Journal of open research software

ExPSO-DL: An Exponential Particle Swarm Optimization Package for Deep Learning Model Optimization

SOFTWARE METAPAPER

INSAF KRAIDIA
KHELIL KASSOUL
NAOUFEL CHEIKHROUHOU
SAIMA HASSAN
SAMIR BRAHIM BELHAOUARI

*Author affiliations can be found in the back matter of this article

ubiquity press

CORRESPONDING AUTHORS:
Insaf Kraidia
Faculty of Information Technology, Department of Software Engineering, Al-Ahliyya Amman University, Amman, 19328, Jordan
i.kraidia@ammanu.edu.jo

Khelil Kassoul
A School of Business Administration University of Applied Sciences Western Switzerland HES-SO, 1227 Geneva, Switzerland
khelil.kassoul@hesge.ch

Samir Brahim Belhaouari
Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha P.O. Box 5825, Qatar
sbelhaouari@hbku.edu.qa

KEYWORDS:
Particle Swarm Optimization; Python; Machine Learning; Deep Learning; Optimization Techniques; Heuristic Algorithms

TO CITE THIS ARTICLE:
Kraidia I, Kassoul K, Cheikhrouhou N, Hassan S, Belhaouari SB 2025 ExPSO-DL: An Exponential Particle Swarm Optimization Package for Deep Learning Model Optimization. *Journal of Open Research Software*, 13: 27. DOI: https://doi.org/10.5334/jors.521

ABSTRACT

This paper presents ExPSO, a Python package designed to simplify parameter selection in deep learning models. ExPSO utilizes the Exponential Particle Swarm Optimization (ExPSO) method for global optimization problems, which has a superior ability to balance exploration and exploitation in search spaces. This package provides a user-friendly framework that promises to enhance the performance and evaluation of various deep learning algorithms through its exponential selection technique. In addition to its primary features, ExPSO is designed with extensibility in mind. It serves as a robust foundation for the development of innovative selection methodologies and can be easily adapted to incorporate other optimization algorithms and techniques. This flexibility ensures ExPSO remains relevant and useful as new advancements in the field of optimization and deep learning emerge.

(1) OVERVIEW

INTRODUCTION

The use of machine learning, including deep learning, algorithms has expanded substantially across numerous domains [1, 2, 3]. These algorithms are used for various tasks, including classification, regression, anomaly detection, and segmentation [4]. Training these algorithms depends on parameters such as the learning rate and the number of epochs. Selecting the right parameters is vital. It accelerates convergence, enhances generalization, prevents overfitting, and improves robustness. This step significantly impacts the model's ability to effectively address real-world problems [5].

To address the challenge of selecting optimal parameters, researchers propose several optimization methods, including swarm-based selection algorithms inspired by social behavior. The complexity of the selection problem varies significantly based on the attributes and properties of the objective function. Among these methods, Particle Swarm Optimization (PSO) stands out as a highly durable and efficient swarm-based selection algorithm, widely appreciated for its simplicity and versatility in various scenarios and applications [6, 7, 8]. PSO consistently delivers high performance in multi-objective optimization [9], constraint optimization [10], and global optimization [11].

Xia, et al [12] introduced Fitness-based Multi-role PSO (FMPSO), an extension that adds a sub-social learning aspect. This enables the swarm to employ different search strategies by utilizing both local and global information. Another work presented Expanded Particle Swarm Optimization (XPSO) [13], which is based on biological and human society's approaches to discarding unused data. XPSO incorporates this into standard PSO by utilizing global best and local best as examples and assigning each particle a forgetting ability. This improves the algorithm's adaptability and efficiency [14]. Phasor Particle Swarm Optimization (PPSO) [15] is a new adaptation. It replaces control parameters with a scalar phasor angle derived from trigonometric functions, offering a new approach to enhance PSO.

However, achieving convergence is not always guaranteed, and there is a risk of encountering premature convergence. To address this issue, the authors in [16] introduce Exponential Particle Swarm Optimization (ExPSO), a novel variation of the standard PSO. ExPSO incorporates a leaping strategy based on dynamic parameters. It divides the swarm population into three sub-populations and employs a search strategy using an exponential function, enabling particles to make significant leaps in the search space. Additionally, it adjusts the control of each particle's velocity range to balance exploration and exploitation phases. ExPSO is designed to take large jumps at the start of the search, followed by smaller jumps for refining solutions in specific regions of the search space [17, 18].

The selection method in ExPSO lacks comprehensive details for its re-implementation, creating a gap in transferring the developed method into a usable function for other research groups. Additionally, designing a package that excels in multiple criteria, demonstrates fast convergence, thoroughly explores the search space, and accommodates various types of objective functions presents a significant challenge. Meeting these criteria requires careful consideration and innovation. Table 1 shows a comparative summary of key features across ExPSO and four established PSO-based algorithms (XPSO, PPSO, FMPSO, and TAPSO). The table highlights the distinctive capabilities of ExPSO, including its exponential

FEATURE	ExPSO	XPSO	PPSO	FMPSO	TAPSO
Exploration-Exploitation Balance	Dynamic via exponential control	Fixed coefficient decay	Phasor-based control	Fuzzy logic-based mutation	Time-adaptive parameter tuning
Velocity Control	Adaptive exponential velocity bounds	×	Scalar phasor-based updates	Mutation-induced adjustments	Time-based inertia weight updates
Leaping Strategy	Exponential leaping function	×	×	×	×
Personal/Global Best/ Worst Usage	Full use of *pbest*, *gbest*, *pworst*	with forgetting mechanism	✓	✓	✓
Adaptivity	High (dynamic γ and w updates)	Medium (decay memory)	Medium (phasor response)	Medium (fuzzy decisions)	High (time-adaptive parameters)
DL/ML Framework Integration	Supports PyTorch/ TensorFlow APIs	×	×	×	×
Parallel Evaluation Support	Parallel training support	×	×	×	×
Code Extensibility	Modular Python classes	×	×	×	×

Table 1 Feature Comparison Between ExPSO and Selected PSO Variants.

leaping strategy, adaptive subpopulation dynamics, and integration with deep learning frameworks, which collectively enhance its suitability for complex optimization and hyperparameter tuning tasks.

In this paper, we present a Python package that extends the Exponential Particle Swarm Optimization (ExPSO) algorithm with a robust, accessible implementation tailored for deep learning model optimization. This implementation surpasses the original by incorporating enhanced usability features, including a modular API, automatic neural network parameter tuning, and integration with machine learning libraries such as TensorFlow [19] and PyTorch [20]. We further provide deployment-oriented benefits, such as parallelized evaluation for faster convergence and customizable termination criteria for real-world applications. These improvements make ExPSO more practical for researchers and practitioners seeking effective and reliable hyperparameter optimization in deep learning.

IMPLEMENTATION AND ARCHITECTURE

To enhance the balance between exploration and exploitation in swarm-based optimization, we introduce a novel variant of Particle Swarm Optimization (PSO). Our approach is designed to adaptively regulate particle movement using exponential selection pressures and velocity control (see Figure 1), enabling the swarm to effectively escape local optima and converge toward global optima with improved stability.

The complete workflow of our method appears in Algorithm 1. This algorithm outlines the initialization, evaluation, and iterative update steps of the ExPSO optimizer. The population of particles is initialized with random positions and velocities within defined bounds. Each particle maintains a record of its personal best *pbest* and personal worst *pworst* positions. The swarm also keeps track of the global best *pgbest* and global worst *pgworst* particles.

Our design features three distinct subpopulations. Each subgroup employs its own velocity update strategy, utilizing adaptive exponential coefficients. Algorithm 2 describes this process: particles are divided into three subgroups based on their index and remain in these groups throughout the algorithm:

- The first subgroup uses a velocity equation that attracts particles to both their personal best and worst positions, as well as their global best and worst positions.
- The second subgroup excludes the global worst components, simplifying the trajectory.
- The third subgroup follows classic PSO, using only personal and global best terms.

Each velocity update employs an exponential decay term:

$$\gamma = e^{\left(\frac{1}{||pbest_i - x_i|| + \epsilon}\right)} \tag{1}$$

which increases convergence pressure as particles approach their personal best, thus refining local search. This formulation enables adaptive intensification or diversification based on particle experience.

To further manage convergence behavior and avoid premature stagnation, we incorporate a Velocity Controller described in Algorithm 3. This mechanism adaptively compresses the velocity range after a specified number of iterations by scaling the velocity bounds and updating the inertia weight, w, dynamically using:

$$w = r\left(\frac{1-w}{1+w}\right) \tag{2}$$

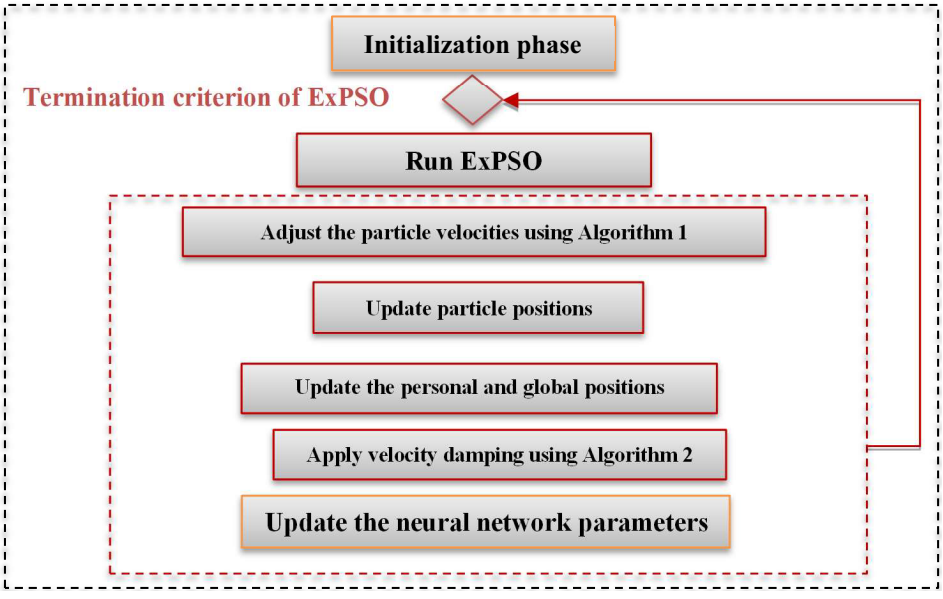

Figure 1 ExPSO approach.

Input:	- Optimization problem with n variables - Population size N - Maximum number of function evaluations $MaxFES$
Output:	Fitness value $pgbest$ of the globally best solution found
1	Set initial parameters: $a, b, c, d, e, c1, c2, r, w, N, MaxFES$
2	For each particle i in the population ($i = 1\ to\ N$):
3	Randomly initialize position x_i within search bounds
4	Randomly initialize velocity v_i
5	Evaluate fitness $f(x_i)$
6	Set personal best $pbest_i = x_i$and personal worst $pworst_i = x_i$
7	end For
8	Identify global best $pgbest$ and global worst $pgworst$ from the population
9	While stopping condition not met (e.g., FES < MaxFES):
10	For each particle i = 1 to N:
11	Update velocity v_i using ExPSO velocity equation (Algorithm 1)
12	Clip velocity: $v_i = min(max(v\_i, v\_min), v\_max)$
13	Update position: $x_i = x_i + v_i$
14	Clip position: $x_i = min(max(x_i, x\_min), x\_max)$
15	end For
16	end while
17	For each particle i:
18	Evaluate fitness $f(x_i)$
19	If f(x_i) better than $f(pbest_i)$, then update $pbest_i = x_i$
20	If f(x_i) worse than f($pworst_i$), then update $pworst_i = x_i$
21	end For
22	Update $pgbest$and $pgworst$based on the new population
23	Update dynamic parameter $a = r \times a$
24	Apply velocity damping using Algorithm 2
25	end while

Algorithm 1 ExPSO Algorithm.

This damping mechanism ensures smoother convergence by gradually reducing the exploration radius while maintaining directional diversity. The optimization loop continues until a stopping criterion is met, typically based on the maximum number of function evaluations *MaxFES*. At each iteration, the fitness of each particle is evaluated, and the personal/global best and worst records are updated accordingly.

To construct an effective model with optimal parameters, specific steps are followed. First, data are processed and split into training, validation, and test sets. The training and validation sets are used to develop the model. The test set evaluates the model's performance and generalization.

During the parameter selection phase, the user employs the ExPSO package to find the most suitable parameters. This package is built around a primary Python file that includes several PyTorch classes, such as ExPSO and Particle. These classes work together to implement the ExPSO algorithm effectively. The ExPSO class is initialized with several parameters, including the objective function, dimensions, number of particles, maximum iterations, bounds, and runs. Below, we provide a detailed description of each parameter:

- Objective function (*ObjFunction*): The function that the algorithm aims to optimize using a given set of input parameters.
- Dimensions (*D*): Refers to the shape or size of the input parameters in the objective function.
- The number of particles (*nPop*): The number of particles used to explore the solution space. A higher number of particles can improve solution quality, but it also increases computational cost.
- Maximum iteration number (*MaxIt*): Sets the maximum number of times the algorithm runs before it stops. Set this to a reasonable number to balance good solutions and to avoid long runtimes.
- Upper and lower bounds (*ub, Ib*): These are the limits within which the algorithm searches for solutions for each input parameter. Properly set bounds ensure the algorithm searches within the appropriate range.

Input: N// N is the population size, $N_1 = N_2 = N_3$ //N_1, N_2, N_3 are the number of particles in subpopulations

While (the maximum number of iterations is not reached), do

  If $i \leq N1$

    $\gamma = e^{\left(\frac{1}{\|pbest_i - x_i\| + \epsilon}\right)}$

    $v_i = wv_i + a\gamma R_{vec+} br_1(pbest_i - x_i) + cr_2(pgbest - x_i) + dr_3(pworst_i - x_i) + er_4(pgworst - x_i)$

  Else if $N1 < i \leq N1 + N2$

    $v_i = wv_i + a\gamma R_{vec+} br_1(pbest_i - x_i) + cr_2(pgbest - x_i)$

  Else

    $v_i = wv_i + a\gamma R_{vec+} c_1 r_1(pbest_i - x_i) + c_2 r_2(pgbest - x_i)$

end while

Algorithm 2 ExPSO Subpopulations Algorithm.

Input: t_v , k // t_v is the exploration number of iterations.

$v_{max=} x_{max}$

$v_{min=} x_{min}$

While (number of iterations $\geq t_v$**), do**

  $v_{max=} k x_{max}$

  $v_{min=} k x_{min}$

  $w = r\left(\frac{1-w}{1+w}\right)$

end while

Algorithm 3 Velocity Controller Algorithm.

- Number of runs (*runs*): Specifies how many times the algorithm runs with the same parameters to ensure robust and reliable results.

When the user invokes the *optimize*() method, the ExPSO algorithm starts by creating multiple combinations of parameters, which are randomly initialized. These combinations are then used in multiple iterations of the algorithm. During each iteration, the particles are converted into parameter sets and the algorithm is trained. Through iterative communication, the particles collaborate to identify the optimal solution. The objective is to discover a parameter combination that results in the most effective model. The flowchart of the optimization algorithm is detailed in [16]. Upon termination, the ExPSO algorithm extracts and stores the parameters of the best-performing model within the instantiated ExPSO object. Various factors influence the computation time of ExPSO, including the number of particles, the number of iterations, and the maximum allowed iterations. Additionally, supplementary constraints may impact the computation time by causing the optimization process to stop if no improvements are detected over a series of consecutive iterations.

The ExPSO package is designed to optimize a range of problems by treating them as a black box, making it particularly suitable for parameter selection tasks. This package supports a variable number of parameters and accommodates multiple parameter types across various algorithms. It is especially beneficial for researchers seeking to investigate the impact of various selection methods on the effectiveness of their models. The package enables users to visualize multiple metrics, providing insight into how different parameter settings affect model performance.

- *GlobalBestPosition*: Represents the optimal or near-optimal solution obtained by the ExPSO algorithm in the search space.
- *GlobalBestCost*: Represents the optimal or near-optimal value achieved by the ExPSO algorithm.
- *WorstSol*: Refers to the highest value of the best cost found during the optimization process. It helps

evaluate the quality of the obtained solutions and identify the worst performing solution.
- *MEAN*: Represents the average value of the best cost found across multiple optimization runs.
- *BestSol*: Refers to the lowest value of the best cost found during the optimization process.
- *STD* (Standard Deviation): Refers to the standard deviation, which is often used to assess the diversity or convergence of the obtained solutions. A lower standard deviation indicates that the solutions are more closely clustered and concentrated around the optimal solution.
- *AvgFES* (Average Function Evaluations): Represents the average number of function evaluations performed during the optimization process. It indicates the computational effort required to find the optimal solution.

By leveraging the flexibility of PyTorch and its ability to define new structures, users can customize the optimization process and experiment with different variations of the ExPSO algorithm.

IMPLEMENTATION

PyTorch was selected for this work due to its dynamic computation graph (eager execution), which provides more flexibility and transparency during model development, debugging, and implementation of custom optimization or attack strategies. This is particularly advantageous for research-focused tasks such as counterfactual generation, adversarial attacks, and iterative model updates. Compared to TensorFlow [19] and Keras [22], PyTorch offers more intuitive control flow and better support for low-level operations, which are essential when fine-tuning optimization processes or integrating explainability components [20]. While (Just After eXecution (JAX) [23] is also a strong candidate for high-performance computing, its functional paradigm and limited debugging support make PyTorch more suitable for iterative and experimental work. While JAX provides strong capabilities for high-performance and accelerated computing through its functional programming paradigm and XLA compilation, the limited flexibility in debugging and dynamic execution makes PyTorch a more practical and accessible framework for iterative development and experimental research workflows.

HYPERPARAMETERS

In our proposed approach, we adopted a carefully tuned set of parameters to balance exploration, exploitation, and diversity within the swarm. The exponential weight parameter (a = 2) ensures moderate sensitivity to fitness differences, allowing meaningful selection pressure without premature convergence. The cognitive and social acceleration coefficients (b = c = 2) are set to their empirically optimal values, maintaining a stable trade-off between individual learning and collective guidance. To avoid stagnation in local optima, we introduce worst-case coefficients (d = e = –1) that occasionally push particles away from suboptimal regions. The cognitive scaling factor ($c1$ = –1) promotes individual diversity, while the social scaling factor ($c2$ = 2) fosters cohesion within the swarm. We maintain a relatively high inertia weight (w = 0.9) to encourage broader exploration, and apply a damping factor (r = 0.9) to stabilize the velocity updates. During the exploration phase, 10 iterationsare executed with a controlled velocity decay (k = 0.2) to gradually narrow the search. Additionally, the swarm is partitioned into three equally sized subpopulations ($N1$ = $N2$ = $N3$ = 10) to ensure parallel and diversified exploration across multiple regions of the solution space.

QUALITY CONTROL

To ensure the reliability and correctness of the ExPSO library, we developed a comprehensive unit testing suite using the *pytest* framework. These unit tests validate the core functionalities of the package, including particle initialization, velocity and position updates, fitness evaluations, convergence criteria, and integration with external deep learning frameworks such as PyTorch and TensorFlow. Table 2 summarizes the unit tests that verify core functionalities of the ExPSO algorithm, all of which have successfully passed, ensuring reliable initialization, updates, evaluation, and integration.

EXAMPLE USAGES

We provide two examples designed to help users become acquainted with the ExPSO package. These examples gradually increase in complexity, allowing users to gain familiarity with the package. The full code for the experiments is available on our GitHub site.

Example 01: ExPSO with Rosenbrock function

In this example, we demonstrate the utilization of the ExPSO package for a straightforward parameter optimization task. Our objective is to find optimal solutions for the Rosenbrock function [21] (Figure 2). The code begins by importing necessary libraries (lines 1 and 2) and initializing inputs such as *runs, Ib, ub, D, nPop,* and *MaxIt* (lines 3–8). We then define the objective function specifically for the Rosenbrock function and instantiate an *ExPSOCIass* (lines 9 and 12). The optimization process starts with a call to the *optimize*() function (line 14), and the resulting best parameters are captured in a variable presented on line 16. Figure 3 illustrates the multiple metric outputs obtained from executing the code.

TEST NAME	PURPOSE	STATUS
test_initialize_particles()	Verifies correct generation of initial positions and velocities	Passed
test_velocity_update()	Ensures accurate computation of velocity updates per ExPSO equations	Passed
test_position_update()	Checks boundary handling and correct position updates	Passed
test_fitness_evaluation()	Confirms objective function evaluation returns valid and expected outputs	Passed
test_global_best_selection()	Validates that global best particle is correctly identified and tracked	Passed
test_integration()	Tests compatibility with PyTorch models and parameter space setup	Passed
test_termination_criteria()	Checks early stopping and max iteration limits work as intended	Passed
test_invalid_input_handling()	Ensures graceful handling of invalid configuration or input parameters	Passed
test_initialize_particles()	Verifies correct generation of initial positions and velocities	Passed

Table 2 Unit Tests for ExPSO Library Functions.

```
import numpy as np
from ExPSO import ExPSOClass
runs = 30
lb = -30
ub = 30
D = 1000
nPop = 30
MaxIt = 10
def ObjFunction(x):
    return np.sum(100 * (x[1:] - x[:-1]**2)**2 + (1 - x[:-1])**2)
# create an instance of the ExPSOClass class with the specified parameters
pso = ExPSOClass.ExponentialParticleSwarmOptimizer(ObjFunction, D=D, nPop=
    nPop, MaxIt=MaxIt,lb=lb, ub=ub, runs=runs)
# optimize the function using ExPSO and retrieve the best solution
best_solution = pso.optimize()
# print the best solution found
print(f"Best solution found: {best_solution}")
```

Figure 2 Illustrative example of ExPSO with Rosenbrock function.

```
Best solution found:
{'GlobalBestCost': 0.0,
 'GlobalBestPosition': array([[-2.32808064e+00,  3.72638986e+00,  1.18030275e+01,
        -2.80199098e+01,  2.59328519e+01, -1.49057827e+01,
        -2.77256194e+00,  4.47054248e+00,  2.44699480e+01,
        -2.68899252e+01, -9.64479003e+00,  1.09886986e+01,
        -3.16572820e+00, -1.23867646e+01, -2.06098389e+01,
         1.86093654e+00,  1.71640639e+01,  1.89470347e+01,
         ..................................................]]),
'Metrics': {'ExPSO': array([0., 0., 0., ..., 0., 0., 0.]),
 'MEAN': 0.0, 'WorstSol': 0.0, 'BestSol': 0.0, 'STD': 0.0, 'Avg_FES': 0.033296337402885685}}
```

Figure 3 Output result of ExPSO after optimizing for the Rosenbrock function.

Example 02: ExPSO with MLP Model

In this example, we illustrate the application of ExPSO for fine-tuning the parameters of a Multilayer Perceptron (MLP) model. Figure 4 outlines the initial steps familiar from the previous example: importing necessary modules, initializing an ExPSO instance, and initiating the optimization process. However, instead of employing the Rosenbrock function, we define an objective function tailored for the MLP model, detailed in Figure 5. The objective function calculates the loss by invoking the MLP method, as illustrated in Figure 6. This method utilizes the "neurons" and "epochs" values obtained in each iteration (lines 2 and 3) to train the model, as shown in lines 7 and 17.

ANALYSIS USING BENCHMARK DATA

To validate the effectiveness of ExPSO across diverse scenarios, we selected a range of optimization algorithms for comparison. For standard benchmark functions (both unimodal and multi-modal), we included ExPSO, Phasor

```python
def main():
    nPop = 30
    runs = 10
    lb = 1
    ub = 500
    D = 2
    MaxIt = 100
    # create an instance of the ExPSOClass class with the specified parameters
    pso = ExPSOClass.ExponentialParticleSwarmOptimizer(ObjFunction, D=D, nPop=nPop, MaxIt=MaxIt,lb=lb, ub=ub, runs=runs)
    # optimize the function using ExPSO and retrieve the best solution
    cost, pos, _ = pso.optimize()
```

Figure 4 Illustrative example of ExPSO with MLP model.

```python
def ObjFunction(particles):
        allLosses = mlp(particleDimensions=particles, x_train=x_train, x_test=x_test, y_train=y_train,y_test=y_test)
        return allLosses
```

Figure 5 MLP model objective function.

```python
def mlp(particleDimensions, x_train, x_test, y_train, y_test):
        neurons = int(particleDimensions[0][0])
        epochs = int(particleDimensions[0][1])
        y_train = keras.utils.to_categorical(y_train, numberClasses)
        y_test = keras.utils.to_categorical(y_test, numberClasses)
        model = Sequential()
        model.add(Dense(neurons, input_shape=(numberFeatures,)))
        model.add(Activation('relu'))
        model.add(Dropout(rate=0.1))
        model.add(Dense(50))  # FULL CONNECTED LAYER 2
        model.add(Activation('relu'))
        model.add(Dropout(rate=0.1))
        model.add(Dense(units=numberClasses))
        model.add(Activation('softmax'))
        model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
        # TRAIN MODEL
        model.fit(x=x_train,y=y_train,epochs=epochs,verbose=0,
        batch_size=batch_size,validation_split=0.3,
        validation_data=(x_test, y_test)  )
        finalScores = model.evaluate( x=x_test, y=y_test, batch_size=batch_size, verbose=1)
        return finalScores[0]
```

Figure 6 An illustrative example of train and evaluating the MLP model with ExPSO parameters.

Particle Swarm Optimization (PPSO) [15], Fitness-based Multi-role PSO (FMPSO) [12], and eXpanded PSO (XPSO) [13] to ensure fair assessment across different landscape complexities, highlighting the balance between exploration and exploitation achieved by our proposed method. For real-world engineering problems, algorithms like PPSO, FMPSO, MSPSO, XPSO, TAPSO, EO, DSOS, HHO, IRGA, EBO with CMAR, and IMODE were chosen due to their proven performance in handling constrained, high-dimensional, and non-convex tasks. We conducted tests on 12 benchmark functions and compared its results with three state-of-the-art algorithms. These benchmark functions, extensively described in [16], cover a variety of problem types and dimensions, specifically 12 scalable problems with dimension $D = 30$. The chosen functions represent different search landscape characteristics; some are unimodal with a single global minimum, which tests the algorithm's exploitation ability, while others are multimodal with numerous local minima, testing the algorithm's exploration capabilities. The benchmark functions include:

- Unimodal benchmark functions (f_1 to f_6)—namely, the Sphere function (f_1) [24], Schwefel's functions v2.22 (f_2), v1.2 (f_3), and v2.21 (f_4) [25], Rosenbrock function (f_5) [21], and Step function (f_6) [26], respectively—are used to evaluate exploitation capability and algorithm convergence performance.

- Multimodal benchmark functions (f_7 to f_{12})—namely, Schwefel's function (f_7) [25], Rastrigin's function (f_8) [25], Ackley's function (f_9) [27], Griewank function (f_{10}) [28], Generalized Penalized function 1 (f_{11}) [29], and Generalized Penalized function 2 (f_{12}) [30], respectively—are used to evaluate exploration capability and the ability to handle multiple local optima.

Tables 3 and 4 present the comparison results, detailing average values (Avg.) and standard deviations (S.D.) across each experimental setup. For unimodal functions, Table 3 shows that all algorithms successfully identify the global optimum in most cases. ExPSO consistently achieves the global minimum across all 30 runs for functions f_1 to f_6, with average values of 0.00e+00, indicating not only exact convergence but also high numerical stability. Standard deviations for ExPSO in these cases are either exactly zero or on the order of 1e−10, which is significantly lower than competing methods, where deviations often range from 1e−4 to 1e+01. This highlights ExPSO's strong exploitation capabilities, attributed to its exponential PSO strategy.

For multimodal functions, Table 4 reveals that ExPSO consistently identifies the global minimum for all functions tested. In contrast, PPSO, XPSO, and FMPSO occasionally fail to achieve the global minimum for specific functions (i.e., PPSO for f_7, XPSO for f_{10}, and FMPSO for f_8 and f_{10}). The repeated achievement of 0.00e+00 values across multiple runs, along with consistently lower standard deviations compared to other methods, further underscores ExPSO's robust and repeatable exploration capabilities.

ANALYSIS USING ENGINEERING DATA

In this section, ExPSO is tested on four well-known engineering design problems: Pressure Vessel Design (PVD) [31], Compression Spring Design (CSD) [32], Welded Beam Design (WBD) [33], and Speed Reducer Design (SRD) [34]. These problems have different constraints that should not be violated by the optimal solution(s) obtained, and thus, a constraint handling method must be utilized.

Tables 5–8 present comprehensive performance comparisons of twelve optimization algorithms across four classic engineering design problems. The results demonstrate varying algorithmic effectiveness depending on the problem characteristics. For the Pressure Vessel Design (PVD) problem, the results indicate that ExPSO, EO, and DSOS achieve nearly identical minimum cost values. In the Coil Spring Design (CSD) task, most algorithms, except XPSO, converge to comparable solutions, while

METHOD	RESULT	f_1	f_2	f_3	f_4	f_5	f_6
ExPSO	**Avg.**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**3.17e+09**	**0.00e+00**
XPSO	Avg.	2.64e−05	0.00e+00	8.33e+00	1.81e−01	9.25e+00	0.00e+00
	S.D.	1.24e−04	5.65e−10	3.31e−04	2.05e−01	9.16e+00	0.00e+00
PPSO	Avg.	0.00e+00	0.00e+00	0.00e+00	7.99e−05	0.00e+00	0.00e+00
	S.D.	5.54e−10	1.70e−09	1.25e−10	1.52e−04	1.18e−09	0.00e+00
FMPSO	Avg.	0.00e+00	0.00e+00	0.00e+00	0.00e+00	7.97e−01	0.00e+00
	S.D.	2.43e−09	6.30e−10	4.55e−10	1.50e−10	1.62e+00	0.00e+00

Table 3 Comparison of optimization algorithms on unimodal benchmark functions.

METHOD	RESULT	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
ExPSO	**Avg.**	−1.25e+04	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
	S.D.	**1.91e−01**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**2.73e−09**	**3.21e−09**
XPSO	Avg.	**−1.08e+04**	3.98e+00	0.00e+00	3.53e−02	3.44e−08	0.00e+00
	S.D.	3.19e+02	6.12e+02	6.12e−10	3.41e−02	1.36e−07	7.51e−10
PPSO	Avg.	−1.19e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	S.D.	7.26e+02	1.96e−10	2.80e−10	2.24e−10	3.07e−09	1.27e−10
FMPSO	Avg.	−1.10e+04	2.28e+01	0.00e+00	5.34e−02	1.38e−02	2.20e−03
	S.D.	4.22e+02	1.64e+01	3.66e−10	4.58e−02	5.92e−02	4.47e−03

Table 4 Comparison of optimization algorithms on multimodal benchmark functions.

F	EXPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBCMAR	IMODE
f(x)	**6059**	6216	6771	6090	44232	6424	**6059**	**6059**	6064	6118	6123	6156
Best	**6059**	6216	6771	6090	44232	6424	**6059**	**6059**	6064	6118	6123	6156
Worst	7332	13846	27782	7461	99145	**6424**	7544	6820	7544	7544	7473	20218
Mean	6197	6473	17380	6547	70587	6424	6641	**6095**	6684	6863	6979	9355
S.D.	350	1852	10507	791	27522	**0.000**	566	148	425	372	399	3079

Table 5 Comparison of optimum results and statistical results for the PVD problem. Values represent the objective function $f(x)$, with Best, Worst, Mean, and Standard Deviation (S.D.) across 30 independent runs. Bold values indicate the best performance for each metric.

ExPSO, DSOS, HHO, EBOwithCMAR, and IMODE attain the same optimal result. For the Welded Beam Design (WBD) problem, ExPSO delivers a solution similar to EO, DSOS, EBOwithCMAR, and IMODE, surpassing the remaining algorithms in performance. Statistical analysis further shows that the mean solution produced by ExPSO is very close to its best outcome, confirming its consistency and robustness across multiple runs. In the Speed Reducer Design (SRD) problem, comparison of the optimal and statistical results reveals that EBOwithCMAR attains the best overall cost (2639.499), whereas most other algorithms yield 2994.424. In contrast, FMPSO, XPSO, and HHO record slightly higher costs of 3048.377, 3060.097, and 2997.639, respectively.

CLASSIFICATION ANALYSIS

In Figure 7, we present a detailed comparison of the performance achieved using the proposed ExPSO algorithm against four baseline optimizers across five different model architectures: CNN [35], LSTM [36], XLNet [37], MLP [38], and VGGNet [39]. The experiments were conducted on benchmark classification datasets, including MNIST [40] for image-based models (CNN and VGGNet), and IMDB [41] for text-based models (LSTM and XLNet), while the UCI Breast Cancer dataset was used for the MLP model. Each dataset includes clearly defined class labels—for example, *digit categories (0–9)* in MNIST, *sentiment polarity (positive/negative)* in IMDB, *news topic categories* in AG News, and *benign vs. malignant* in the breast cancer dataset.

Across all tested models, ExPSO consistently achieves the highest classification accuracy, demonstrating strong robustness and generalization across both deep and traditional neural architectures. On the CNN model, ExPSO reaches an accuracy of 97%, significantly outperforming PSO (91.4%), QPSO (92%), and FastPSO (90%), suggesting superior early-stage exploration capabilities. Similarly, for the LSTM model, ExPSO again achieves 97%, surpassing PSO (92.11%) and QPSO (94%), indicating that its fine-grained control of position updates is particularly effective in sequence-based learning tasks. In more complex transformer-based architectures, such as XLNet, ExPSO outperforms QPSO (81%) and FastPSO (82%), notably by 6 percentage points, which underscores its efficiency in navigating high-dimensional attention-based search spaces. Even in simpler feedforward networks, such as MLP, where optimizer performance tends to converge, ExPSO still maintains a slight edge at 89%, compared to QPSO at 88% and FST-PSO at 87%, reflecting its stable optimization behavior. Finally, on VGGNet, a deep convolutional architecture, ExPSO attains 92%, considerably outperforming PSO (85.19%) and FastPSO (87%), further demonstrating its ability to escape suboptimal minima in deep layered structures. These consistent improvements across a variety of models validate ExPSO's design as a balanced optimizer capable of adapting to both the structural complexity and training dynamics of modern learning systems.

The performance results reveal several consistent trends that underscore the effectiveness of ExPSO as an optimization framework. Notably, performance gains are more pronounced in complex architectures such as CNN, LSTM, and XLNet, where ExPSO demonstrates its capacity to scale with increasing model depth and parameter complexity. This suggests that ExPSO is particularly well-suited for high-dimensional and non-convex search spaces that often challenge conventional PSO variants. Unlike standard PSO and FastPSO, which exhibit fluctuating performance across tasks, ExPSO displays stable convergence behavior, maintaining a narrow and consistently high accuracy range of 87% to 97%.

LIMITATIONS

While the results demonstrate that ExPSO performs competitively across a range of benchmark functions, engineering problems, and classification models, several limitations remain that warrant further investigation. First, although ExPSO achieves high accuracy and repeatability in both unimodal and multimodal optimization tasks, its current performance evaluation is limited to standard benchmark functions (f_1–f_{12}) and four engineering design scenarios. A broader evaluation on real-world, noisy, and dynamic optimization problems—such as scheduling, resource allocation, and neural architecture search—would provide a more comprehensive understanding of its robustness under diverse conditions. Second,

F	EXPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBCMAR	IMODE
f(x)	0.012665	0.012669	0.012719	0.012719	0.97149	0.012793	**0.012250**	0.012666	0.012665	0.012719	0.012666	0.012665
Best	0.012665	0.012669	0.012719	0.012719	0.97149	0.012793	**0.012250**	0.012666	0.012665	0.012719	0.012666	0.012665
Worst	0.015234	0.017773	5.4901	0.20984	2.0172	0.012793	0.013318	**0.012686**	0.017275	0.016659	0.012971	0.013004
Mean	0.012857	0.013277	2.3035	0.094071	1.3531	0.012793	0.012866	**0.012673**	0.136438	0.013684	0.012718	0.012692
S.D.	0.000523	0.001271	2.8465	0.10297	0.5773	**0.000000**	0.000173	0.000004	0.001001	0.000913	0.000006	0.000007

Table 6 Comparison of optimum results and statistical results for the CSD problem.

F	EXPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBCMAR	IMODE
f(x)	**1.690**	1.6956	1.69746	1.70960	1.891	1.6954	1.6952	1.6952	1.7119	1.7212	1.6952	1.6953
Best	**1.695**	**1.695**	1.697	1.709	1.861	**1.695**	**1.695**	**1.695**	1.711	1.721	**1.695**	**1.695**
Worst	1.841	1.793	2.004	1.955	24.15	**1.695**	1.697	**1.695**	2.242	3.235	**1.695**	2.123
Mean	1.707	1.743	1.798	1.816	9.458	**1.695**	**1.695**	**1.695**	1.816	2.201	**1.695**	1.830
S.D.	0.029	0.040	0.178	0.126	6.68	**0.000**	**0.000**	**0.00**	0.107	0.364	**0.000**	0.117

Table 7 Comparison of optimum results and statistical results for the WBD problem.

F	EXPSO	PPSO	FMPSO	MSPSO	XPSO	TAPSO	EO	DSOS	HHO	IRGA	EBOWITH CMAR	IMODE
f(x)	2994	2994	3048	2994	3060	2994	2994	2994	2997	2994	**2639**	2994
Best	2994	2994	3048	2994	3060	2994	2994	2994	2997	2994	**2639**	2994
Worst	3033	3.055	3285	3058	5383	2994	2994	2994	3803	2994	**2992**	2994
Mean	2999	3019	3156	3020	4153	2994	2994	2994	3114	2994	**2848**	2994
S.D.	9.18	3.246	119.73	33.59	676.1	**0.00**	**0.00**	**0.00**	180.9	**0.00**	59.768	**0.00**

Table 8 Comparison of optimal results and statistical results for the SRD problem.

Figure 7 Accuracy Comparison of ExPSO Library with FST-PSO, Pyswarms, QPSO, and FastPSO Libraries for Deep and Machine Learning Models.

the exponential search component, which enhances convergence, introduces moderate computational overhead. While this is mitigated through parallelism and adaptive control, the exact trade-off between added complexity and optimization gain remains to be formally quantified. In future work, we plan to benchmark computational time and memory usage across scalable high-dimensional problems to further validate ExPSO's efficiency in time-constrained or resource-limited environments. Third, while ExPSO supports integration with PyTorch and TensorFlow, current deep learning tests focus primarily on classification tasks. Future iterations will explore regression, generative modeling, and reinforcement learning settings to better understand ExPSO's applicability across varied ML pipelines. Finally, the optimizer's parameter sensitivity (particularly with respect to subpopulation size, velocity damping factor, and exponential decay rate—deserves deeper empirical tuning). Automated meta-optimization or self-adaptive control mechanisms could enhance usability by reducing manual tuning and improving generalizationacross tasks.

DEPENDENCIES
pytorch, numpy, math, tensorflow, keras, scikit-learn

SOFTWARE LOCATION
Archive

Name: Codeocean
Persistent identifier: https://codeocean.com/capsule/5975162/tree/v1
Licence: GNU General Public License (GPL)
Publisher: insaf kraidia
Version published: 1.0
Date published: 12/12/23

Code repository

Name: Github
Identifier: https://github.com/insafkraidia/ExPSO
Licence: GNU General Public License (GPL)
Date published: 13/06/23

LANGUAGE
English

(2) AVAILABILITY

OPERATING SYSTEM
This package can be run on any operating system where Python can be run (GNU/Linux, Mac OSX, Windows).

PROGRAMMING LANGUAGE
python 3.6.1+

ADDITIONAL SYSTEM REQUIREMENTS
None.

(3) REUSE POTENTIAL

The ExPSO package stands out by potentially supporting researchers across diverse fields with comparable methodologies. The GitHub repository offers a range of examples to help users become proficient with ExPSO. Starting from fundamental concepts and progressing to more advanced applications, these examples facilitate gradual familiarity with the package. Users are invited to provide feedback via the GitHub issue tracker or directly to the authors via email.

ACKNOWLEDGEMENTS

We extend our heartfelt appreciation to Qatar National Library for their invaluable support in covering the publication charge.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Insaf Kraidia orcid.org/0000-0001-5538-9883
Faculty of Information Technology, Department of Software Engineering, Al-Ahliyya Amman University, Amman, 19328, Jordan

Khelil Kassoul orcid.org/0000-0002-0792-0534
A School of Business Administration University of Applied Sciences Western Switzerland HES-SO, 1227 Geneva, Switzerland

Naoufel Cheikhrouhou orcid.org/0000-0003-2497-2528
A School of Business Administration University of Applied Sciences Western Switzerland HES-SO, 1227 Geneva, Switzerland

Saima Hassan orcid.org/0000-0002-6115-4092
Institute of Computing, Kohat University of Science and Technology, Kohat 26000, Khyber Pakhtunkhwa, Pakistan

Samir Brahim Belhaouari orcid.org/0000-0003-2336-0490
Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha P.O. Box 5825, Qatar

REFERENCES

1. **Kraidia I, Ghenai A, Zeghib N.** HST-Detector: A Multimodal Deep Learning System for Twitter Spam Detection. In: *Computational Intelligence, Data Analytics and Applications*, vol. 643, Cham; 2023. pp. 91–103. DOI: https://doi.org/10.1007/978-3-031-27099-4_8
2. **Kraidia I, Ghenai A, Zeghib N.** A Multimodal Spam Filtering System for Multimedia Messaging Service. In: *International Conference on Artificial Intelligence Science and Applications* (CAISA), vol. 1441, Cham; 2023. pp. 121–131. DOI: https://doi.org/10.1007/978-3-031-28106-8_9
3. **Kraidia I, Ghenai A, Belhaouari SB.** A Multi-Faceted Approach to Trending Topic Attack Detection Using Semantic Similarity and Large-Scale Datasets. *IEEE Access*. 2025;13:21005–21028. DOI: https://doi.org/10.1109/ACCESS.2025.3535996
4. **Abualhaj MM, Al-Shamayleh AS, Munther A, Alkhatib SN, Hiari MO, Anbar M.** Enhancing spyware detection by utilizing decision trees with hyperparameter optimization. *Bulletin EEI*. Oct. 2024;13(5):3653–3662. DOI: https://doi.org/10.11591/eei.v13i5.7939
5. **Kraidia I, Ghenai A, Belhaouari SB.** Defense against adversarial attacks: robust and efficient compressed optimized neural networks. *Scientific Reports*. Mar. 2024;14(1):6420. DOI: https://doi.org/10.1038/s41598-024-56259-z
6. **Liu W, Wang Z, Yuan Y, Zeng N, Hone K, Liu X.** A Novel Sigmoid-Function-Based Adaptive Weighted Particle Swarm Optimizer. *IEEE Trans. Cybern*. Feb. 2021;51(2):1085–1093. DOI: https://doi.org/10.1109/TCYB.2019.2925015
7. **Chen K, Zhou F, Wang Y, Yin L.** An ameliorated particle swarm optimizer for solving numerical optimization problems. *Applied Soft Computing*. Dec. 2018;73:482–496. DOI: https://doi.org/10.1016/j.asoc.2018.09.007
8. **Tian D, Zhao X, Shi Z.** Chaotic particle swarm optimization with sigmoid-based acceleration coefficients for numerical function optimization. *Swarm and Evolutionary Computation*. Dec. 2019;51:100573. DOI: https://doi.org/10.1016/j.swevo.2019.100573
9. **Raza A,** *et al.* Multi-Objective Optimization of VSC Stations in Multi-Terminal VSC-HVdc Grids, Based on PSO. *IEEE Access*. 2018;6:62995–63004. DOI: https://doi.org/10.1109/ACCESS.2018.2875972
10. **Yang J, Wang X, Bauer P.** Extended PSO Based Collaborative Searching for Robotic Swarms With Practical Constraints. *IEEE Access*. 2019;7:76328–76341. DOI: https://doi.org/10.1109/ACCESS.2019.2921621
11. **Rauf HT, Shoaib U, Lali MI, Alhaisoni M, Irfan MN, Khan MA.** Particle Swarm Optimization With Probability Sequence for Global Optimization. *IEEE Access*. 2020;8:110535–110549. DOI: https://doi.org/10.1109/ACCESS.2020.3002725
12. **Xia X,** *et al.* A fitness-based multi-role particle swarm optimization. *Swarm and Evolutionary Computation*. Feb. 2019;44:349–364. DOI: https://doi.org/10.1016/j.swevo.2018.04.006
13. **Xia X,** *et al.* An expanded particle swarm optimization based on multi-exemplar and forgetting ability. *Information Sciences*. Jan. 2020;508:105–120. DOI: https://doi.org/10.1016/j.ins.2019.08.065
14. **Sanjalawe Y, Al-E'mari S, Fraihat S, Abualhaj M, Alzubi E.** A deep learning-driven multi-layered steganographic approach for enhanced data security. *Sci Rep*. Feb. 2025;15(1). DOI: https://doi.org/10.1038/s41598-025-89189-5
15. **Ghasemi M, Akbari E, Rahimnejad A, Razavi SE, Ghavidel S, Li L.** Phasor particle swarm optimization: a simple and efficient variant of PSO. *Soft Comput*. Oct. 2019;23(19):9701–9718. DOI: https://doi.org/10.1007/s00500-018-3536-8
16. **Kassoul K, Zufferey N, Cheikhrouhou N, Belhaouari SB.** Exponential Particle Swarm Optimization for Global Optimization. *IEEE Access*. 2022;10:78320–78344. DOI: https://doi.org/10.1109/ACCESS.2022.3193396
17. **Belhaouari SB, Kraidia I.** Efficient self-attention with smart pruning for sustainable large language models. *Sci Rep*. Mar. 2025;15(1):10171. DOI: https://doi.org/10.1038/s41598-025-92586-5

18. **Kraidia I, Belhaouari SB.** Towards Robust SEA Detection: Leveraging Model Diversity and Randomization Against Adversarial Attacks. In: *2025 12th International Conference on Information Technology (ICIT)*; May 2025. pp. 84–90. DOI: https://doi.org/10.1109/ICIT64950.2025.11049179
19. **Abadi M,** *et al.* TensorFlow: A system for large-scale machine learning; May 31, 2016. *arXiv*: arXiv:1605.08695. DOI: https://doi.org/10.48550/arXiv.1605.08695
20. **Paszke A,** *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library; Dec. 03, 2019. *arXiv*: arXiv:1912.01703. DOI: https://doi.org/10.48550/arXiv.1912.01703
21. **Emiola I, Adem R.** Comparison of Minimization Methods for Rosenbrock Functions; Apr. 23, 2021. *arXiv*: arXiv:2101.10546. DOI: https://doi.org/10.1109/MED51440.2021.9480200
22. **Joseph FJJ, Nonsiri S, Monsakul A.** Keras and TensorFlow: A Hands-On Experience. In: Prakash KB, Kannan R, Alexander SA, Kanagachidambaresan GR, editors. *Advanced Deep Learning for Engineers and Scientists*. In EAI/Springer Innovations in Communication and Computing. Cham: Springer International Publishing; 2021. pp. 85–111. DOI: https://doi.org/10.1007/978-3-030-66519-7_4
23. **Frostig R, Johnson MJ, Leary C.** Compiling machine learning programs via high-level tracing.
24. **Groot RD, Van Der Eerden JP, Faber NM.** The direct correlation function in hard sphere fluids. *The Journal of Chemical Physics*. Aug. 1987;87(4):2263–2270. DOI: https://doi.org/10.1063/1.453155
25. **Bäck T, Schwefel H-P.** An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, Mar. 1993;1(1):1–23. DOI: https://doi.org/10.1162/evco.1993.1.1.1
26. **Kyurkchiev N, Markov S.** On the Hausdorff distance between the Heaviside step function and Verhulst logistic function. *J Math Chem*. Jan. 2016;54(1):109–119. DOI: https://doi.org/10.1007/s10910-015-0552-0
27. **Pinnau R, Totzeck C, Tse O, Martin S.** A consensus-based model for global optimization and its mean-field limit. *Math. Models Methods Appl. Sci.* Jan. 2017;27(01):183–204. DOI: https://doi.org/10.1142/S0218202517400061
28. **Locatelli M.** A Note on the Griewank Test Function. *Journal of Global Optimization*, Feb. 2003;25(2):169–174. DOI: https://doi.org/10.1023/A:1021956306041
29. **Ivanescu AE, Staicu A-M, Scheipl F, Greven S.** Penalized function-on-function regression. *Computational Statistics*. June 2015;30(2)539–568. DOI: https://doi.org/10.1007/s00180-014-0548-4
30. **Liu X-R, Pawitan Y, Clements M.** Parametric and penalized generalized survival models. *Statistical Methods in Medical Research*. 2018;27(5):1531–1546. DOI: https://doi.org/10.1177/0962280216664760
31. **Prasanth D.** Design and Analysis of Pressure Vessel.
32. **Paredes M, Sartor M, Masclet C.** Obtaining an optimal compression spring design directly from a user specification. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*. Mar. 2002;216(3):419–428. DOI: https://doi.org/10.1243/0954405021519906
33. The Commonwealth Scientific and Industrial Research Organisation. *Current Biology*. Mar. 1997;7(3):R126. DOI: https://doi.org/10.1016/S0960-9822(97)70976-X
34. **Mehmood R, Qazi MH, Ata H, Zaheer R.** Golinski's Speed Reducer Problem Revisited Using Genetic Algorithm. 2016;16(1).
35. **Lecun Y, Bottou L, Bengio Y, Haffner P.** Gradient-based learning applied to document recognition. *Proc. IEEE*. Nov. 1998; 86(11):2278–2324. DOI: https://doi.org/10.1109/5.726791
36. **Hochreiter S, Schmidhuber J.** Long Short-Term Memory. *Neural Computation*. Nov. 1997;9(8):1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735
37. **Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov R, Le QV.** XLNet: Generalized Autoregressive Pretraining for Language Understanding; Jan. 02, 2020. *arXiv*: arXiv:1906.08237. DOI: https://doi.org/10.48550/arXiv.1906.08237.
38. **Costin A, Isacenkova J, Balduzzi M, Francillon A, Balzarotti D.** The role of phone numbers in understanding cyber-crime schemes. In: *2013 Eleventh Annual Conference on Privacy, Security and Trust*, Tarragona, Spain: IEEE; July 2013. pp. 213–220. DOI: https://doi.org/10.1109/PST.2013.6596056
39. **Simonyan K, Zisserman A.** Very Deep Convolutional Networks for Large-Scale Image Recognition; Apr. 10, 2015. *arXiv*: arXiv:1409.1556. DOI: https://doi.org/10.48550/arXiv.1409.1556.
40. **Deng L.** The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*. 2012;29(6):pp. 141–142. DOI: https://doi.org/10.1109/MSP.2012.2211477
41. **Bahraminasr M, Sadr AV.** IMDb data from Two Generations, from 1979 to 2019; Part one, Dataset Introduction and Preliminary Analysis; Sept. 06, 2020. *arXiv*: arXiv:2005.14147. DOI: https://doi.org/10.48550/arXiv.2005.14147.

TO CITE THIS ARTICLE:
Kraidia I, Kassoul K, Cheikhrouhou N, Hassan S, Belhaouari SB 2025 ExPSO-DL: An Exponential Particle Swarm Optimization Package for Deep Learning Model Optimization. *Journal of Open Research Software,* 13: 27. DOI: https://doi.org/10.5334/jors.521

Submitted: 18 June 2024 **Accepted:** 27 September 2025 **Published:** 11 November 2025

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press.