes limited by practical factors such as noise, synchronization/communication, and the effective allocation of the evaluation budget.

Relative to the baselines, PPSO at $T = 10$ achieves a substantially lower average error: 22.19% versus 28.65% for GEN (a reduction of 6.46 percentage points, approximately a 22.5%

relative decrease), 37.38% for BFGS (approximately a 40.7% relative decrease), and 32.49% for RBF (approximately a 31.7% relative decrease). Interpreted scientifically, this indicates that the proposed approach more consistently identifies higher-quality parameterizations (e.g., network weights) that translate into improved predictive performance across a broad and diverse benchmark suite. The especially large gap versus BFGS is aligned with the non-convex, multi-modal nature of neural training objectives, where gradient-based local methods can become trapped, while population-based stochastic strategies retain greater capacity to escape suboptimal basins.

At the dataset level, two main patterns emerge. First, there are datasets where PPSO delivers very large improvements (e.g., DERMATOLOGY, WDBC, WINE, ALCOHOL, SEGMENT, OPTDIGITS), suggesting that optimization difficulty is a primary bottleneck and that enhanced distributed exploration yields markedly better solutions. Second, there are datasets where differences are small or not strictly monotone with thread count, and in a few cases a baseline is marginally competitive (e.g., SATIMAGE for RBF, or datasets where GEN/RBF already achieve very low error). Such cases typically reflect performance saturation on easier tasks, stronger inductive bias of a particular model family, or stochastic variability and hyperparameter interactions that can produce minor reversals (e.g., T=4 slightly better than T=10 on some rows) without contradicting the overall downward trend.

A key interpretive point is that the Table 2 captures not only "optimization strength" but also the interaction between the optimizer, the underlying modeling assumptions, and the way computational budget is distributed. If the total number of evaluations (or total time) is held constant across thread counts, the improvement with larger T indicates a genuine efficiency gain from distributed exploration. If the total budget increases with T, then the observed gains also reflect increased computational effort. In either case, the results demonstrate that PPSO is competitive across a wide range of datasets and that increased parallelism shifts the error profile toward consistently lower values, which is the intended outcome of a parallel stochastic training procedure.

Based on Table 2, statistical processing was carried out via R scripts in order to quantify the significance of performance differences between the proposed PPSO method and the baseline approaches (BFGS, GEN, RBF) across different thread counts. Figure 3 summarizes the resulting significance levels using the critical p-value criterion, following the standard categories ns ($p > 0.05$), significant ($p < 0.05$), highly significant ($p < 0.01$), extremely significant ($p < 0.001$), and very extremely significant ($p < 0.0001$). The global comparison across all methods is overwhelmingly significant: the overall Friedman test yields $p = 2.2 \times 10^{-16}$, which strongly rejects the null hypothesis of equivalent performance and confirms that genuine differences exist among the evaluated methods over the full dataset collection.

At the level of pairwise comparisons by thread count, a clear strengthening of statistical evidence is observed as T increases. For $T = 1$, PPSO does not differ significantly from BFGS ($p = 0.6371$), GEN ($p = 1$), or RBF ($p = 1$), indicating that with a single thread the observed performance differences cannot be distinguished from dataset-to-dataset variability under the adopted experimental conditions. For $T = 2$, the situation changes: PPSO shows a highly robust advantage over BFGS ($p = 1.01 \times 10^{-6}$), whereas comparisons against GEN ($p = 0.2124$) and RBF ($p = 0.1312$) remain non-significant. For $T = 4$, the evidence further intensifies: against BFGS the p-value is 1.37×10^{-10} (very extremely significant), while PPSO also becomes statistically superior to GEN ($p = 0.0035$, highly/extremely significant) and to RBF ($p = 9.57 \times 10^{-4}$, extremely significant). Finally, for $T = 10$, PPSO exhibits very strong statistical superiority against all three baselines, with $p = 2.46 \times 10^{-13}$ versus BFGS, $p = 1.33 \times 10^{-5}$ versus GEN, and $p = 1.97 \times 10^{-6}$ versus RBF, corresponding to extremely to very extremely significant differences.

Overall, these findings indicate that increasing the number of threads not only improves PPSO's average accuracy but also turns this improvement into a consistent and statistically defensible advantage over established baselines across a broad suite of classification datasets. The lack of significance at $T = 1$ suggests that, under limited parallelism, the stochastic gains are not sufficiently strong to dominate inter-dataset variability. In contrast, for $T \geq 2$ PPSO first establishes a clear advantage over BFGS, and as T increases to 4 and 10 this advantage extends to GEN and RBF as well, supporting the conclusion that distributed search substantially enhances PPSO's ability to locate higher-quality training solutions and achieve better generalization.

Table 2. Experiments with a variety of machine learning methods on the used datasets.

DATASET	BFGS	GEN	RBF	PPSO $T = 1$	PPSO $T = 2$	PPSO $T = 4$	PPSO $T = 10$
ALCOHOL	41.50%	39.57%	49.38%	20.73%	18.25%	16.53%	16.99%
AUSTRALIAN	38.13%	32.21%	34.89%	32.62%	30.76%	29.61%	26.13%
BALANCE	8.64%	8.97%	33.53%	8.32%	7.53%	7.97%	8.13%
BEED	60.73%	58.94%	65.12%	42.63%	41.10%	41.73%	40.44%
CLEVELAND	77.55%	51.60%	67.10%	58.14%	48.66%	47.41%	47.07%
DERMATOLOGY	52.92%	30.58%	62.34%	11.66%	9.91%	9.52%	9.49%
ECOLI	69.52%	54.67%	59.48%	56.76%	53.76%	50.61%	47.24%
GLASS	60.48%	55.00%	50.05%	56.52%	54.48%	51.53%	51.53%
HABERMAN	29.34%	28.66%	25.10%	28.60%	29.90%	30.67%	30.27%
HEART	39.44%	28.34%	31.20%	33.30%	27.07%	22.11%	20.37%
HEARTATTACK	46.67%	29.03%	29.00%	30.57%	24.93%	21.60%	21.04%
HOUSEVOTES	7.13%	6.62%	6.13%	7.22%	7.35%	6.52%	7.31%
IONOSPHERE	15.29%	15.14%	16.22%	15.54%	15.34%	14.49%	14.34%
LIVERDISORDER	42.59%	31.11%	30.84%	39.18%	35.12%	33.03%	33.47%
LYMOGRAPHY	35.43%	28.42%	25.50%	26.93%	26.72%	28.36%	27.50%
MAGIC	17.30%	21.75%	21.28%	15.58%	13.98%	14.15%	13.87%
MAMMOGRAPHIC	17.24%	19.88%	21.38%	18.75%	16.88%	17.17%	16.89%
OPTDIGITS	67.63%	64.96%	81.65%	51.02%	48.17%	46.41%	41.97%
PARKINSONS	27.58%	18.05%	17.41%	19.89%	17.68%	17.00%	15.32%
PIMA	35.59%	32.19%	25.78%	33.54%	32.14%	30.49%	29.47%
PIRVISION	16.94%	9.50%	24.58%	9.64%	7.49%	6.26%	7.72%
POPFAILURES	5.24%	5.94%	7.04%	7.65%	7.72%	7.09%	7.08%
REGIONS2	36.28%	29.39%	38.29%	29.92%	28.47%	27.70%	27.65%
SAHEART	37.48%	34.86%	32.19%	35.09%	34.61%	34.11%	34.41%
SATIMAGE	71.73%	63.50%	40.19%	55.21%	45.60%	48.06%	40.84%
SEGMENT	68.97%	57.72%	59.68%	32.83%	27.70%	21.49%	24.34%
SONAR	25.85%	22.40%	27.90%	22.90%	21.30%	21.40%	20.85%
SPAMBASE	18.16%	6.37%	29.35%	7.67%	6.52%	6.34%	6.39%
SPIRAL	47.99%	48.66%	44.87%	46.18%	44.48%	42.65%	42.81%
STATHEART	39.65%	27.25%	31.36%	32.48%	26.56%	22.22%	21.15%
TRANSFUSION	25.84%	24.87%	26.41%	25.11%	24.37%	25.23%	24.12%
WDBC	29.91%	8.56%	7.27%	7.96%	7.16%	6.11%	5.36%
WINE	59.71%	19.20%	31.41%	27.94%	21.00%	16.53%	13.00%
Z_F_S	39.37%	10.73%	13.16%	26.73%	15.33%	11.90%	10.47%
ZO_NF_S	43.04%	21.54%	9.02%	29.38%	12.38%	11.44%	7.52%
ZONF_S	15.62%	4.36%	4.03%	3.80%	2.46%	2.94%	2.60%
ZOO	10.70%	9.50%	21.93%	7.60%	6.20%	6.30%	5.70%
AVERAGE	**37.38%**	**28.65%**	**32.49%**	**27.45%**	**24.30%**	**23.10%**	**22.19%**

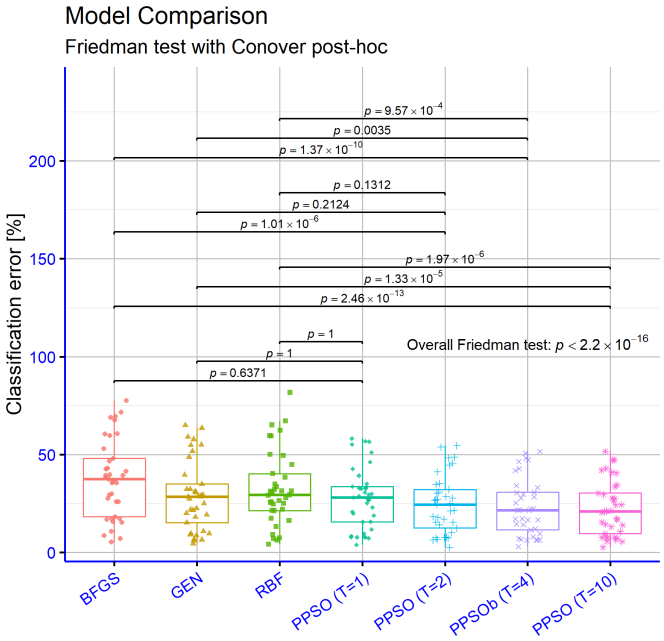

Figure 3. Statistical comparison for the obtained experimental results using the series of the machine learning methods.

3.3. Experiments with the propagation method

Additionally, another experiment was conducted where two different techniques for propagating the best particles between processing threads were used. On the one hand, the present technique, which is referred to in the experiment as NxN, and on the other hand, another propagation technique, entitled $1x1$, where a random processing thread is selected after a predetermined number of iterations to send its best particles to another randomly selected processing thread. The experimental results for the cases of four processing threads ($T = 4$) and ten processing threads ($T = 10$) are presented in Table 3.

The Table 3 reports classification error rates (%) for the proposed model trained/optimized with PPSO under four configurations that combine two factors: the level of parallelism ($T = 4$ vs $T = 10$ threads) and the information-propagation scheme among parallel workers (1to1 vs NtoN). Each cell is an empirical estimate of generalization for a given dataset: lower percentages indicate fewer misclassifications and therefore better predictive performance. Consequently, each row can be interpreted as a localized study of how parallel search capacity and communication intensity influence the quality of the final trained solution on that specific classification task.

The AVERAGE row provides an aggregated view over a heterogeneous benchmark suite. In this Table 3, $T = 10$ with 1to1 propagation yields the best mean error (22.11%), followed by $T = 4$ with 1to1 (22.96%), while the NtoN variants exhibit higher average error (24.30% for $T = 4$, NtoN and 23.10% for $T = 10$, NtoN). This ordering does not imply that 1to1 is universally superior to NtoN, but rather that, across this collection of datasets, the more restrained propagation scheme more frequently leads to better end solutions. From a distributed optimization perspective, this is consistent with a well-known trade-off: stronger propagation (NtoN) can accelerate convergence by rapidly homogenizing information across workers, but it can also reduce search diversity and increase the risk of premature convergence to suboptimal regions. In contrast, more selective propagation (1to1) preserves greater independence across threads, maintains alternative promising trajectories for longer, and often improves the final generalization outcome.

The effect of increasing the number of threads is not uniform across datasets, which is expected. Moving from $T = 4$ to $T = 10$ typically increases coverage of the solution space by enabling more independent search trajectories and a richer set of initializations. This tends to be particularly beneficial when the optimization landscape is highly multi-modal or when training is sensitive to initialization. In the Table 3, there are clear cases where $T = 10$ substantially improves performance, such as OPTDIGITS (38.30% vs 48.13% for 1to1), WINE (13.00% vs 16.53% for 1to1), ZO_NF_S (7.52% vs 11.44% for 1to1), and STATHEART (20.87% vs 22.22% for 1to1). These patterns indicate that additional parallel exploration more reliably discovers parameterizations that generalize better. At the same time, there are datasets where the difference between $T = 10$ and $T = 4$ is small or slightly reversed (e.g., ALCOHOL, BEED, SONAR). Such behavior typically reflects either performance saturation ($T = 4$ already explores sufficiently), or stochastic variability and configuration-dependent synchronization/communication effects that produce different but not consistently better solutions on that dataset.

The comparison between 1to1 and NtoN is also dataset-dependent. In many rows, 1to1 has a clear advantage, for example HEART (20.37% vs 22.11% at $T = 10$), OPTDIGITS (38.30% vs 46.41% at $T = 10$), and WDBC (5.55% vs 6.11% at $T = 10$). These cases are consistent with the interpretation that NtoN can drive all threads too quickly toward a shared basin of attraction that is not optimal for generalization, especially when multiple competing parameter regions yield similar training quality but different test behavior. However, there are also datasets where NtoN appears beneficial, such as SEGMENT at $T = 10$ (21.49% vs 23.85% for 1to1) and PIRVISION (6.26% at $T = 10$, NtoN vs 7.72% at $T = 10$, 1to1), suggesting that, for certain tasks, faster collective exploitation of a strong solution region can be advantageous when the landscape is less deceptive or when the best region is strongly attracting.

Importantly, the fact that no single configuration dominates every dataset is scientifically meaningful and expected. Datasets differ in noise level, dimensionality and feature correlations, nonlinearity, and class separability, all of which shape the error landscape being optimized. As a result, parallelism and propagation act as regulators of the exploration–exploitation balance and can steer PPSO toward solutions with better average generalization, while individual datasets may favor different balances. Overall, the Table 3 demonstrates that thread count and propagation strategy are not merely implementation details but materially affect the generalization quality of the proposed model. The best mean performance of $T = 10$ with 1to1 propagation suggests that, for this benchmark suite, higher parallelism is most effective when coupled with controlled information exchange that preserves diversity and mitigates premature convergence, whereas NtoN can still be competitive on specific datasets where rapid collective exploitation is advantageous.

Figure 4 reports the significance levels of performance differences among PPSO variants with respect to the propagation mechanism (1to1 versus NtoN) under two parallelism settings ($T = 4$ and $T = 10$). The interpretation relies on the critical p-value and the standard significance categories ns ($p > 0.05$), significant ($p < 0.05$), highly significant ($p < 0.01$), extremely significant ($p < 0.001$), and very extremely significant ($p < 0.0001$). The overall multi-method comparison indicates that meaningful differences exist across the evaluated PPSO configurations, as the global Friedman test yields 4.28×10^{-6}, thereby rejecting the null hypothesis of equal performance among variants over the full suite of classification datasets.

At the pairwise level, the comparison between PPSO ($T = 4$, 1to1) and PPSO ($T = 4$, NtoN) produces $p = 0.0236$, which is statistically significant under the conventional $p < 0.05$ criterion. This result implies that, at a moderate parallelism level (4 threads), the choice of propagation scheme systematically affects the performance of the proposed

model across datasets, and the observed differences are unlikely to be explained solely by random dataset-to-dataset variability. In contrast, for $T = 10$ the comparison between PPSO ($T = 10$, 1to1) and PPSO ($T = 10$, NtoN) yields $p = 0.0567$, which lies slightly above the 0.05 threshold and is therefore classified as non-significant (ns) under that cutoff. This near-threshold value suggests a tendency for differentiation between the two propagation schemes at 10 threads, but the available empirical evidence is not strong enough to support the same level of statistical confidence as in the $T = 4$ case.

Overall, these findings support the conclusion that the propagation mechanism can materially influence the performance of the proposed PPSO training scheme, particularly at moderate parallelism ($T = 4$), whereas at higher parallelism ($T = 10$) the differences appear weaker or less stable across the heterogeneous dataset collection. From an interpretive standpoint, this is consistent with the notion that increasing the number of threads already enhances search diversity and may partially absorb the effect of propagation, reducing the net separability between 1to1 and NtoN when performance is aggregated across many diverse classification problems.

Table 3. Experiments using the proposed method and different propagation techniques.

	$1to1$		$NtoN$	
DATASET	PPSO $T = 4$	PPSO $T = 10$	PPSO $T = 4$	PPSO $T = 10$
ALCOHOL	16.34%	17.36%	18.25%	16.53%
AUSTRALIAN	29.61%	26.13%	30.76%	29.61%
BALANCE	7.97%	8.13%	7.53%	7.97%
BEED	39.07%	40.59%	41.10%	41.73%
CLEVELAND	47.42%	47.07%	48.66%	47.41%
DERMATOLOGY	9.52%	9.49%	9.91%	9.52%
ECOLI	50.61%	47.24%	53.76%	50.61%
GLASS	51.53%	51.49%	54.48%	51.53%
HABERMAN	30.67%	30.27%	29.90%	30.67%
HEART	22.11%	20.37%	27.07%	22.11%
HEARTATTACK	21.60%	22.04%	24.93%	21.60%
HOUSEVOTES	6.52%	7.31%	7.35%	6.52%
IONOSPHERE	14.49%	14.34%	15.34%	14.49%
LIVERDISORDER	33.03%	33.47%	35.12%	33.03%
LYMOGRAPHY	28.36%	27.50%	26.72%	28.36%
MAGIC	14.06%	13.72%	13.98%	14.15%
MAMMOGRAPHIC	17.17%	16.89%	16.88%	17.17%
OPTDIGITS	48.13%	38.30%	48.17%	46.41%
PARKINSONS	17.00%	15.32%	17.68%	17.00%
PIMA	30.59%	29.47%	32.14%	30.49%
PIRVISION	6.56%	7.72%	7.49%	6.26%
POPFAILURES	7.09%	7.12%	7.72%	7.09%
REGIONS2	27.70%	27.65%	28.47%	27.70%
SAHEART	34.11%	34.41%	34.61%	34.11%
SATIMAGE	44.17%	40.10%	45.60%	48.06%
SEGMENT	21.70%	23.85%	27.70%	21.49%
SONAR	21.40%	20.85%	21.30%	21.40%
SPAMBASE	6.15%	6.28%	6.52%	6.34%
SPIRAL	42.68%	42.81%	44.48%	42.65%
STATHEART	22.22%	20.87%	26.56%	22.22%
TRANSFUSION	25.23%	24.12%	24.37%	25.23%
WDBC	5.95%	5.55%	7.16%	6.11%
WINE	16.53%	13.00%	21.00%	16.53%
Z_F_S	11.90%	10.47%	15.33%	11.90%
ZO_NF_S	11.44%	7.52%	12.38%	11.44%
ZONF_S	2.94%	2.62%	2.46%	2.94%
ZOO	5.80%	6.50%	6.20%	6.30%
AVERAGE	**22.96%**	**22.11%**	**24.30%**	**23.10%**

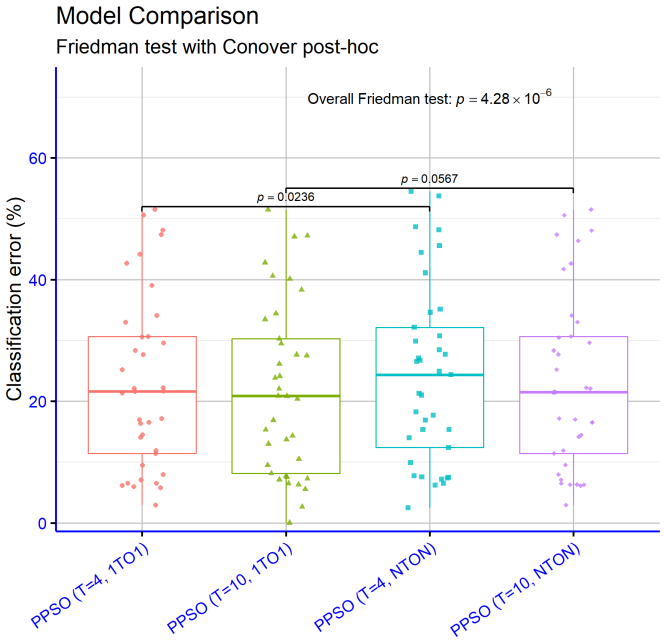

Figure 4. Statistical comparison for the results by the application of the proposed method and the propagation techniques.

3.4. Experiments with the termination strategy

Furthermore, to determine the effectiveness of the proposed technique in relation to the termination strategy, another experiment was conducted in which the number of processing threads was set to $T = 10$. In this experiment, the following termination strategies were used, as reflected in the relevant table with the results (Table 4):

1. ANY. In this case, the method terminates when the termination rule is applied to one of the processing threads.
2. MAJORITY. In this case, the optimization method terminates when the predefined termination rule applies to the majority of processing threads.
3. ALL. In this case, termination is successful when the termination rule applies to all processing threads.

The Table 4 compares three termination policies applied to the same proposed PPSO-based training/optimization procedure across a suite of classification datasets, using classification error (%) as the evaluation criterion. The columns ANY, MAJORITY, and ALL do not correspond to different models; instead, they implement different rules for deciding when the parallel procedure is considered "finished" and which outcome is accepted as final. Therefore, each cell reflects the generalization performance of the same underlying model when only the termination/synchronization logic is changed. This is important because it highlights that, even with a fixed optimizer, the decision policy can materially influence which solution is ultimately selected, effectively controlling how conservatively or aggressively the algorithm commits to a final set of parameters.

The AVERAGE row indicates that the ALL policy achieves the lowest mean error (22.19%), followed by MAJORITY (22.74%) and ANY (22.99%). Although the differences are moderate, the ordering is consistent with a clear interpretation: ALL is the strictest rule, requiring completion (or convergence) of all parallel workers before finalizing the result. This tends to favor higher-quality solutions because it reduces the chance of stopping due to a single "lucky" or prematurely stabilized worker and allows slower but potentially more informative search trajectories to contribute. In contrast, ANY is the most aggressive rule, since the procedure can terminate as soon as one worker finishes, which can more often

lead to premature commitment to solutions that are not optimal in terms of generalization. MAJORITY lies between these extremes by requiring agreement or completion from most workers, partially balancing decision speed with outcome stability.

At the dataset level, there is no universal dominance of a single policy in every row, yet ALL shows clear advantages on several challenging or highly variable tasks. For example, ALL produces noticeably lower errors on OPTDIGITS (41.97% versus 42.96% and 46.06%), SATIMAGE (40.84% versus 45.78% and 43.34%), SEGMENT (24.34% versus 25.68% and 25.94%), SONAR (20.85% versus 22.45% and 22.30%), and WDBC (5.36% versus 5.79% and 6.55%). This pattern aligns with the idea that, on difficult optimization landscapes where different workers may converge to substantially different solutions, waiting for all workers reduces the probability of selecting an inferior solution that simply appeared earlier. On the other hand, there are datasets where MAJORITY or ANY is marginally better (e.g., BEED and LYMOGRAPHY for MAJORITY, or ALCOHOL where MAJORITY slightly outperforms ALL). These cases suggest that the added strictness of ALL does not always translate into improved generalization, particularly when the incremental benefit from the "late" workers is small or when extended search increases the risk of overfitting on certain datasets.

Conceptually, the three policies can be viewed as mechanisms that regulate the exploration–commitment trade-off in parallel stochastic search. ANY emphasizes fast commitment: if one worker quickly reaches a good region, the process stops early, which can be computationally efficient but more sensitive to random favorable or unfavorable outcomes. MAJORITY provides greater robustness by requiring collective confirmation from more workers, acting as a filter against isolated extreme behaviors. ALL is the most conservative and, according to the mean performance in the Table 4, tends to yield a more reliable selection of high-quality solutions because it fully exploits parallelism until completion. The observed improvement in the overall average suggests that, for this benchmark suite, the additional "patience" imposed by ALL more frequently translates into better generalization, especially on problems where ignoring slower workers means losing valuable alternative solutions.

Overall, the Table 4 demonstrates that the termination policy is not a minor implementation detail but a mechanism that directly affects which solution is chosen by a parallel stochastic training procedure. Based on both the average values and the behavior on several difficult datasets, ALL appears to provide the most consistently high-quality outcomes, while ANY and MAJORITY can be viewed as alternatives that prioritize faster decisions or an intermediate balance, respectively, at the cost of slightly higher mean classification error.

Figure 5 presents the significance levels of performance differences obtained by comparing three alternative termination policies of the same proposed PPSO procedure (ANY, MAJORITY, ALL) over a suite of classification datasets. Significance is interpreted via the p-value using the standard categories ns for $p > 0.05$, significant for $p < 0.05$, highly significant for $p < 0.01$, extremely significant for $p < 0.001$, and very extremely significant for $p < 0.0001$. The three policies differ in when the parallel training process is considered complete and the final solution is accepted: ANY terminates when a single thread finishes, MAJORITY terminates when a predefined number or fraction of threads finish, and ALL terminates only when all threads finish. The analysis therefore tests whether the strictness of the termination rule systematically alters the final performance of the same model across different datasets.

The overall Friedman test yields $p = 0.0031$, which is highly significant ($p < 0.01$). This indicates that, when all three termination policies are considered jointly, the observed performance differences across the dataset suite are sufficiently consistent to reject the null hypothesis of equal performance. In other words, the termination policy is not a negligible

implementation detail but a factor that can meaningfully influence the final classification error achieved by the proposed procedure.

At the same time, the reported pairwise comparisons do not reach statistical significance under the conventional 0.05 threshold. Specifically, PPSO (ANY) versus PPSO (MAJORITY) yields $p = 0.1211$ (non-significant), and PPSO (MAJORITY) versus PPSO (ALL) yields $p = 0.5477$ (also non-significant). This apparent discrepancy between a significant global test and non-significant pairwise contrasts is common in multi-method, heterogeneous dataset evaluations: a global test can detect that systematic differences exist somewhere among the methods, while individual pairwise tests may lack sufficient power to cross the same significance threshold, especially when differences are modest, unevenly distributed across datasets, and/or when multiple-comparison adjustments are applied.

Interpretively, these findings suggest that the termination rule affects PPSO's overall behavior as a family of configurations, yet the specific pairwise contrasts reported here do not provide strong evidence of a uniform advantage of one policy over another across the full dataset suite. This aligns with the conceptual role of termination policies as regulators of the "fast commitment" versus "full parallel exploitation" trade-off: ANY emphasizes rapid completion with a higher risk of premature solution selection, ALL is more conservative and may stabilize the final choice by fully utilizing parallel search, and MAJORITY lies between these extremes. The significant overall Friedman result supports that this regulation matters at the aggregate level, even if the presented pairwise p-values are not sufficient, on their own, to establish a definitive ordering between those particular policy pairs under $p < 0.05$.

Table 4. Experiments with different termination strategy and the proposed method. The number of threads was set to $T = 10$

DATASET	ANY	MAJORITY	ALL
ALCOHOL	19.40%	16.97%	16.99%
AUSTRALIAN	25.28%	26.26%	26.13%
BALANCE	7.55%	8.02%	8.13%
BEED	41.28%	39.87%	40.44%
CLEVELAND	49.03%	48.66%	47.07%
DERMATOLOGY	10.31%	10.03%	9.49%
ECOLI	48.60%	48.52%	47.24%
GLASS	54.24%	54.05%	51.53%
HABERMAN	29.57%	29.97%	30.27%
HEART	21.74%	21.56%	20.37%
HEARTATTACK	21.94%	22.17%	21.04%
HOUSEVOTES	7.04%	6.91%	7.31%
IONOSPHERE	15.14%	14.86%	14.34%
LIVERDISORDER	31.38%	32.00%	33.47%
LYMOGRAPHY	28.79%	26.50%	27.50%
MAGIC	13.62%	13.56%	13.87%
MAMMOGRAPHIC	17.24%	16.89%	16.89%
OPTDIGITS	46.06%	42.96%	41.97%
PARKINSONS	15.63%	15.26%	15.32%
PIMA	29.66%	29.40%	29.47%
PIRVISION	9.46%	8.20%	7.72%
POPFAILURES	7.13%	6.70%	7.08%
REGIONS2	27.63%	27.61%	27.65%
SAHEART	33.83%	34.72%	34.41%
SATIMAGE	43.34%	45.78%	40.84%
SEGMENT	25.94%	25.68%	24.34%
SONAR	22.30%	22.45%	20.85%
SPAMBASE	6.40%	6.09%	6.39%
SPIRAL	42.33%	42.50%	42.81%
STATHEART	22.04%	22.00%	21.15%
TRANSFUSION	24.49%	24.26%	24.12%
WDBC	6.55%	5.79%	5.36%
WINE	17.71%	17.82%	13.00%
Z_F_S	10.70%	10.67%	10.47%
ZO_NF_S	8.02%	7.06%	7.52%
ZONF_S	2.84%	2.88%	2.60%
ZOO	6.30%	6.90%	5.70%
AVERAGE	**22.99%**	**22.74%**	**22.19%**

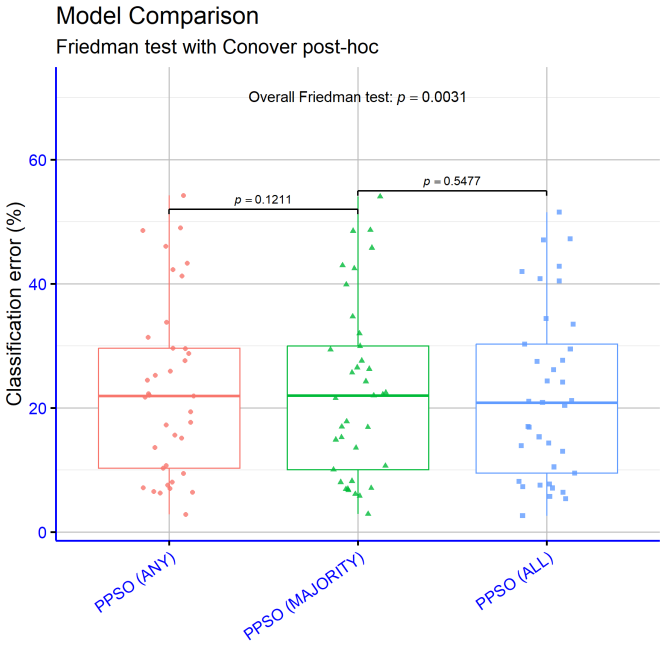

Figure 5. Statistical comparison for the results produced by the usage of different termination strategies.

4. Conclusions

This work showed that training the proposed machine learning model can be substantially improved when formulated as a stochastic optimization task and implemented through a parallel swarm execution. The classical PSO core is used with two explicit modifications, inertia computed according to Equation 7 and termination controlled by Equation 8, so that the search dynamics become more controlled and less prone to premature stagnation. The parallel formulation then orchestrates multiple independent search trajectories and empirically demonstrates that increasing the number of threads acts as a reliability mechanism, because it increases the probability of locating better minima on the error landscape.

Across a broad collection of classification datasets, the results indicate a consistent reduction in classification error as parallelism increases. The average error decreases from 27.45% with one thread to 22.19% with ten threads, reflecting a clear generalization gain and a diminishing returns pattern, since most benefits are already captured up to four threads and additional improvements become smaller thereafter. Relative to the reference methods, the best PPSO configuration achieves markedly lower mean error, corresponding to an approximately 40.6% reduction versus BFGS, approximately 22.5% versus GEN, and approximately 31.7% versus RBF, indicating that the advantage is not confined to isolated datasets but also holds at the aggregate level.

The statistical evaluation further supports that these differences are not random. The overall Friedman test yields $p = 2.2 \times 10^{-16}$, strongly rejecting performance equivalence. Moreover, the pairwise comparisons show that increasing thread count strengthens the consistency of the PPSO advantage against the baselines, with extremely small p values at higher parallelism levels. The propagation analysis suggests that the information exchange scheme can materially influence performance at moderate parallelism, where the difference between one to one and N to N is significant for four threads, while for ten threads it becomes borderline and less stable. Similarly, the termination policy analysis indicates that termination affects the overall behavior with a significant global effect, yet without

a universal dominance in every pairwise contrast, which is consistent with termination acting mainly as a regulator of when and how the procedure commits to a final solution.

A direct next step is to more strictly disentangle the effect of parallelism from the total computational budget, clarifying how much of the gain stems from improved exploration and how much from increased computational effort. In this context, systematic reporting of runtime, energy cost, and scalability is essential, together with the study of asynchronous communication protocols that reduce synchronization overhead and enable smoother exploitation of parallel resources.

In addition, propagation and termination can be upgraded to adaptive strategies that dynamically select one to one or N to N propagation and ANY, MAJORITY, or ALL termination based on stagnation indicators, solution diversity, and improvement rate. There is also clear potential for hybrid schemes that combine PPSO with local optimization, as well as for multiobjective training where, beyond error, complexity, stability, and interpretability are optimized. Finally, evaluation should be expanded to larger and more imbalanced datasets, online learning settings, and distribution shift scenarios, in order to substantiate robustness under real conditions and nonstationary data streams.

Author Contributions: V.C. and I.G.T. conducted the experiments, employing several datasets and provided the comparative experiments. V.C. performed the statistical analysis and prepared the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been financed by the European Union : Next Generation EU through the Program Greece 2.0 National Recovery and Resilience Plan , under the call RESEARCH – CREATE – INNOVATE, project name “iCREW: Intelligent small craft simulator for advanced crew training using Virtual Reality techniques" (project code:TAEDK-06195).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
2. G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control Signals and Systems **2**, pp. 303-314, 1989.
3. P. Baldi, K. Cranmer, T. Faucett et al, Parameterized neural networks for high-energy physics, Eur. Phys. J. C **76**, 2016.
4. J. J. Valdas and G. Bonham-Carter, Time dependent neural network models for detecting changes of state in complex processes: Applications in earth sciences and astronomy, Neural Networks **19**, pp. 196-207, 2006
5. G. Carleo,M. Troyer, Solving the quantum many-body problem with artificial neural networks, Science **355**, pp. 602-606, 2017.
6. Lin Shen, Jingheng Wu, and Weitao Yang, Multiscale Quantum Mechanics/Molecular Mechanics Simulations with Neural Networks, Journal of Chemical Theory and Computation **12**, pp. 4934-4946, 2016.
7. Sergei Manzhos, Richard Dawes, Tucker Carrington, Neural network-based approaches for building high dimensional and quantum dynamics-friendly potential energy surfaces, Int. J. Quantum Chem. **115**, pp. 1012-1020, 2015.
8. Jennifer N. Wei, David Duvenaud, and Alán Aspuru-Guzik, Neural Networks for the Prediction of Organic Chemistry Reactions, ACS Central Science **2**, pp. 725-732, 2016.
9. Igor I. Baskin, David Winkler and Igor V. Tetko, A renaissance of neural networks in drug discovery, Expert Opinion on Drug Discovery **11**, pp. 785-795, 2016.
10. Ronadl Bartzatt, Prediction of Novel Anti-Ebola Virus Compounds Utilizing Artificial Neural Network (ANN), Chemistry Faculty Publications **49**, pp. 16-34, 2018.
11. Lukas Falat and Lucia Pancikova, Quantitative Modelling in Economics with Advanced Artificial Neural Networks, Procedia Economics and Finance **34**, pp. 194-201, 2015.

12. Mohammad Namazi, Ahmad Shokrolahi, Mohammad Sadeghzadeh Maharluie, Detecting and ranking cash flow risk factors via artificial neural networks technique, Journal of Business Research **69**, pp. 1801-1806, 2016.
13. G. Tkacz, Neural network forecasting of Canadian GDP growth, International Journal of Forecasting **17**, pp. 57-69, 2001.
14. Y. Shirvany, M. Hayati, R. Moradian, Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations, Applied Soft Computing **9**, pp. 20-29, 2009.
15. A. Malek, R. Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network—Optimization method, Applied Mathematics and Computation **183**, pp. 260-271, 2006.
16. A. Topuz, Predicting moisture content of agricultural products using artificial neural networks, Advances in Engineering Software **41**, pp. 464-470, 2010.
17. A. Escamilla-García, G.M. Soto-Zarazúa, M. Toledano-Ayala, E. Rivas-Araiza, A. Gastélum-Barrios, Abraham,Applications of Artificial Neural Networks in Greenhouse Technology and Overview for Smart Agriculture Development, Applied Sciences **10**, Article number 3835, 2020.
18. H. Boughrara, M. Chtourou, C. Ben Amar et al, Facial expression recognition based on a mlp neural network using constructive training algorithm. Multimed Tools Appl **75**, pp. 709–731, 2016.
19. H. Liu, H.Q Tian, Y.F. Li, L. Zhang, Comparison of four Adaboost algorithm based artificial neural networks in wind speed predictions, Energy Conversion and Management **92**, pp. 67-81, 2015.
20. J. Szoplik, Forecasting of natural gas consumption with artificial neural networks, Energy **85**, pp. 208-220, 2015.
21. H. Bahram, N.J. Navimipour, Intrusion detection for cloud computing using neural networks and artificial bee colony optimization algorithm, ICT Express 5, pp. 56-59, 2019.
22. I.G. Tsoulos, D. Gavrilis, E. Glavas, Neural network construction and training using grammatical evolution, Neurocomputing 72, pp. 269-277, 2008.
23. S. Guarnieri, F. Piazza, A. Uncini, Multilayer feedforward networks with adaptive spline activation function, IEEE Transactions on Neural Networks **10**, pp. 672-683, 1999.
24. Ö.F. Ertuğrul, A novel type of activation function in artificial neural networks: Trained activation function, Neural Networks **99**, pp. 148-157, 2018.
25. A. D. Rasamoelina, F. Adjailia, P. Sinčák, A Review of Activation Function for Artificial Neural Network, In: 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), Herlany, Slovakia, pp. 281-286, 2020.
26. D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by back-propagating errors, Nature **323**, pp. 533 - 536 , 1986.
27. T. Chen and S. Zhong, Privacy-Preserving Backpropagation Neural Network Learning, IEEE Transactions on Neural Networks **20**, , pp. 1554-1564, 2009.
28. M. Riedmiller and H. Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm, Proc. of the IEEE Intl. Conf. on Neural Networks, San Francisco, CA, pp. 586–591, 1993.
29. T. Pajchrowski, K. Zawirski and K. Nowopolski, Neural Speed Controller Trained Online by Means of Modified RPROP Algorithm, IEEE Transactions on Industrial Informatics **11**, pp. 560-568, 2015.
30. Rinda Parama Satya Hermanto, Suharjito, Diana, Ariadi Nugroho, Waiting-Time Estimation in Bank Customer Queues using RPROP Neural Networks, Procedia Computer Science **135**, pp. 35-42, 2018.
31. Neural Networks, Procedia Computer Science **135**, pp. 35-42, 2018.
32. B. Robitaille and B. Marcos and M. Veillette and G. Payre, Modified quasi-Newton methods for training neural networks, Computers & Chemical Engineering **20**, pp. 1133-1140, 1996.
33. Q. Liu, J. Liu, R. Sang, J. Li, T. Zhang and Q. Zhang, Fast Neural Network Training on FPGA Using Quasi-Newton Optimization Method,IEEE Transactions on Very Large Scale Integration (VLSI) Systems **26**, pp. 1575-1579, 2018.
34. A. Yamazaki, M. C. P. de Souto,T. B. Ludermir, Optimization of neural network weights and architectures for odor recognition using simulated annealing, In: Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 **1**, pp. 547-552 , 2002.
35. Y. Da, G. Xiurun, An improved PSO-based ANN with simulated annealing technique, Neurocomputing **63**, pp. 527-533, 2005.
36. F. H. F. Leung, H. K. Lam, S. H. Ling and P. K. S. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, IEEE Transactions on Neural Networks **14**, pp. 79-88, 2003
37. X. Yao, Evolving artificial neural networks, Proceedings of the IEEE, 87(9), pp. 1423-1447, 1999.
38. C. Zhang, H. Shao and Y. Li, Particle swarm optimisation for evolving artificial neural network, IEEE International Conference on Systems, Man, and Cybernetics, , pp. 2487-2490, 2000.
39. Jianbo Yu, Shijin Wang, Lifeng Xi, Evolving artificial neural networks using an improved PSO and DPSO **71**, pp. 1054-1060, 2008.
40. J. Ilonen, J.K. Kamarainen, J. Lampinen, Differential Evolution Training Algorithm for Feed-Forward Neural Networks, Neural Processing Letters **17**, pp. 93–105, 2003.

41. M. Rocha, P. Cortez, J. Neves, Evolution of neural networks for classification and regression, Neurocomputing **70**, pp. 2809-2816, 2007.
42. I. Aljarah, H. Faris, S. Mirjalili, Optimizing connection weights in neural networks using the whale optimization algorithm, Soft Comput **22**, pp. 1–15, 2018.
43. S.M.J. Jalali, S. Ahmadian, P.M. Kebria, A. Khosravi, C.P. Lim, S. Nahavandi, Evolving Artificial Neural Networks Using Butterfly Optimization Algorithm for Data Classification. In: Gedeon, T., Wong, K., Lee, M. (eds) Neural Information Processing. ICONIP 2019. Lecture Notes in Computer Science(), vol 11953. Springer, Cham, 2019.
44. R.S. Sexton, B. Alidaee, R.E. Dorsey, J.D. Johnson, Global optimization for artificial neural networks: A tabu search application. European Journal of Operational Research **106**, pp. 570-584, 1998.
45. J.-R. Zhang, J. Zhang, T.-M. Lok, M.R. Lyu, A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training, Applied Mathematics and Computation **185**, pp. 1026-1037, 2007.
46. G. Zhao, T. Wang, Y. Jin, C. Lang, Y. Li, H. Ling, The Cascaded Forward algorithm for neural network training, Pattern Recognition **161**, 111292, 2025.
47. K-Su Oh, K. Jung, GPU implementation of neural networks, Pattern Recognition **37**, pp. 1311-1314, 2004.
48. M. Zhang, K. Hibi, J. Inoue, GPU-accelerated artificial neural network potential for molecular dynamics simulation, Computer Physics Communications **285**, 108655, 2023.
49. I. Ivanova, M. Kubat, Initialization of neural networks by means of decision trees, Knowledge-Based Systems **8**, pp. 333-344, 1995.
50. J.Y.F Yam, T.W.S. Chow, A weight initialization method for improving training speed in feedforward neural network, Neurocomputing **30**, pp. 219-232, 2000.
51. F. Itano, M. A. de Abreu de Sousa, E. Del-Moral-Hernandez, Extending MLP ANN hyper-parameters Optimization by using Genetic Algorithm, In: 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, pp. 1-8, 2018.
52. K. Chumachenko, A. Iosifidis, M. Gabbouj, Feedforward neural networks initialization based on discriminant learning, Neural Networks **146**, pp. 220-229, 2022.
53. Gropp, W., Lusk, E., Skjellum, A. (1999) Using MPI: portable parallel programming with the message-passing interface, MIT, Cambridge.
54. Lotrič, U., Dobnikar, A. (2005). Parallel implementations of feed-forward neural network using MPI and C# on .NET platform. In: Ribeiro, B., Albrecht, R.F., Dobnikar, A., Pearson, D.W., Steele, N.C. (eds) Adaptive and Natural Computing Algorithms. Springer, Vienna.
55. B. P. Gonzalez, G. G. Sánchez, J. P. Donate, P. Cortez and A. S. de Miguel, "Parallelization of an evolving Artificial Neural Networks system to Forecast Time Series using OPENMP and MPI," 2012 IEEE Conference on Evolving and Adaptive Intelligent Systems, Madrid, Spain, 2012, pp. 186-191.
56. R. Gu, F. Shen and Y. Huang, "A parallel computing platform for training large scale neural networks," 2013 IEEE International Conference on Big Data, Silicon Valley, CA, USA, 2013, pp. 376-384.
57. Chanthini, P., & Shyamala, K. (2016). A survey on parallelization of neural network using MPI and Open MP. Indian Journal of Science and Technology, 9(19).
58. J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.
59. Anderson Alvarenga de Moura Meneses, Marcelo Dornellas, Machado Roberto Schirru, Particle Swarm Optimization applied to the nuclear reload problem of a Pressurized Water Reactor, Progress in Nuclear Energy **51**, pp. 319-326, 2009.
60. Ranjit Shaw, Shalivahan Srivastava, Particle swarm optimization: A new tool to invert geophysical data, Geophysics **72**, 2007.
61. C. O. Ourique, E.C. Biscaia, J.C. Pinto, The use of particle swarm optimization for dynamical analysis in chemical processes, Computers & Chemical Engineering **26**, pp. 1783-1793, 2002.
62. H. Fang, J. Zhou, Z. Wang et al, Hybrid method integrating machine learning and particle swarm optimization for smart chemical process operations, Front. Chem. Sci. Eng. **16**, pp. 274–287, 2022.
63. M.P. Wachowiak, R. Smolikova, Yufeng Zheng, J.M. Zurada, A.S. Elmaghraby, An approach to multimodal biomedical image registration utilizing particle swarm optimization, IEEE Transactions on Evolutionary Computation **8**, pp. 289-301, 2004.
64. Yannis Marinakis. Magdalene Marinaki, Georgios Dounias, Particle swarm optimization for pap-smear diagnosis, Expert Systems with Applications **35**, pp. 1645-1656, 2008.
65. Jong-Bae Park, Yun-Won Jeong, Joong-Rin Shin, Kwang Y. Lee, An Improved Particle Swarm Optimization for Nonconvex Economic Dispatch Problems, IEEE Transactions on Power Systems **25**, pp. 156-16**216**6, 2010.
66. B. Liu, L. Wang, Y.H. Jin, An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **37**, pp. 18-27, 2007.
67. J. Yang, L. He, S. Fu, An improved PSO-based charging strategy of electric vehicles in electrical distribution grid, Applied Energy **128**, pp. 82-92, 2014.

68. K. Mistry, L. Zhang, S. C. Neoh, C. P. Lim, B. Fielding, A Micro-GA Embedded PSO Feature Selection Approach to Intelligent Facial Emotion Recognition, IEEE Transactions on Cybernetics. **47**, pp. 1496-1509, 2017.
69. S. Han, X. Shan, J. Fu, W. Xu, H. Mi, Industrial robot trajectory planning based on improved pso algorithm, J. Phys.: Conf. Ser. **1820**, 012185, 2021.
70. F. Marini, B. Walczak, Particle swarm optimization (PSO). A tutorial, Chemometrics and Intelligent Laboratory Systems **149**, pp. 153-165, 2015.
71. Zhou, J., Duan, Z., Li, Y., Deng, J., & Yu, D. (2006). PSO-based neural network optimization and its utilization in a boring machine. Journal of Materials Processing Technology, 178(1-3), 19-23.
72. Meissner, M., Schmuker, M. & Schneider, G. Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training. BMC Bioinformatics 7, 125 (2006).
73. Chau, K. W. (2007). Application of a PSO-based neural network in analysis of outcomes of construction claims. Automation in construction, 16(5), 642-646.
74. Mohandes, M. A. (2012). Modeling global solar radiation using Particle Swarm Optimization (PSO). Solar Energy, 86(11), 3137-3145.
75. Yang, X.S.; Deb, S. Cuckoo search via Lévy flights. In Proceedings of the IEEE World Congress on Nature and Biologically Inspired Computing (NaBIC 2009), Coimbatore, India, 9–11 December 2009; pp. 210–214.
76. Chen, J. F., Do, Q. H., & Hsieh, H. N. (2015). Training artificial neural networks by a hybrid PSO-CS algorithm. Algorithms, 8(2), 292-308.
77. V. Charilogis, I.G. Tsoulos, Toward an Ideal Particle Swarm Optimizer for Multidimensional Functions, Information **13**, 217, 2022.
78. M. Kelly, R. Longjohn, K. Nottingham, The UCI Machine Learning Repository. 2023. Available online: https://archive.ics.uci.edu (accessed on 18 February 2024).
79. J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17, pp. 255-287, 2011.
80. Tzimourta, K.D.; Tsoulos, I.; Bilero, I.T.; Tzallas, A.T.; Tsipouras, M.G.; Giannakeas, N. Direct Assessment of Alcohol Consumption in Mental State Using Brain Computer Interfaces and Grammatical Evolution. Inventions 2018, 3, 51.
81. J.R. Quinlan, Simplifying Decision Trees. International Journal of Man-Machine Studies **27**, pp. 221-234, 1987.
82. T. Shultz, D. Mareschal, W. Schmidt, Modeling Cognitive Development on Balance Scale Phenomena, Machine Learning **16**, pp. 59-88, 1994.
83. Banu PK, N. (2024). Feature Engineering for Epileptic Seizure Classification Using SeqBoostNet. International Journal of Computing and Digital Systems, 16(1), 1-14.
84. Z.H. Zhou,Y. Jiang, NeC4.5: neural ensemble based C4.5," in IEEE Transactions on Knowledge and Data Engineering **16**, pp. 770-773, 2004.
85. R. Setiono , W.K. Leow, FERNN: An Algorithm for Fast Extraction of Rules from Neural Networks, Applied Intelligence **12**, pp. 15-25, 2000.
86. G. Demiroz, H.A. Govenir, N. Ilter, Learning Differential Diagnosis of Eryhemato-Squamous Diseases using Voting Feature Intervals, Artificial Intelligence in Medicine. **13**, pp. 147–165, 1998.
87. P. Horton, K.Nakai, A Probabilistic Classification System for Predicting the Cellular Localization Sites of Proteins, In: Proceedings of International Conference on Intelligent Systems for Molecular Biology **4**, pp. 109-15, 1996.
88. I. Kononenko, E. Šimec, M. Robnik-Šikonja, Overcoming the Myopia of Inductive Learning Algorithms with RELIEFF, Applied Intelligence **7**, pp. 39–55, 1997
89. R.M. French, N. Chater, Using noise to compute error surfaces in connectionist networks: a novel means of reducing catastrophic forgetting, Neural Comput. **14**, pp. 1755-1769, 2002.
90. J. Garcke, M. Griebel, Classification with sparse grids using simplicial basis functions, Intell. Data Anal. **6**, pp. 483-502, 2002.
91. G. Cestnik, I. Konenenko, I. Bratko, Assistant-86: A Knowledge-Elicitation Tool for Sophisticated Users. In: Bratko, I. and Lavrac, N., Eds., Progress in Machine Learning, Sigma Press, Wilmslow, pp. 31-45, 1987.
92. Heck, D., Knapp, J., Capdevielle, J. N., Schatz, G., & Thouw, T. (1998). CORSIKA: A Monte Carlo code to simulate extensive air showers. Report fzka, 6019(11).
93. Kaynak, C. (1995). Methods of combining multiple classifiers and their applications to handwritten digit recognition. Unpublished master's thesis, Bogazici University.
94. M.A. Little, P.E. McSharry, E.J. Hunter, J. Spielman, L.O. Ramig, Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. IEEE Trans Biomed Eng. **56**, pp. 1015-1022, 2009.
95. J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, R.S. Johannes, Using the ADAP learning algorithm to forecast the onset of diabetes mellitus, In: Proceedings of the Symposium on Computer Applications and Medical Care IEEE Computer Society Press, pp.261-265, 1988.

96. Emad-Ud-Din, M., & Wang, Y. (2023). Promoting occupancy detection accuracy using on-device lifelong learning. IEEE Sensors Journal, 23(9), 9595-9606.
97. D.D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, Y. Zhang, Failure analysis of parameter-induced simulation crashes in climate models, Geoscientific Model Development **6**, pp. 1157-1171, 2013.
98. N. Giannakeas, M.G. Tsipouras, A.T. Tzallas, K. Kyriakidi, Z.E. Tsianou, P. Manousou, A. Hall, E.C. Karvounis, V. Tsianos, E. Tsianos, A clustering based method for collagen proportional area extraction in liver biopsy images (2015) Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, 2015-November, art. no. 7319047, pp. 3097-3100.
99. T. Hastie, R. Tibshirani, Non-parametric logistic and proportional odds regression, JRSS-C (Applied Statistics) **36**, pp. 260–276, 1987.
100. M. Dash, H. Liu, P. Scheuermann, K. L. Tan, Fast hierarchical clustering and its validation, Data & Knowledge Engineering **44**, pp 109–138, 2003.
101. R.P. Gorman, T.J. Sejnowski, Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets, Neural Networks **1**, pp. 75-89, 1988.
102. Cranor, Lorrie F., LaMacchia, Brian A. Spam!, Communications of the ACM, 41(8):74-83, 1998.
103. W.H. Wolberg, O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proc Natl Acad Sci U S A. **87**, pp. 9193–9196, 1990.
104. M. Raymer, T.E. Doom, L.A. Kuhn, W.F. Punch, Knowledge discovery in medical and biological datasets using a hybrid Bayes classifier/evolutionary algorithm. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society, **33** , pp. 802-813, 2003.
105. P. Zhong, M. Fukushima, Regularized nonsmooth Newton method for multi-class support vector machines, Optimization Methods and Software **22**, pp. 225-236, 2007.
106. R.G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state, Phys. Rev. E **64**, pp. 1-8, 2001.
107. M. Koivisto, K. Sood, Exact Bayesian Structure Discovery in Bayesian Networks, The Journal of Machine Learning Research **5**, pp. 549–573, 2004.
108. Tsoulos, I.G.; Charilogis, V.; Kyrou, G.; Stavrou, V.N.; Tzallas, A. OPTIMUS: A Multidimensional Global Optimization Package. J. Open Source Softw. 2025, 10, 7584.
109. M.J.D Powell, A Tolerant Algorithm for Linearly Constrained Optimization Calculations, Mathematical Programming **45**, pp. 547-566, 1989.
110. J. Park and I. W. Sandberg, Universal Approximation Using Radial-Basis-Function Networks, Neural Computation **3**, pp. 246-257, 1991.
111. G.A. Montazer, D. Giveki, M. Karami, H. Rastegar, Radial basis function neural networks: A review. Comput. Rev. J **1**, pp. 52-74, 2018.