A Novel Approach for 1D-CNN Hyperparameter Optimization in IoT Attack Detection using Particle Swarm Optimization

DOI: https://doi.org/10.54654/isj.v1i24.1097

Le Thi Hong Van, Nguyen Quang Minh, Pham Van Huong*, Nguyen Hieu Minh

***Abstract*—This study proposes a hyperparameter optimization method for one-dimensional convolutional neural network using the Particle Swarm Optimization (PSO) algorithm based on a Pareto multi-objective approach to improve the performance of IoT attack detection systems. Specifically, this study enhances the PSO algorithm by introducing an automatic termination criterion for optimization loops and proposes an early stopping mechanism, along with the optimization of the early stopping patience during the 1D-CNN model training process, thereby reducing computational costs and aligning with the resource-constrained hardware conditions of IoT. Additionally, a multi-objective optimization function is developed to balance detection performance and resource efficiency by combining validation accuracy with the 1D-CNN's execution time. The proposed method is evaluated on the Edge-IIoTset dataset. Experimental results demonstrate that the optimized model reduces execution time by 48-63% compared to the baseline model while maintaining high accuracy (over 94%). This research not only provides a practical solution for IoT security but also pioneers a novel approach to integrating evolutionary algorithms into adaptive deep learning systems and introduces a flexible method for hardware-constrained devices.**

***Tóm tắt*—Nghiên cứu này đề xuất một phương pháp tối ưu hóa siêu tham số cho mạng nơ-ron tích chập một chiều (1D-CNN) bằng thuật toán tối ưu bầy đàn (Particle Swarm Optimization - PSO), dựa trên cách tiếp cận đa mục tiêu Pareto, nhằm nâng cao hiệu quả của các hệ thống phát hiện tấn công IoT. Cụ thể, nghiên cứu đã cải tiến thuật toán PSO bằng việc đề xuất một tiêu chí dừng tự động cho các vòng lặp tối ưu hóa và đề xuất cơ chế dừng sớm, cùng với việc tối ưu ngưỡng dừng sớm cho quá trình huấn luyện mô hình 1D-CNN, giúp giảm chi phí tính toán và phù hợp với điều kiện tài nguyên phần cứng hạn chế của IoT. Ngoài ra, một hàm tối ưu đa mục tiêu đã được xây dựng, kết hợp giữa độ chính xác trên tập dữ liệu xác thực và thời gian thực thi của mô hình 1D-CNN, đảm bảo cân bằng giữa hiệu suất phát hiện và hiệu quả tài nguyên. Phương pháp đề xuất được đánh giá trên bộ dữ liệu Edge-IIoTset. Kết quả thực nghiệm cho thấy mô hình đề xuất giúp giảm thời gian thực thi từ 48-63% so với mô hình gốc mà vẫn duy trì độ chính xác cao (trên 94%). Nghiên cứu không chỉ cung cấp một giải pháp thực tiễn cho vấn đề bảo mật thiết bị IoT mà còn mở ra hướng tiếp cận mới trong việc tích hợp các thuật toán tiến hóa vào các hệ thống học sâu tự động và giới thiệu một phương pháp linh hoạt cho các thiết bị có ràng buộc về phần cứng.**

Keywords*— *IoT attack detection, one-dimensional convolutional neural network (1D-CNN), particle swarm optimization (PSO), hyperparameter optimization (HPO), multi-objective optimization.

Từ khóa*—*Phát hiện tấn công IoT, mạng nơ-ron tích chập một chiều, tối ưu bầy đàn, tối ưu hóa siêu tham số, tối ưu hóa đa mục tiêu.

I. INTRODUCTION

The Internet of Things (IoT) has revolutionized industries, healthcare, and smart cities through seamless connectivity and automation. However, the exponential growth of IoT devices projected to exceed 30 billion by 2030, according to Vailshery's research [1], has introduced unprecedented security challenges. These resource-constrained devices, limited in energy, memory, and processing power, often become prime targets for Distributed Denial-of-Service (DDoS) attacks, polymorphic malware, and zero-day vulnerability exploits.

This manuscript was received on March 28, 2025. It was reviewed on May 10, 2025, revised on June 6, 2025 and accepted on June 9, 2025.

* Corresponding author

Traditional signature-based or rule-defined Intrusion Detection Systems (IDS) prove increasingly ineffective against evolving, sophisticated threats [2]. This underscores an urgent demand for automated, adaptive, and resource-efficient solutions, particularly within the context of IoT hardware's inherent computational limitations.

Figure 1 illustrates the exponential growth of global IoT connections from 2022 to 2033, highlighting both their transformative potential and the accompanying security risks [1]. The growing reliance on IoT in critical infrastructure systems—such as smart grids and industrial monitoring systems—renders their protection a matter of critical importance.

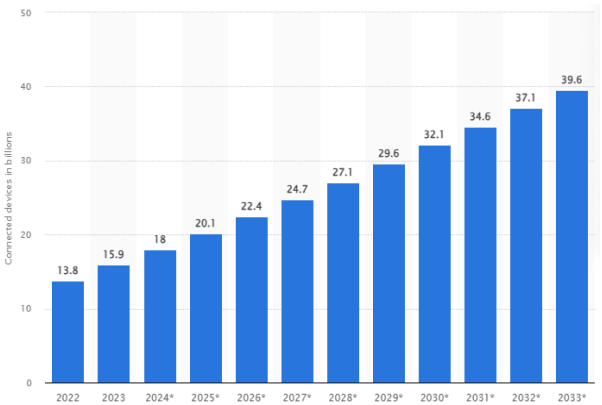

Figure 1. Number of IoT connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033

In this context, deep learning has emerged as a promising tool for IoT attack detection due to its capacity to automatically extract features from raw data. CNN deep learning models are not only suitable for image data, but also adapted to process sequence and stream data in IDS for IoT, which can enhance the detection of zero-day attacks and improve system stability. Combining CNN with configuration optimization methods, especially metaheuristic algorithms, is becoming a potential research direction to further improve detection efficiency and reduce computational costs in practical IoT systems [5, 14].

Among CNN models, one-dimensional convolutional neural networks (1D-CNN) have demonstrated exceptional advantages in processing time-series data, such as network traffic, by employing sliding filters to detect anomalous patterns without requiring manual feature engineering [3]. However, the performance of 1D-CNN heavily relies on hyperparameter selection (e.g., number of layers, filter size, learning rate). Optimizing these parameters is typically performed manually or via grid search, which is time-consuming and inefficient in high-dimensional parameter spaces.

While studies have proposed evolutionary algorithms, such as Particle Swarm Optimization, to automate hyperparameter tuning, most fail to account for the resource-constrained hardware environments during model training, resulting in high computational costs and impractical real-world deployment. To address the dual challenge of balancing performance optimization and resource efficiency, the Pareto multi-objective optimization principle serves as a strategic framework. This approach identifies a Pareto optimal solution set (Pareto Front), where no objective (e.g., accuracy) can be improved without degrading another (e.g., processing time). Building on foundational studies [4, 5] in multi-objective hyperparameter optimization for CNNs, this research advances a dynamically balanced solution that harmonizes model efficacy with practical deployability in resource-limited IoT ecosystems.

This paper focuses on optimizing 1D-CNN configurations for IoT attack detection through three primary contributions:

- Enhancing the PSO algorithm by introducing adaptive termination criteria for optimization loops, reducing computational overhead while maintaining convergence efficiency.

- Integrating an early-stopping mechanism with optimized early-stopping patience hyperparameter tuning to minimize training time without compromising detection accuracy.

- Proposing a novel multi-objective optimization function that jointly maximizes classification accuracy and minimizes execution time, ensuring models are both high-

performing and resource-efficient for deployment on edge devices.

The structure of our paper is as follows: *Section II* - Presents a review of related works. *Section III* - Covers the theoretical background. *Section IV* - Details the proposed method. *Section V* - Describes the experimental setup and evaluation of the approach. *Section VI* - Concludes the paper and suggests future research directions.

II. RELATED WORKS

Kilichev et al. [6] presented a study focused on optimizing nine hyperparameters of 1D-CNN using Genetic Algorithms (GA) and Particle Swarm Optimization for network intrusion detection, with potential applications in IoT. The experiments utilized the UNSW-NB15, CIC-IDS2017, and NSL-KDD datasets. GA and PSO were employed to optimize parameters such as the number of filters, kernel size, pooling size, number of dense layers, dropout rate, learning rate, batch size, and number of epochs. Both algorithms significantly enhanced model performance, with GA achieving accuracies of 99.31%, 99.71%, and 99.63% across the three datasets, while PSO attained 99.28%, 99.74%, and 99.52%, respectively. When compared to other CNN and hybrid models, the optimized approach demonstrates superior detection capabilities. However, the study highlighted that neither optimization method universally outperformed the other; instead, the efficacy of GA and PSO was contingent on specific dataset characteristics. The authors proposed future research directions, including the exploration of additional optimization algorithms, multi-objective optimization strategies, and validation on new datasets.

El-Ghamry et al. [7] proposed a CNN-based intrusion detection system optimized via Particle Swarm Optimization for smart agriculture - an IoT application aimed at mitigating cybersecurity risks posed by network attacks. The study utilized the NSL-KDD dataset to evaluate the performance of pre-trained CNN architectures (VGG16, Xception, Inception) following PSO-driven hyperparameter tuning. PSO was employed to optimize critical hyperparameters, including dropout rate, early-stopping patience, the number of frozen layers, learning rate, and the number of epochs. The findings revealed that the PSO-optimized models significantly enhanced performance metrics, outperforming their non-optimized counterparts. This research underscores the potential of PSO in improving intrusion detection capabilities within specific IoT contexts, such as smart agriculture.

Kan et al. [8] introduced a 1D-CNN-based method for detecting network intrusions in IoT, where hyperparameters are optimized using Adaptive Particle Swarm Optimization (APSO), an enhanced variant of PSO. APSO employs an adaptively varying inertia weight based on the fitness value, which balances exploration and exploitation in the search space, thereby addressing some limitations of traditional PSO. The study utilized a real-world dataset from nine IoT devices, encompassing attack types such as Ack, COMBO, Junk, Scan, Syn, TCP, UDP, and UDPplain. The fitness function was defined as the cross-entropy loss on the validation set after the initial training iteration of the CNN. The results demonstrated that the APSO-CNN model outperformed traditional methods such as SVM, FNN, and manually configured CNN (R-CNN) across all five evaluation metrics. However, the optimization process of APSO can be time-consuming; therefore, the authors suggested future research directions including improving the optimization algorithm and reducing computational complexity.

Bahaa et al. [9] proposed a hybrid optimization algorithm, combining Adaptive Particle Swarm Optimization (APSO) and Whale Optimization Algorithm (WOA), termed APSO-WOA, to optimize hyperparameters of 1D-CNN for detecting attacks in IoT networks. The APSO-WOA algorithm optimizes 10 hyperparameters, including the number of filters, kernel size, activation function, dropout rate, number of neurons in the fully connected layers, batch size and learning rate, with the fitness function defined as the cross-entropy loss. The method was evaluated on the N-BaIoT dataset,

comprising 115 features, and compared against models such as APSO-CNN, SVM, and FNN (Feedforward Neural Network). Experimental results demonstrated that APSO-WOA-CNN improved accuracy by 1.25% and precision by 1% compared to APSO-CNN, while significantly outperforming the other methods, confirming the effectiveness of the approach in detecting a diverse range of IoT attacks. Nevertheless, the optimization process still requires considerable computational time. Future research directions include incorporating other optimization algorithms (e.g., ant colony optimization, genetic algorithms) to enhance computational efficiency, as well as integrating data preprocessing techniques to select the most effective features.

Yang et al. [10] proposed an intrusion detection system (IDS) leveraging transfer learning and ensemble learning, utilizing CNN models with hyperparameters optimized via Particle Swarm Optimization. The optimized hyperparameters include the number of epochs, batch size, early-stopping patience, learning rate, dropout rate, and number of frozen layers. The study employed two datasets: Car-Hacking and CICIDS2017, with hyperparameter tuning primarily focused on CICIDS2017, as the default model configuration already achieved near-perfect accuracy (approximately 100%) on the Car-Hacking dataset. The proposed system attained a detection rate and F1-score exceeding 99.25% across both datasets, demonstrating its effectiveness in identifying cyberattacks within Internet of Vehicles (IoV) systems. However, the authors highlighted a critical limitation: the system's performance heavily depends on dataset diversity and richness, which may hinder its generalization in more complex, real-world scenarios.

Aguerchi et al. [11] focused on developing a novel method for breast cancer detection by optimizing CNN models using the Particle Swarm Optimization algorithm. The proposed PSOCNN model comprises four primary stages and was evaluated on the DDSM and MIAS datasets, achieving impressive accuracy rates of 98.23% (DDSM) and 97.98% (MIAS), surpassing competing algorithms. However, the study exhibits several limitations, including the lack of testing on a broader range of diverse datasets, the absence of comparisons regarding training time and computational cost with traditional methods, and the optimization being limited to a basic set of hyperparameters without extending to other critical parameters such as the number of epochs and the number of convolutional layers.

In summary, existing studies have demonstrated the efficacy of PSO in optimizing hyperparameters for 1D-CNN-based models. However, these works face key limitations: (1) they primarily focus on optimizing a limited subset of hyperparameters; (2) they are restricted to specific IoT datasets, limiting generalizability; and (3) they do not address the Pareto multi-objective optimization problem, which is critical for balancing competing objectives like accuracy and computational efficiency in resource-constrained IoT environments.

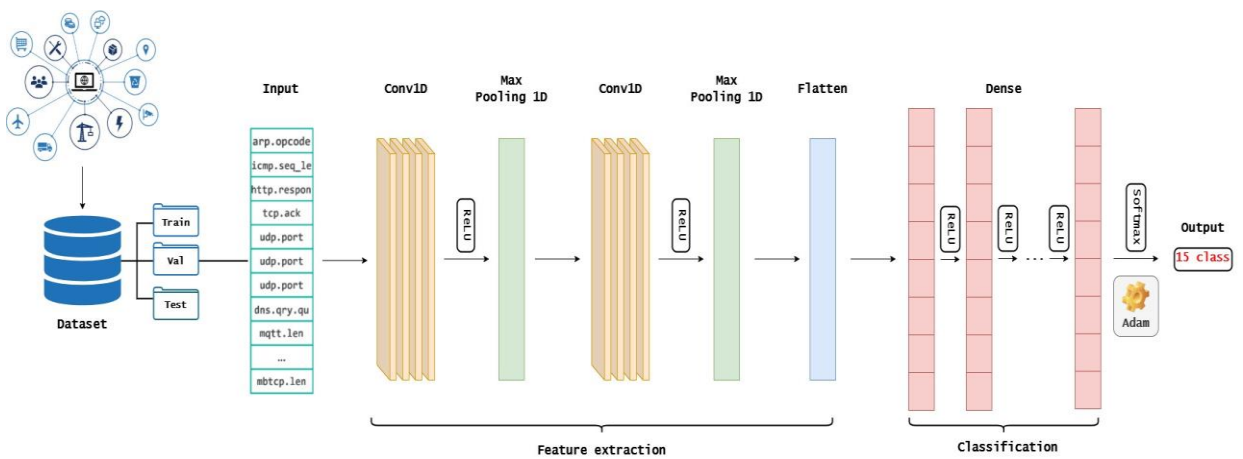

Figure 2. The architecture of the employed 1D-CNN model

III. BACKGROUND

A. 1D-CNN model

In this study, the one-dimensional convolutional neural network (1D-CNN) was selected as the primary model due to the following advantages:

- Suitability for sequential data: Unlike 2D-CNN (commonly used for image processing), 1D-CNN operates on unidimensional data. It scans through sequential data to extract local patterns, such as anomalous traffic signatures within short time intervals.

- Computational efficiency: Compared to 2D-CNN, 1D-CNN have fewer parameters and lower computational demands. For data of size $N \times N$ and a filter of size $K \times K$, the computational complexity of 2D-CNN is $O(N^2K^2)$, whereas for 1D-CNN, it reduces to $O(NK)$. This efficiency is critical for deploying models on IoT devices with constrained processing power and memory.

- Automatic feature learning: 1D-CNN automatically extract intricate features from sequential data without relying on manually feature extraction steps required in traditional machine learning, thereby reducing human effort and potential errors.

The 1D-CNN architecture used for hyperparameter optimization in this work follows a structure similar to that proposed by Kilichev et al. [6]. The detailed architecture of the 1D-CNN model is illustrated in Figure 2.

The model is built using a sequential structure with two primary convolutional blocks. Each block consists of a convolutional layer (Conv1D) with ReLU activation, followed by a max pooling layer (MaxPooling1D). Subsequent to the convolutional blocks, a Dropout layer is incorporated to mitigate overfitting. A Flatten layer then transforms the multidimensional output into a one-dimensional vector. The final segment of the model includes multiple fully connected (Dense) layers, each activated by ReLU and succeeded by a Dropout layer. The output layer employs a Softmax activation function, making the architecture suitable for multiclass classification tasks.

The model is configured with the Adam optimizer and trained using the categorical cross-entropy loss function.

B. Particle Swarm Optimization

The PSO algorithm is a metaheuristic optimization technique developed by James Kennedy and Russell Eberhart in 1995. Inspired by the movement behavior of animal swarms in nature, such as flocks of birds, schools of fish, or other organisms, PSO simulates how individuals (particles) navigate a search space by leveraging their own experience and information from neighboring particles to progress toward the optimal solution. Designed to address complex optimization problems, PSO efficiently explores high-dimensional solution spaces without relying on gradient-based methods.

Figure 3 illustrates the workflow of the PSO algorithm.

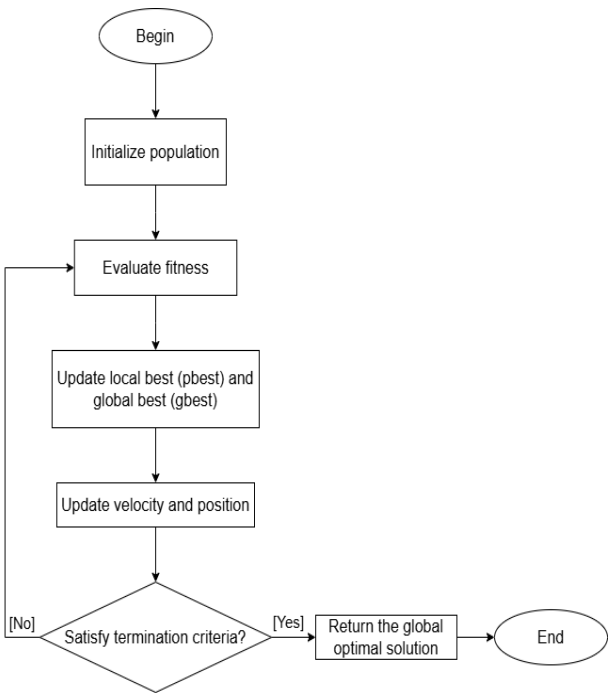

Figure 3. PSO algorithm flowchart

The PSO algorithm operates by maintaining a population of particles, each representing a potential solution to the optimization problem. The search for the optimal solution involves the following steps:

Step 1: Initialize population

A swarm of particles is randomly generated within the search space. Each particle is assigned an initial position x_i (where *i* denotes the particle index) and an individual velocity v_i.

Step 2: Evaluate fitness

Each particle's quality is assessed using a predefined fitness (objective) function, which quantifies its suitability as a solution to the optimization problem.

Step 3: Updating local best (pbest) and global best (gbest)

After evaluating the fitness values, the local best position ($pbest_i$) of each particle is updated. If the fitness of its current position (x_i) surpasses that of its previous $pbest_i$, the local best is updated as follows:

$$pbest_i = x_i \quad (1)$$

Subsequently, the algorithm compares the particle's fitness with the current global best value (gbest). If the particle's fitness exceeds *F*(gbest), where *F*(*x*) denotes the fitness function evaluated at gbest, the global best is updated:

$$gbest = x_i \quad (2)$$

Step 4: Update velocity and position

Each particle adjusts its velocity based on two guiding factors:

(i) Personal Experience (*pbest*): The best position the particle has individually achieved.

(ii) Swarm Intelligence (*gbest*): The best position discovered by any particle in the entire population.

The new velocity is calculated using the formula:

$$v_i^{(t+1)} = \omega * v_i^{(t)} + c_1 * r_1 * \left(pbest_i - x_i^{(t)}\right) + c_2 * r_2 * \left(gbest - x_i^{(t)}\right) \quad (3)$$

where:

- ω: Inertia weight, controlling the influence of the previous velocity.

- c_1, c_2: Acceleration coefficients for pbest and gbest, respectively.

- r_1, r_2: Random values uniformly distributed in [0, 1].

- $v_i^{(t)}$: Velocity of particle *i* at iteration *t*.

- $x_i^{(t)}$: Position of particle *i* at iteration *t*.

- $pbest_i$: Personal best position of particle *i*.

- $gbest$: Global best position of the swarm.

The position of particle *i* at the next iteration *t+1* is then updated using the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \quad (4)$$

Step 5: Check termination criteria

The algorithm verifies whether termination criteria—such as reaching the maximum number of iterations or achieving a satisfactory fitness value (convergence) - is met. If satisfied, the algorithm terminates and returns the global best solution *gbest*. Otherwise, it iterates back to Step 2.

In this study, each particle in the PSO algorithm represents a set of hyperparameters *S*, and the population *P* encompasses all feasible hyperparameter combinations. Our objective is to identify the optimal hyperparameter set $S^* \in P$ that maximizes the performance of the 1D-CNN model.

Definition 1 – Population (Swarm)

The population *P* is defined as:

$$P = \{S_1, S_2, \ldots, S_N\} \quad (5)$$

where: S_i (i =1..*N*) denotes a hyperparameter set of the 1D-CNN model, and *N* is the population size (number of particles).

Definition 2 – Individual (Particle)

An individual *S* is defined as:

$$S = \{s_1, s_2, \ldots, s_M\} \quad (6)$$

where: s_i (i =1..*M*) represents a hyperparameter value, and *M* is the number of hyperparameters in the 1D-CNN model.

Definion 3 – Optimization Objective

The objective of the optimization problem is to find the optimal hyperparameter set S^* that satisfies:

$$S^* = \arg\max_{S \in P} F(S) \quad (7)$$

where: S: A specific hyperparameter set, *P*: The search space of all possible hyperparameter sets, *F*(*S*): The fitness function evaluating the 1D-CNN model's performance with hyperparameter set *S*.

III. PROPOSED METHOD

A. Overall workflow

In this study, we propose an integrated model incorporating the Particle Swarm Optimization (PSO) algorithm to automatically optimize hyperparameters for a one-dimensional convolutional neural network (1D-CNN) to address attack detection in IoT systems. This comprehensive method is designed as a sequential process, combining model training with multi-objective optimization.

The process begins by initializing a population of individuals, each representing a distinct hyperparameter set *S*. These hyperparameter configurations are utilized to train and evaluate the 1D-CNN model on two datasets: training data and validation data.

After training, the model is assessed based on the fitness value of each hyperparameter set. The PSO algorithm then updates the local best (*pbest*) and the global best (*gbest*) using these evaluation results. Concurrently, the velocity and position of each individual are adjusted to guide the search toward improved solutions.

This iterative cycle continues until predefined termination criteria is met. If the stopping condition is not satisfied, the process resumes with training and updating new hyperparameters. Otherwise, the optimal 1D-CNN model, along with its corresponding hyperparameter set *S**, is obtained.

Finally, the optimized model is evaluated on a test dataset to evaluate its effectiveness and generalization capability.

Figure 4 illustrates the key steps in the 1D-CNN training and PSO-based optimization process.

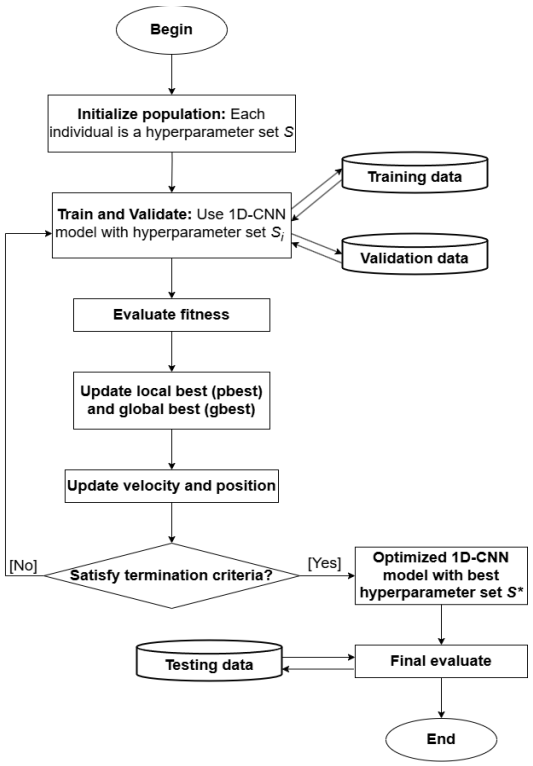

Figure 4. Flowchart of the proposed PSO-based 1D-CNN optimization method

B. Enhanced PSO for 1D-CNN optimization algorithm

The following part presents Algorithm, which integrates PSO to optimize hyperparameters for a one-dimensional convolutional neural network (1D-CNN). This algorithm comprises two phases: initialization and iterative optimization, designed to systematically identify the optimal hyperparameter configuration.

Algorithm 1: Hyperparameter optimization for 1D-CNN model using PSO

Input:

-**N**: Number of individuals in the population (a set of hyperparameter configurations *S*).

-**M**: Number of hyperparameters in an individual (dimensionality of the search space).

-**hyperparameter_bounds**: Search range for each hyperparameter.

-**w**: Inertia weight, controlling the influence of the previous velocity.

-**c1**: Cognitive coefficient (personal), learning from individual experience.

-**c2**: Social coefficient, learning from the swarm.

-**stopping_threshold**: Minimum improvement threshold.

-**max_stagnation**: Maximum number of iterations without improvement.

Output:

gbest: The optimal hyperparameter configuration *S** for the 1D-CNN model.

```
# Initialization
FOR i = 1 to N DO
    position[i] ← Random position of individual i within hyperparameter_bounds
    velocity[i] ← Random velocity
    pbest_position[i] ← NULL
    pbest_fitness[i] ← NULL
END FOR
gbest_position ← NULL
gbest_fitness ← NULL
previous_gbest ← NULL
no_improvement_count ← 0
iteration ← 0
# Optimization Process
WHILE no_improvement_count < max_stagnation DO
  FOR i = 1 to N DO
    fitness ← evaluate_fitness(position[i])
    IF fitness > pbest_fitness[i]:
      pbest_position[i] ← position[i]
      pbest_fitness[i] ← fitness
    END IF
    IF fitness > gbest_fitness:
      gbest_position ← position[i]
      gbest_fitness ← fitness[i]
    END IF
  END FOR
  FOR i = 1 to N DO
    FOR j = 1 to M DO
      r1, r2 ← Random numbers in the range [0, 1]
      velocity[j][i] ← ω * velocity[j][i] +
      c1 * r1 * (pbest[i][j] − position[j][i]) +
      c2 * r2 * (gbest[j] − position[j][i])
      position[j][i] ← position[j][i] + velocity[j][i]
      position[j][i] ← clip(position[j][i], hyperparameter_bounds[j])
    END FOR
  END FOR
  IF previous_gbest is not NULL:
    improvement ← gbest_fitness - previous_gbest
    IF improvement <= stopping_threshold:
      no_improvement_count ← no_improvement_count + 1
    ELSE:
      no_improvement_count ← 0
    END IF
  END IF
  previous_gbest ← gbest_fitness
  iteration ← iteration + 1
END WHILE
RETURN gbest
```

Explanation of steps in the algorithm:

1. Initialization

- Initialize *N* particles with randomized positions and velocities within the hyperparameter search space.

- Set coefficients *c1* (cognitive acceleration), *c2* (social acceleration), and *w* (inertia weight) to regulate particle velocity and movement behavior.

- Establish initial *pbest* (personal best position) and *gbest* (global best position).

- Initialize termination condition trackers and iteration counters.

2. Optimization Process

The algorithm iterates until meeting termination criteria (e.g., no improvement), comprising three phases:

2.1. Evaluate population

- Evaluate fitness: Evaluate the 1D-CNN model's performance using each particle's current hyperparameters.

- Update local best: Update *pbest* if the current fitness exceeds the particle's historical best.

- Update global best: Update *gbest* if the current fitness outperforms the global best, while resetting the stagnation counter.

2.2. Update population (velocity & position)

- Adjust particle velocity using inertial, cognitive, and social components

- Update particle positions while enforcing search space boundaries.

2.3. Check termination condition: Assess fitness improvement and increment the stagnation counter if no significant progress occurs.

3. Terminate and return output: Upon meeting termination criteria, return *gbest* (optimal hyperparameters *S**) and the auto-saved best model.

In the work of Kilichev et al. [6], the PSO algorithm employs a fixed loop mechanism with a predefined maximum number of iterations. This approach generally leads to two main limitations:

(1) Premature termination before convergence, yielding suboptimal solutions.

(2) Redundant post-convergence iterations, wasting computational resources (CPU/GPU time, energy).

To address these issues, we replace static termination with an adaptive stopping criterion combining that combines a minimum improvement threshold with a maximum number of consecutive iterations without improvement:

- Minimum improvement threshold (stopping threshold): This represents the minimum fitness improvement required between two successive iterations to be considered as "progress". In this study, it is set to 10^{-4}. This value was determined through experiments with three threshold levels: 10^{-3}, 10^{-4} và 10^{-5} using the dataset. The results demonstrate that a threshold of 10^{-4} achieves an optimal balance between convergence speed and accuracy. Compared to 10^{-3}, it enhances convergence speed, while relative to 10^{-5}, it delivers comparable performance with reduced computational time.

- Maximum stagnation iterations (max stagnation): This allows the algorithm to terminate if the global fitness (*gbest*) does not significantly improve over a specified number of consecutive iterations. In this model, it is set to 3, based on an analysis balancing exploration capability and computational efficiency, tested with values of 2, 3, and 4 iterations. The findings indicate that 3 iterations allow the algorithm sufficient time to overcome local noise and temporary stagnation, while preventing unnecessary computational costs and avoiding premature termination due to random fluctuations.

These parameter values were derived from a systematic study evaluating six key criteria: convergence speed, improvement magnitude, noise resistance, stability, avoidance of local optima, and computational efficiency. The results confirm that a threshold of 10^{-4} combined with a stagnation limit of 3 iterations optimizes the PSO algorithm's performance, ensuring accuracy, efficiency, and robustness against noise.

This adaptive stopping mechanism offers several important benefits:

(1) Computational efficiency: Eliminates redundant post-convergence iterations.

(2) Overfitting prevention: Limits excessive optimization that may degrade model generalizability.

(3) Dynamic adaptation: Self-adjusts to diverse search spaces by monitoring real-time optimization trends rather than relying on fixed iteration counts.

C. Hyperparameter optimization

Based on related research results mentioned above, in this study, we propose a set of hyperparameters *S*, to be optimized for the 1D-CNN model, defined as follows:

$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\} \quad (8)$$

where:

- s_1: Number of filters - determines the model's feature extraction capability.

- s_2: Kernel size - influences the local analysis scope of the data.

- s_3: Pooling size - affects the degree of feature information compression.

- s_4: Number of dense layers - governs the depth of high-level feature learning.

- s_5: Number of neurons in dense layers - impacts nonlinear representation capacity.

- s_6: Dropout rate - controls regularization strength to prevent overfitting.

- s_7: Learning rate - adjusts the convergence speed of the learning process.

- s_8: Batch size - influences training stability and speed.

- s_9: Number of epochs - determines the total learning duration.

- s_{10}: Early-stopping patience - regulates the optimal stopping point.

This study integrates an early-stopping mechanism into the 1D-CNN model, a significant enhancement compared to the original approach by Kilichev et al. [6]. The mechanism is designed to automatically halt training when performance plateaus after a predefined number of epochs, thereby mitigating overfitting and optimizing training time.

Furthermore, our research incorporates early-stopping patience into the list of hyperparameters optimized via the PSO algorithm. This allows the 1D-CNN model to automatically adjust the maximum waiting epochs before termination (e.g., 5, 10, or 15 epochs) based on data characteristics and convergence process.

The early-stopping mechanism is configured with several critical parameters, including monitor and mode. For the monitor parameter, the study employs 'val_accuracy' to track validation accuracy. Correspondingly, the mode parameter is set to 'max' (training stops when accuracy ceases to improve).

Table I presents the list of hyperparameters optimized for the 1D-CNN model, along with their respective value ranges.

TABLE I. RANGE OF OPTIMIZED HYPERPARAMETER VALUES

Symbols	Hyperparamter	Range
s_1	Number of filters	[16, 32, 64]
s_2	Kernel size	[3, 5, 7]
s_3	Pooling size	(2, 5)
s_4	Number of dense layers	(1, 3)
s_5	Number of neurons in dense layers	[128, 256, 512]
s_6	Dropout rate	(0.1, 0.5)
s_7	Learning rate	$(10^{-5}, 10^{-2})$
s_8	Batch size	[32, 64, 128, 256, 512]
s_9	Number of epochs	(20, 50)
s_{10}	Early-stopping patience	(5, 10)

Unlike the approach in [6], we selectively narrow the value domains of certain hyperparameters. Combined with PSO's velocity update mechanism, this strategy maintains effective exploration while filtering out infeasible configurations, reducing overfitting risks and computational waste. This narrowing reflects domain knowledge-driven design, balancing model capacity with the resource constraints of IoT systems.

The proposed hyperparameter, early-stopping patience (s_{10}), has its value range established through a series of preliminary experiments conducted on the dataset prior to the main model improvement phase. We observed that patience values smaller than 5 epochs often result in premature stopping, whereas values exceeding 10 epochs fail to provide significant

accuracy improvements while unnecessarily prolonging training time.

The findings demonstrate that a patience range of 5-10 epochs not only aligns with the convergence properties of the PSO algorithm but also effectively supports the learning capacity of the 1D-CNN model on IoT data. This ensures the model is afforded sufficient time to capture critical patterns without falling into overfitting or squandering computational resources.

In the problem of optimizing a 1D-CNN model for IoT attack detection, designing a mechanism to map particle positions to model hyperparameters plays a decisive role in the effectiveness of the PSO algorithm. Since PSO typically operates in a continuous space, while many 1D-CNN hyperparameters require discrete or integer values, a suitable conversion mechanism is essential. This mapping process defines the search space *S*—the set of feasible 1D-CNN configurations—enabling the PSO algorithm to efficiently explore *S* despite operating in a continuous space.

The mapping method addresses two main hyperparameter types:

- Continuous hyperparameters: Directly mapped from the PSO search space.
- Discrete hyperparameters: Mapped via a two-step process: continuous value quantization followed by mapping to predefined value sets.

The mapping function is constructed based on three main principles:

(1) Direct dimension-to-parameter mapping

Each dimension in the particle's position vector corresponds to a specific 1D-CNN hyperparameter, as defined by the params_keys list. This mechanism establishes a one-to-one relationship between the PSO space and the parameter space, enabling simultaneous optimization of heterogeneous parameters.

```
    params_keys = ['num_filters', 'kernel_size',
..., 'patience']

    for j, key in enumerate(params_keys):

        value = particles[j][i]
```

(2) Continuous value quantization

Integer hyparameters are quantized from the particle's continuous values using rounding. This resolves the paradox between PSO's continuous space and the discrete nature of many hyperparameters.

```
    if key in ['num_filters', 'kernel_size',
..., 'patience']:

    value = int(round(value))
```

(3) Value constraint to predefined sets

For hyperparameters with finite value domains, a clipping index technique selects values from predefined sets. Specifically, quantized integer values are mapped to indices of the value_choices array, with an upper bound set to the array length to prevent index overflow. This confines the search space to empirically validated values.

```
    value_choices = [16, 32, 64]

    value=value_choices[min(int(value),
len(value_choices)-1)]
```

The integration of quantization and domain-based constraints allows PSO to avoid meaningless search regions while retaining flexibility in exploring optimal configurations for IoT attack detection.

D. Multi-objective optimization function

The optimization of the 1D-CNN model for IoT attack detection employs a global multi-objective optimization function, formulated as:

$$\text{Maximize } \mathbf{F}(S) = \{f_1(S), f_2(S)\} \qquad (9)$$

where:

- F: Global objective function.

- f_1: Accuracy objective function.

- f_2: Execution time objective function.

- S: Hyperparameter set of the 1D-CNN model.

These functions are defined and constructed below.

Definition 4 – Accuracy Objective Function

Accuracy is the core metric for evaluating the classification performance of the model. The accuracy objective function is based on the accuracy in validation dataset, calculated as the

ratio of correctly classified samples to the total validation samples:

$$f_1 = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \quad (10)$$

where:

- *TP* (True Positives): Number of correctly identified attack samples.

- *TN* (True Negatives): Number of correctly identified normal samples.

- *FP* (False Positives): Number of normal samples misclassified as attacks.

- *FN* (False Negatives): Number of attack samples misclassified as normal.

In terms of dimensionality, f_1 is represented as a dimensionless ratio with values ranging from 0 to 1, where 0 indicates the lowest accuracy and 1 indicates perfect accuracy. Therefore, no additional normalization steps are required for this function.

The goal is to maximize f_1 to ensure accurate attack detection. However, overemphasizing accuracy may lead to overly complex models with excessive computational demands.

Definition 5 – Execution Time Objective Function

Execution time reflects the computational efficiency of the model, which is critical for real-time IoT systems. This value is measured as the total time for model training and evaluation on validation datasets.

To normalize the execution time value, this study constructs the function f_2 as an inverse exponential function as follows:

$$f_2 = \frac{1}{e^{k.T}} \quad (11)$$

where: k is the decay coefficient (unit: s^{-1}), determined as the reciprocal of the reference time T_0 ($k = \frac{1}{T_0}$); T is the execution time of the 1D-CNN model (unit: s).

The normalization mechanism of f_2 relies on the principle of nonlinear exponential transformation to map execution time values from the extended domain $(0,+\infty)$ to the bounded domain (0,1). Specifically, as the execution time T approaches 0, the value of f_2 approaches 1 (optimal); conversely, as T increases towards infinity, f_2 asymptotically approaches 0 (suboptimal). Notably, the use of the inverse exponential function also eliminates the unit of f_2. In the expression $k.T$, k has the unit s^{-1} and T has the unit s, making $k.T$ a dimensionless quantity. Consequently, f_2 is also dimensionless.

The decay coefficient k plays a crucial role in adjusting the degree of influence of the execution time on the value of f_2. Specifically:

- Larger k (corresponding to smaller T_0): Sharp decline in f_2 as T increases, imposing stricter penalties.

- Smaller k (corresponding to larger T_0): Reduced sensitivity to T, allowing a trade-off between execution time and other factors.

The choice of k depends on the specific expected range of execution time values for each problem and the priority level for time performance. In this study, k=0.001 (s^{-1}) is selected, corresponding to a reference time T_0=1000 (s), based on an analysis of the execution time range T of the 1D-CNN model in the IoT attack detection problem. In this problem, the execution time typically fluctuates between 100 and 1500 seconds, with only a few special cases exceeding 1500 seconds.

Definition 6 – Global Objective Function

The global optimization function F is designed to evaluate the balance between the individual optimization objectives. The aim in this problem is to find the maximum value of F.

The function F combines f_1 and f_2 through a weighting factor w, as follows:

$$\mathrm{F} = w \times f_1 + (1 - w) \times f_2 \quad (12)$$

where: w is the weight parameter adjusting the priority between objectives ($w \in [0,1]$).

In multi-objective optimization, w is tuned based on the relative importance of accuracy versus computational efficiency or specific system requirements. Adjustments to w should

follow a cost-benefit analysis to ensure the model satisfies IoT system constraints.

In this study, to identify the optimal weight values, we conducted experiments with 9 different weight values ranging from 0.1 to 0.9, with increments of 0.1, on the dataset. The evaluation criteria were established based on three key factors: attack detection accuracy, execution time, and the optimization function value (*F-value*). Through analysis of the results, we observed that the weight values could be classified into three main groups according to their optimization characteristics.

The first group (w = 0.1-0.3) prioritizes execution time. However, the results indicate that an excessive focus on time optimization significantly reduces the model's accuracy, achieving only 80-86%, which does not meet the high-performance requirements for attack detection. The second group (w = 0.4-0.6) demonstrates a balance between objectives, with w = 0.5 achieving an accuracy of 94.77% and a 48.06% reduction in execution time compared to the baseline. The third group (w = 0.7-0.9) emphasizes accuracy, with w = 0.8 attaining an accuracy of 94.41% while still maintaining computational efficiency, reducing execution time by 63.69% relative to the original configuration.

The analysis reveals that while low weight values (group 1) provide high speed, but they lack reliability in attack detection. In contrast, higher weight values (group 3) ensure accuracy while preserving good computational efficiency.

Detailed experimental results for w = 0.5 (representing the second group) and w = 0.8 (representing the third group) will be presented in the next section.

IV. EXPERIMENTS AND RESULTS

A. Dataset

The dataset used in this study is Edge-IIoTset, developed by Ferrag et al. [12]. Edge-IIoTset aggregates data from over 10 distinct IoT devices, categorized into two primary groups as follows:

(1) Normal: Comprising attack-free samples, representing the system's secure operational state.

(2) Attack: Encompassing 14 attack types across five major threat categories:

- DDoS: Distributed Denial-of-Service attacks through protocols such as UDP (User Datagram Protocol), TCP (Transmission Control Protocol), HTTP (HyperText Transfer Protocol), and ICMP (Internet Control Message Protocol).

- Injection: Including SQL Injection and Cross-site Scripting (XSS).

- Reconnaissance: Activities such as Port Scanning and Fingerprinting.

- Malware: Ransomware, Backdoor, and Uploading attacks.

- MITM: Man-in-the-Middle attacks.

The sample distribution across classes in Edge-IIoTset is illustrated in Figure 5.

The initial preprocessing steps align with those outlined by Ferrag et al. [12], including the removal of low-significance columns, outlier handling, label encoding, and similar procedures.

Additionally, this study introduces a novel method to reduce dataset memory footprint, optimizing processing efficiency for large-scale data. Specifically:

- For integer columns (int64): Values are converted to smaller integer or unsigned integer types, contingent on the int2uint flag. This minimizes storage requirements without data loss.

- For float columns (float64): Values are downcast to float32, halving memory usage while preserving high precision.

This memory reduction is critical for handling large datasets like Edge-IIoTset, accelerating processing speed and conserving system resources without compromising data integrity.

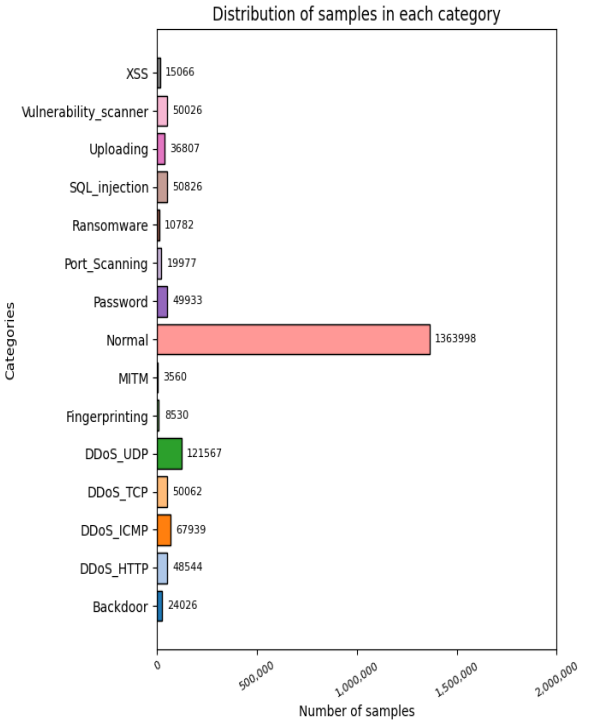

Figure 5. The distribution of the number of samples within each class in Edge-IIoTset

Furthermore, we implemented a data processing method from Kilichev et al. [13], where a hash function is applied per column to identify identical features. By comparing hash values, duplicate column groups were detected and removed. This step is vital for eliminating redundancy, thereby enhancing model efficiency.

After the preprocessing and cleaning procedures, the number of features was reduced from 95 to 86.

Finally, the dataset was partitioned into three subsets:

- Training set: 70% of the data.

- Validation set: 10% of the data.

- Test set: 20% of the data.

B. Evaluation metrics

To evaluate the performance of the IoT attack detection model based on 1D-CNN with hyperparameter optimization via PSO, this study employs Accuracy as the primary metric, supplemented by three additional metrics: Precision, Recall, and F1-score. The primary metric (Accuracy) is applied to the training, validation, and test datasets. The supplementary metrics (Precision, Recall, F1-score) are calculated post-training through evaluation on the test set, providing a comprehensive overview of the model's effectiveness in accurately detecting attacks and the reliability of its predictions.

Accuracy measures the ratio of correct predictions to the total number of samples, reflecting the model's overall classification capability:

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \qquad (13)$$

where:

- *TP* (True Positives): Number of correctly identified attack samples.

- *TN* (True Negatives): Number of correctly identified normal samples.

- *FP* (False Positives): Number of normal samples misclassified as attacks.

- *FN* (False Negatives): Number of attack samples misclassified as normal.

Precision (class-specific accuracy) quantifies the proportion of correct predictions among all samples predicted as a specific class:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad (14)$$

where: TP (True Positives) is the number of correctly identified attack samples; *FP* (False Positives) is the number of normal samples misclassified as attacks.

Recall (class-specific sensitivity) measures the proportion of correctly identified samples relative to all actual samples of a class:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \qquad (15)$$

where: TP (True Positives) is the number of correctly identified attack samples; *FN* (False Negatives) is the number of attack samples misclassified as normal.

F1-score: The F1-score is the harmonic mean of Precision and Recall.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (16)$$

C. Model evaluation

To evaluate the effectiveness of the proposed method, we conducted a series of experiments on the aforementioned dataset to compare two models: the baseline model from the original study [6] (Model 0) and our proposed improved model (Model 1). Both models utilize the multi-objective optimization function F (defined above) with different w values (where w = 0.5 represents a balanced emphasis on accuracy and execution time, while w = 0.8 prioritizes accuracy more heavily).

Table II and Figure 6 presents a comparative performance analysis between the two models, including the *F*-value, validation accuracy, and execution time.

TABLE II. PERFORMANCE AND EXECUTION TIME COMPARISON

Model	*w*	Max F-value	Validation accuracy	Execution time (s)
Model 0 (Original method)	0.5	0.8752	0.9474	219.45
	0.8	0.9097	0.9478	278.16
Model 1 (Proposed method)	0.5	0.9200	0.9477	113.98
	0.8	0.9361	0.9441	101.00

With w = 0.5, Model 1 achieved an F-value of 0.9200, 5.12% higher than Model 0 (0.8752). Notably, Model 1 significantly reduced execution time to 113.98 seconds, a 48.06% reduction compared to Model 0 (219.45 seconds), while maintaining competitive accuracy (a marginal 0.03% improvement over Model 0).

When w was increased to 0.8, both models exhibited improved *F-value*, but Model 1 retained superiority with an *F-value* of 0.9361 (the highest in the experiment), outperforming Model 0 (0.9097) by 2.9%. Model 1 also demonstrated markedly lower execution time (101.00 seconds), representing a 63.69% reduction compared to Model 0 (278.16 seconds).

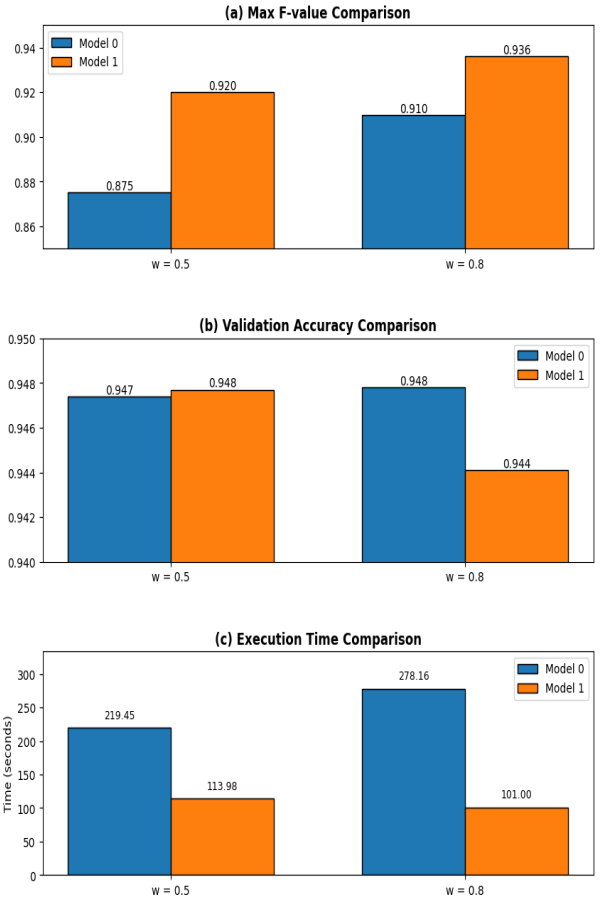

Figure 6. Comparison chart of global optimal F-value, accuracy and execution time of original model and proposed model

Table 3 provides a comprehensive comparison, including the optimized hyperparameter set, overall model training time, the number of parameters used, and the evaluation performance on the test set.

Both models achieved high accuracy (over 94%), with negligible differences (0.02%-0.35%). Precision and Recall exceeded 97% and 92%, respectively, for both models, indicating robust attack detection capabilities.

At w = 0.5, despite an slightly increase in training time (attributed to additional iterations required to confirm the lack of further improvement) and equivalent parameter counts (168,529), Model 1 achieved a superior *F1-score* (0.7881 compared to 0.7804), reflecting better balance between true positives and false alarm reduction.

TABLE III. OPTIMAL CONFIGURATION AND OVERALL COMPARISON

Model	w	Optimal hyperparameter set	Total training time (s)	Number of parameters	Performance evaluation on test set			
					Acc.	Precision	Recall	F1-score
Model 0 (Original method)	0.5	{64, 7, 5, 1, 512, 0.5, 0.0009, 512, 20}	12736.94	168,529	0.9479	0.9831	0.9226	0.7804
	0.8	{64, 7, 5, 3, 512, 0.1, 0.0016, 512, 22}	16332.09	693,841	0.9479	0.9828	0.9222	0.7976
Model 1 (Proposed method)	0.5	{64, 7, 5, 1, 512, 0.2416, 0.0022, 512, 44, 5}	13806.38	168,529	0.9477	0.9822	0.9221	0.7881
	0.8	{64, 7, 5, 2, 512, 0.1, 0.0065, 512, 20, 5}	15127.00	431,185	0.9444	0.9775	0.9201	0.7458

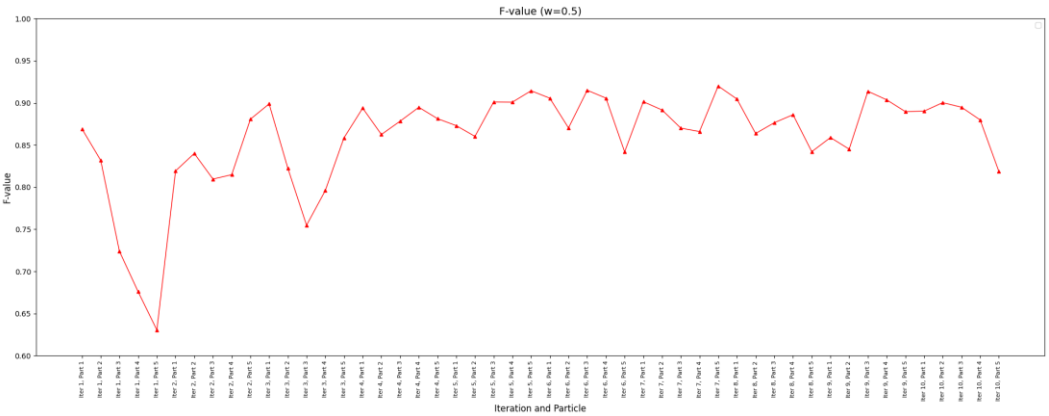

Figure 7. F-value convergence chart of proposed model (w = 0.5)

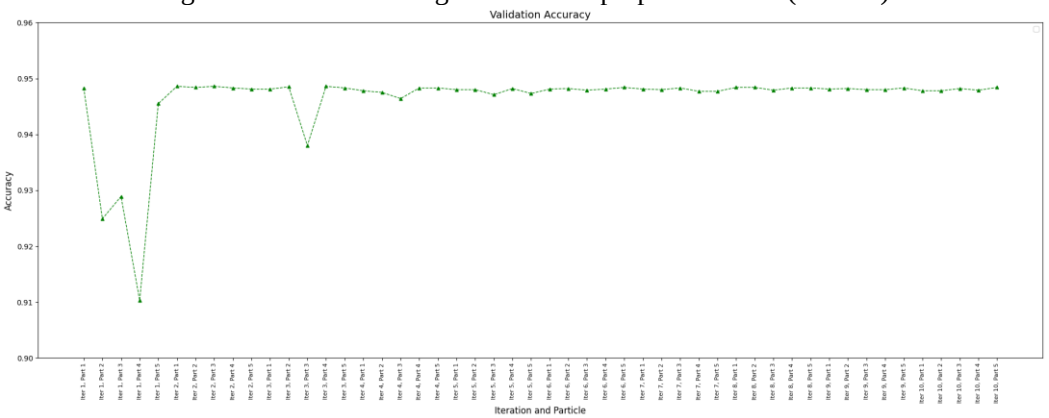

Figure 8. Validation accuracy values convergence chart of proposed model (w = 0.5)

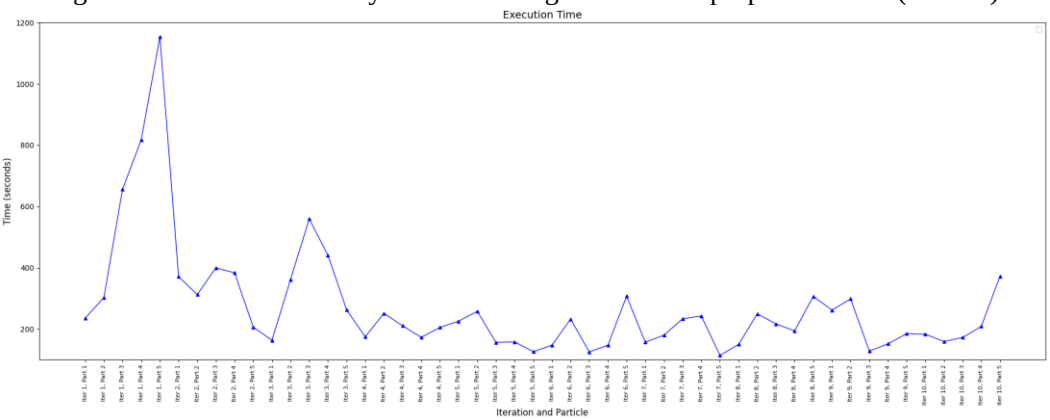

Figure 9. Execution time convergence chart of proposed model (w = 0.5)

At w = 0.8, Model 1 demonstrated remarkable efficiency:

- 37.85% fewer parameters (431,185 compared to 693,841).

- 7.38% reduction in total training time compared to Model 0.

For the case w = 0.5, the experimental program stopped after 10 iterations and selected the 5th particle at iteration 7 (iter 7, part 5) as the best CNN configuration. Figure 7, Figure 8 and Figure 9 respectively illustrate the convergence behavior of F-value, validation accuracy, and execution time across all iterations and particles during the PSO optimization process for the proposed 1D-CNN model.

These results underscore the effectiveness of the proposed early stopping mechanism, adaptive termination criteria and the multi-objective optimization function in the enhanced PSO algorithm, in maintaining high accuracy while optimizing computational resource utilization.

IV. CONCLUSION

This study proposes an optimization method for 1D-CNN model in IoT attack detection using the PSO algorithm. The key contributions include:

- An enhanced PSO algorithm with automatic termination criteria based on convergence analysis;

- Integration of an early-stopping mechanism with optimized patience thresholds to prevent overfitting and reduce computational overhead in 1D-CNN training;

- A multi-objective optimization function that balances detection accuracy and execution time.

Experimental results demonstrate that the proposed method achieves significant reductions in execution time (up to 63.69%) and parameter count (37.85%) compared to the baseline model, while maintaining high detection accuracy (over 94%). These outcomes validate its efficacy in resource optimization for edge devices.

However, the method occasionally exhibits slight declines in accuracy when prioritizing execution time optimization, highlighting the need for further research into adaptive weight calibration. Current limitations also include the lack of real-time performance evaluations in dynamic environments and cross-dataset validation, which restricts generalizability. Our future work will focus on testing the method across diverse datasets and evaluating its performance on real-world resource-constrained IoT edge devices. This research has paved the way for deploying lightweight, high-efficiency models in IoT security systems with stringent resource constraints, emphasizing the trade-off between computational efficiency and detection reliability.

REFERENCES

[1] L. S. Vailshery, "Number of IoT connections worldwide 2022-2033" (2025), [Accessed: 01/03/2025], *https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/*.

[2] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?", *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, 2018, doi: 10.1109/MSP.2018.2825478.

[3] M. K. Hooshmand and M. D. Huchaiah, "Network Intrusion Detection with 1D Convolutional Neural Networks", *Digital Technologies Research and Applications*, vol. 1, no. 2, pp. 66–75, 2022, doi: 10.54963/dtra.v1i2.64.

[4] L. T. H. Van, P. V. Huong, and N. H. Minh, "The Multi-objective Optimization of the Convolutional Neural Network for the Problem of IoT System Attack Detection", in *Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO 2021)*, Lecture Notes in Networks and Systems, vol. 363, Springer, 2021, pp. 350–360.

[5] L. T. H. Van, L. D. Thuan, P. V. Huong, and N. H. Minh, "A New Method to Improve the CNN Configuration for IoT Attack Detection Problem based on the Genetic Algorithm and Multi-Objective Approach", in *2024 1st International Conference On Cryptography And Information Security (VCRIS)*, Hanoi, Vietnam, 2024, pp. 1-9.

[6] D. Kilichev and W. Kim, "Hyperparameter Optimization for 1D-CNN-Based Network Intrusion Detection Using GA and PSO", *Mathematics*, vol. 11, no. 17, 2023, doi: 10.3390/math11173724.

[7] A. El-Ghamry, A. Darwish and A. E. Hassanien, "An optimized CNN-based intrusion detection system for reducing risks in smart farming", *Internet of Things*, vol. 22, 2023, doi: 10.1016/j.iot.2023.100709.

[8] X. Kan, Y. Fan, Z. Fang, L. Cao, N. N. Xiong, D. Yang and X. Li, "A novel IoT network intrusion detection approach based on Adaptive

Particle Swarm Optimization Convolutional Neural Network", *Information Sciences*, vol. 568, pp. 147-162, 2021, doi: 10.1016/j.ins.2021.03.060.

[9] A. Bahaa, A. Sayed, L. Elfangary, and H. Fahmy, "A novel hybrid optimization enabled robust CNN algorithm for an IoT network intrusion detection approach", *PLOS ONE*, vol. 17, no. 12, 2022, doi: 10.1371/journal.pone.0278493.

[10] L. Yang and A. Shami, "A Transfer Learning and Optimized CNN Based Intrusion Detection System for Internet of Vehicles", in *ICC 2022 - IEEE International Conference on Communications*, Seoul, Republic of Korea, 2022, pp. 2774–2779.

[11] K. Aguerchi, Y. Jabrane, M. Habba, and A. H. El Hassani, "A CNN hyperparameters optimization based on particle swarm optimization for mammography breast cancer classification", *Journal of Imaging*, vol. 10, no. 2, 2024, doi: 10.3390/jimaging10020030.

[12] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras and H. Janicke, "Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning", *IEEE Access*, vol. 10, pp. 40281-40306, 2022, doi: 10.1109/ACCESS.2022.3165809.

[13] D. Kilichev, D. Turimov and W. Kim, "Next–Generation Intrusion Detection for IoT EVCS: Integrating CNN, LSTM, and GRU Models", *Mathematics*, vol. 12, no. 4, 2024, doi: 10.3390/math12040571.

[14] P. V. Huong, L. T. H. Van, và P. S. Nguyen, "Detecting Web Attacks Based on Clustering Algorithm and Multi-branch CNN," *Journal of Science and Technology on Information Security*, vol. 2, no. 12, pp. 31–37, Jul. 2021, doi: 10.54654/isj.v2i12.120.

ABOUT THE AUTHOR

Le Thi Hong Van

Workplace: Academy of Cryptography Techniques, Vietnam.

Email: lthvan@actvn.edu.vn

Education: Bachelor of Information Security (2009), Master of Cryptography Engineering (2013), currently pursuing a Ph.D. in Information Security at the Academy of Cryptography Techniques.

Recent research direction: Cryptographic algorithms, information security systems, cyber attack methods, deep learning models, IoT system security.

Tên tác giả: **Lê Thị Hồng Vân**

Cơ quan công tác: Học viện Kỹ thuật mật mã.

Email: lthvan@actvn.edu.vn

Quá trình đào tạo: Kỹ sư An toàn thông tin (2009), Thạc sỹ Kỹ thuật mật mã (2013), hiện đang là nghiên cứu sinh ngành An toàn thông tin tại Học viện Kỹ thuật mật mã.

Hướng nghiên cứu hiện nay: Thuật toán mật mã, các hệ thống an toàn thông tin, các phương pháp tấn công mạng, các mô hình học sâu, bảo mật hệ thống IoT.

Nguyen Quang Minh

Workplace: Academy of Cryptography Techniques, Vietnam.

Email: quangminhnguyen1025@gmail.com

Education: Fourth-year student, majoring in Information Technology.

Recent research direction: Machine learning, deep learning, AI in information security.

Tên tác giả: **Nguyễn Quang Minh**

Cơ quan công tác: Học viện Kỹ thuật mật mã.

Email: quangminhnguyen1025@gmail.com

Quá trình đào tạo: Sinh viên năm bốn, chuyên ngành Công nghệ thông tin.

Hướng nghiên cứu hiện nay: Học máy, học sâu, trí tuệ nhân tạo trong bảo mật thông tin.

Pham Van Huong

Workplace: Academy of Cryptography Techniques, Vietnam.

Email: huongpv@actvn.edu.vn

Education: Received a Bachelor's degree in 2005, a Master's degree in 2008, and a Ph.D. in Software Engineering in 2015 from the University of Engineering and Technology, Vietnam National University, Hanoi.

Recent research direction: AI application, blockchain, mobile and IoT.

Tên tác giả: **Phạm Văn Hưởng**

Cơ quan công tác: Học viện Kỹ thuật mật mã.

Email: huongpv@actvn.edu.vn

Quá trình đào tạo: Nhận bằng Cử nhân năm 2005, Thạc sĩ năm 2008 và Tiến sĩ Kỹ thuật phần mềm năm 2015 tại Đại học Công nghệ - Đại học Quốc gia Hà Nội.

Hướng nghiên cứu hiện nay: Trí tuệ nhân tạo ứng dụng, blockchain, phần mềm di động và hệ thống nhúng.

Nguyen Hieu Minh

Workplace: Academy of Cryptography Techniques, Vietnam

Email: hieuminh@actvn.edu.vn

Education: Received a Bachelor's degree in Radio Electronics and Telecommunications in 1993 and a Master's degree in 1999 from the Military Technical Academy (Vietnam). Earned a Ph.D. in 2005 from Saint Petersburg State Electrotechnical University (LETI), Russian Federation. Conferred the title of Professor in 2023.

Recent research direction: Electronics and Telecommunications Engineering, Information Technology, Cryptography, and Information Security.

Tên tác giả: **Nguyễn Hiếu Minh**

Cơ quan công tác: Học viện Kỹ thuật mật mã.

Email: hieuminh@actvn.edu.vn

Quá trình đào tạo: Nhận bằng đại học chuyên ngành Vô tuyến điện tử/thông tin năm 1993, thạc sĩ năm 1999 tại Học viện Kỹ thuật quân sự, bằng tiến sĩ tại ĐH Tổng hợp kỹ thuật điện Xanh Pê téc bua – Liên bang Nga năm 2005, được phong hàm Giáo sư năm 2023.

Hướng nghiên cứu hiện nay: Kỹ thuật điện tử viễn thông, Công nghệ thông tin, mật mã và an toàn thông tin.