2025 International Conference on Intelligent Computing
July 26-29, Ningbo, China
https://www.ic-icc.cn/2025/index.php

Evolutionary Replay-Driven Federated Class-Incremental Learning for Cyber-attack Detection

Junyan Su[1,3], Wenbo Fang[1,3]✉, Linlin Zhang[2], Wengang Ma[1,3], Junjiang He[1,3]

Xiaolong Lan[1,3] and Tao Li[1,3]

[1] School of Cyber Science and Engineering, Sichuan University, Chengdu, 610065, China
[2] School of Software, Xinjiang University, Urumqi, 830000, China
[3]Key Laboratory of Data Protection and Intelligent Management (Sichuan University), Ministry of Education, Chengdu, China
fangwenbo@stu.scu.edu.cn

Abstract. With the continuous evolution of cyber-attack patterns and strategies, the task of detecting cyber-attacks in real-time has become increasingly critical. However, existing replay-based class-incremental learning methods face two fundamental challenges: **a)** relying on continuous sample aggregation may raise concern about privacy data leakage, and **b)** lacking careful consideration of evolving cyber-attacks, leading to insufficient detection capabilities for attack variants. In this paper, we propose an evolutionary replay-driven federated class-incremental learning for cyber-attack detection, which effectively enhances the detection of variants in incremental learning tasks while protecting data privacy. Specifically, at task T, each local client trains a classification model and stores prototypical features ('genes') for each class, accompanied by a category-specific convolutional autoencoder (CAE) model. Under privacy-preserving mechanisms, a global network attack detection model is trained via federated learning, with subsequent updates propagated to local client models. At task $T+1$, old knowledge genes are generated from the stored prototypical sample library using a gene evolution strategy and the pre-trained CAE model. These generated features are integrated with new data for model update. Finally, the detection model is updated again through the federated learning mechanism. Extensive experiments conducted on authoritative datasets demonstrate the effectiveness of our proposed method. Experimental results show that our method achieves 90.16% accuracy in Task 2 and 85.90% accuracy in Task 3. Notably, in Task 3, our method outperforms the random replay method by 4.66%, the GAN-based replay method by 6.91%, and the VAE-based replay method by 22.32%.

Keywords: Federated Learning, Incremental Learning, Cyber-attack Detection.

✉ *Corresponding author*

1 Introduction

In recent years, with the successive open-sourcing and advancement of AI models such as ChatGPT[1] and DeepSeek[2], related technologies have gradually permeated the cybersecurity domain. According to GlobeNewswire [1], global cyberattacks in 2024 increased by 44% compared to 2023, with generative AI (GenAI) being leveraged to accelerate attack development. This trend mandates that existing detection methods continuously identify cyber-attacks without forgetting old knowledge and effectively detect attack variants.

Class-incremental learning (CIL) [2]–[5] has attracted significant attention due to its effectiveness in mitigating catastrophic forgetting in models. In particular replay-based methods are widely regarded as the most effective approach to overcoming catastrophic forgetting. For example, Sylvestre et al. [6] introduced iCaRL, a novel training strategy enabling CIL: it requires only a small subset of training data from existing classes to coexist with new data and allows gradual addition of new classes. Despite the promising performance of incremental learning methods in cyberattack detection, they still face two fundamental challenges: **a)** First, CIL approaches that continuously aggregate samples may lead to privacy data leakage, especially for sensitive cyberattack data. **b)** Existing incremental learning methods lack adequate consideration of the continuous evolution of cyberattacks, making them ineffective in dealing with ever-emerging attack variants.

One potential solution to the first challenge is federated learning (FL) [7], a concept proposed by Google in 2016. FL trains an optimal global model by enabling local clients to train models individually and only upload model parameters to the server, thereby effectively avoiding privacy data leakage through the non-aggregation of raw data.

A potential solution to the second challenge is evolutionary algorithms (EAs), a class of global optimization algorithms that mimic biological evolution mechanisms to solve complex problems through processes such as natural selection, genetic variation, and population iteration. Indeed, cyberattack variants essentially involve combinatorial variation of features, which aligns well with the crossover and mutation mechanisms of biological genes. As such, evolutionary algorithms provide an effective approach to enhancing the ability of the model to detect cyberattack variants by leveraging these evolutionary principles.

In this paper, we propose evolutionary replay-driven federated CIL for cyber-attack detection. The main contributions of this paper are as follows:

1) To protect privacy data, we propose an evolutionary algorithm-driven feature replay-based CIL method within a federated learning framework. This approach stores prototypical samples and evolves them through evolutionary algorithms during the new task learning phase to generate old knowledge, thereby mitigating the model's catastrophic forgetting issue.

[1] https://openai.com/chatgpt/

[2] https://www.deepseek.com/

2) To enhance the model's capability to detect attack variants, we develop an attack variant generation method driven by evolutionary algorithms. By leveraging gene evolution strategies and CAEs from prototypical samples, this method generates attack variants to improve detection performance on evolving cyberattack patterns.
3) We construct a cyber-attack detection model based on deep residual networks and conduct extensive experiments on public datasets. The results demonstrate that our proposed method achieves an accuracy of 90.16% in Task 2 and 85.90% in Task 3. Notably, in Task 3, it outperforms the random replay method by 4.66%, the GAN-based replay method by 6.91%, and the VAE-based replay method by 22.32%.

The remainder of this paper is structured as follows: **Section 2** reviews the latest research on replay-based CIL and federated CIL. **Section 3** describes in detail the proposed method. **Section 4** presents the experimental design and results. Finally, **Section 5** concludes the paper and discusses future work.

2 Related Work

Replay-based CIL is widely regarded as the most effective approach to addressing the catastrophic forgetting problem in models. Meanwhile, to protect privacy data, researchers have proposed federated CIL methods. For example, Mao et al. [8] introduced a novel Hierarchical Federated Incremental Learning for Network Intrusion Detection (HFIN) method. Chen et al. [9] proposed a federated CIL framework named Generative Federated CIL (GenFCIL), which introduces a lightweight generator to facilitate knowledge sharing among clients and preserve cumulative knowledge from all clients. Dong et al. [10] developed a novel Local-Global Anti-Forgetting (LGA) model to address catastrophic forgetting at both local and global levels. Lu et al. [11] presented FCIDF, a new federated CIL method based on dynamic feature extractor fusion. FCIDF integrates global knowledge into local features through personalized fusion rates, enabling each client to learn a personalized incremental model. By leveraging meta-learning in each incremental learning round, FCIDF ensures that knowledge from both old and new tasks is incorporated into personalized training. Zhang et al. [12] proposed Cross-FCL, a cross-edge federated continuous learning framework that uses a parameter-decomposition-based federated continuous learning model to allow devices to retain previously learned knowledge while participating in new task training. Li et al. [13] introduced Re-Fed+, a general and low-cost Federal Incremental Learning (FIL) framework designed to help clients cache important samples for replay.

3 Method

In this section, we first introduce the overall framework of the proposed evolutionary replay-driven federated CIL for cyber-attack detection. Second, we describe the selection of exemplar samples. Third, we explain the generation of replay samples, where a

genetic evolution strategy is introduced to produce variant samples with evolutionary characteristics. Finally, we detail the complete workflow of incremental learning in the federated learning scenario.

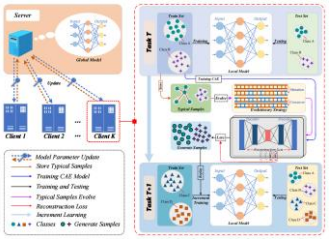

Fig. 1. Overall Framework

3.1 Overall Framework

The overall framework of the evolutionary replay-driven federated CIL for cyber-attack detection is illustrated in **Fig. 1**. It consists of three main components: exemplar sample selection, replay sample generation and federated learning training.

(1) Exemplar Sample Selection: Upon receiving a new task each time, the local client saves a certain number of exemplar samples for each class of data into an exemplar memory bank. In subsequent tasks, these exemplars are used to generate replay samples for past tasks.

(2) Replay Sample Generation: Using the genetic evolution strategy and a CAE model, replay samples are generated from the exemplars in the memory bank. This process aims to alleviate catastrophic forgetting in incremental learning and helps the model adapt to the evolutionary characteristics of cyber-attacks.

(3) Federated Learning Training: The framework primarily consists of a server and multiple local clients. The server aims to communicate with each client, aggregate their model parameters, train a global model, and distribute this global model to all clients for detecting cyber-attacks across all datasets. The primary tasks of the client include receiving data, storing exemplar samples, training the CAE and local models, interacting with the server to update the local models. Upon receiving new training data, the client evolves the stored exemplar samples from past tasks to generate variant samples, and trains a new local model to meet the requirements of incremental learning.

We will describe these steps in detail below. It is important to note that this paper assumes both clients and the server are honest. Specifically, clients strictly follow the configured protocol for local model training, accurately upload their model parameters, The server also adheres to honest behavior. It correctly receives model updates from all clients, properly aggregates the global model.

3.2 Exemplar Sample Selection

Exemplar samples refer to those that are most representative of a specific category and can capture the core characteristics of that category. In incremental learning, selecting representative exemplar samples is critical for retaining knowledge from previous tasks. To identify these exemplar samples, we propose a selection method based on

distance and density weighting, and store them in an exemplar memory bank for subsequent replay sample generation.

(1) Cluster Partitioning. For samples of each category, we use the K-means clustering algorithm to divide them into three clusters. The cluster centers, obtained by calculating the geometric mean of samples within each cluster, represent the feature distribution centers of the clusters.

(2) Distance and Density Calculation. For samples in each cluster, we first compute the Euclidean distance between each sample and the cluster center. The Euclidean distance measures the similarity between a sample and the cluster center, with samples closer to the center better representing the cluster's core features. Additionally, we introduce sample density as a weighting factor, as higher-density samples are more likely to reside in the cluster's core region. Combining these two factors, we calculate a weighted score for each sample as follows:

$$Score(x) = \alpha * \frac{1}{d(x,C)} + \beta * density(x) \quad (1)$$

where, $d(x, C)$ denotes the Euclidean distance between sample x and cluster center C, $density(x)$ represents the density of sample x, and α , β are hyperparameters that balance the weights of distance and density in the selection of exemplar samples. It is important to note that both distance and density values are normalized prior to computing the weighted score to eliminate scale differences.

(3) **Selection.** Exemplar samples are selected by integrating distance and density metrics. We rank the samples within each cluster based on their weighted scores and select the top-k samples. These selected samples, covering both cluster centers and high-density core regions, maximize the retention of core knowledge from previous tasks and provide a high-quality data foundation for subsequent replay sample generation.

3.3 Replay Sample Generation

To better capture the evolutionary characteristics of cyber-attacks and enhance the generalization ability of the model, we propose a genetic evolution strategy-based method for generating replay samples. The genetic evolution strategy simulates biological evolutionary processes, drawing inspiration from natural selection and genetic principles. Through this strategy, we simulate the evolutionary process of cyber-attacks to produce diverse attack variants, thereby improving the model's ability to recognize evolving attack types.

Next, we detail the replay sample generation process, which comprises two main steps: generating variant samples via evolution strategies, and selecting high-quality variants.

(1) Evolution Strategies. We employ three genetic evolution strategies to generate variant samples: Single Gene Mutation (SGM), Multi-point Crossover (MPC), and Differential Evolution (DE).

a. SGM. Single Gene Mutation involves randomly selecting one feature value in a sample for mutation. For the mutated gene, a new random value is generated based on the allowable range of the original feature. Specifically, a position j is randomly selected

from the feature vector of the current sample, and its value is mutated within the pre-defined range associated with that feature. The implementation process is described in detail in **Algorithm 1.**

Algorithm 1 SGM

Input: Sample *S*, Previous labels *Y*, Feature dimension *D*
Output: $SGM_{offspring}$

Initialize:
  $\mathcal{R} \leftarrow$ Initial Range Dictionary (6, *D*)
if y = 0 **then**
  $\mathcal{R}[0][0] \leftarrow (min_{0,0}, max_{0,0})$; $\mathcal{R}[0][1] \leftarrow (min_{0,1}, max_{0,1})$...
else if y = 1 **then**
  $\mathcal{R}[1][0] \leftarrow (min_{1,0}, max_{1,0})$; $\mathcal{R}[1][1] \leftarrow (min_{1,1}, max_{1,1})$...
end if
for i ← 1 to |*S*| **do**
  f ~ U(0, *D*-1)
  y ← *Y*[i]
  (min, max) ← $\mathcal{R}$[y][f]
δ ← min + (max-min)×U(0,1)
  $SGM_{offspring}$[i,f] ← δ
end for
return $SGM_{offspring}$

b. MPC. Multi-point Crossover refers to generating new samples by exchanging features between two samples of different classes, i.e., swapping gene segments within the feature space. Specifically, two samples $mal_a^{parent} = [g_a^1, g_a^2, \ldots, g_a^M]$ and $mal_b^{parent} = [g_b^1, g_b^2, \ldots, g_b^M]$ are randomly selected, and a crossover point $\boldsymbol{g_a^i}$ is randomly chosen. The gene segments before the crossover point in mal_a^{parent} are combined with the gene segments after the crossover point in mal_b^{parent}, and vice versa. This results in two new variant samples mal_a^{child} and mal_b^{child}, which inherit partial genetic characteristics from the parent samples. The implementation process is described in detail in **Algorithm 2**.

Algorithm 2 MPC

Input: G_{SGZ}, Population size *N*, Crossover points C_n
Output: $MPC_{offspring}$

Initialize:
  $MPC_{offspring} \leftarrow \emptyset$; $\mathcal{L} \leftarrow$ Unique(G_{SGZ}.labels)
for k = 1 to *N* **do**
  Randomly select $L_i, L_j \in L$ where i≠j
  $\mathcal{G}^{L_i}, \mathcal{G}^{L_j} \leftarrow$ RandomSelection(G_{SGZ}, L_i, L_j)
    CP ← Sort(RandomSample(1, $|\mathcal{G}| - 1$, C_n)); $\boldsymbol{CP} \leftarrow [0] \cup \boldsymbol{CP} \cup [|\mathcal{G}|]$
for m = 1 to |**CP**| − 1 **do**
  if m is odd **then**
    MPC_i[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$] ← $\mathcal{G}^{L_j}$[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$]
    MPC_j[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$] ← $\mathcal{G}^{L_i}$[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$]
  else
    MPC_i[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$] ← $\mathcal{G}^{L_i}$[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$]
    MPC_j[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$] ← $\mathcal{G}^{L_j}$[$\mathbf{CP}_m$: $\mathbf{CP}_{m+1}$]
  end if
end for

16: $MPC_{offspring} \leftarrow MPC_{offspring} \cup \{MPC_i, MPC_j\}$
17: **end for**
18: **return** $MPC_{offspring}$

c. DE. Differential Evolution is a population-based evolutionary algorithm that involves iterative operations such as mutation, crossover, and selection on a randomly initialized population. Specifically, two parent samples $mal_c^{parent} = [g_c^1, g_c^2, \ldots, g_c^M]$ and $mal_d^{parent} = [g_d^1, g_d^2, \ldots, g_d^M]$ are first randomly selected from the population. These parent samples undergo random mutation and crossover operations to generate new variant samples mal_c^{child} and mal_d^{child}. The implementation process is described in detail in **Algorithm 3**.

(2) **Sample Selection.** After applying the evolution strategies, a large number of cyber-attack variant samples are generated. However, some samples may not conform to the category requirements after evolution. To minimize the interference of such samples during local model training, it is necessary to perform a quality-based selection of the generated samples. Upon receiving new task data, we train a CAE for each class to learn and memorize the feature distribution of that class. After generating samples via genetic evolution strategies, the reconstruction error of the CAE is utilized to evaluate whether the newly generated samples align with the characteristics of the original class. Samples meeting the criteria are preserved. The detailed implementation process is described in **Algorithm 4**.

Algorithm 3 DE

Input: Population matrix $\mathbf{P} \in \mathbb{R}^{N \times D}$,
Scale factor $F \in [0, 2]$, Crossover rate $CR \in [0, 1]$
Output: $DE_{offspring}$

Initialize:
$NP, D \leftarrow \dim(\mathbf{P})$; $\mathbf{V} \leftarrow \text{zeros}(NP, D)$; $DE_{offspring} \leftarrow \text{zeros}(\text{N P}, \text{D})$
for i $\leftarrow$ 1 to NP **do**
  Randomly select $\{r_1, r_2, r_3\}$; $\mathbf{V}[i, :] \leftarrow \mathbf{P}[r_1, :] + F \cdot (\mathbf{P}[r_2, :] - \mathbf{P}[r_3, :])$
end for
for i $\leftarrow$ 1 to NP **do**
  $j_{rand} \leftarrow$ random integer in $[0, D-1]$
  for $j \leftarrow 1$ to D **do**
  if rand() $< CR$ or $j = j_{rand}$ **then** $U[i,j] \leftarrow V[i,j]$
  else $U[i,j] \leftarrow P[i,j]$
  end if
  end for
end for
$DE_{offspring} \leftarrow \text{clip}(DE_{offspring}, \text{lb}, \text{ub})$
return $DE_{offspring}$

3.4 Overall Workflow of Incremental Learning in Federated Scenarios

Our proposed method aims to achieve CIL in federated scenarios for cyber-attack detection. In this subsection, we detail the complete workflow.

(1) Problem Definition. In the federated learning framework, there is one central server S and K local clients, where each client receives T incremental tasks. In federated incremental learning, each task is independent and requires R communication rounds. The label sets of different tasks are disjoint. To mitigate catastrophic forgetting, we maintain an exemplar memory bank M to store samples of old classes. It is worth mentioning that the exemplar storage, CAE training per class, and replay sample generation occur only in the first communication round of each federated incremental task. Subsequent rounds focus solely on local model training.

(2) Local Model Training at Task T.

a. Local Model Architecture. In federated CIL, the model must handle dynamically shifting task data. However, deeper networks are at risk of suffering from gradient vanishing and degraded feature propagation. Additionally, varying feature distributions across tasks challenge traditional convolutional networks in dynamically adjusting channel-wise importance. To address these issues, we design a dual-attention residual network comprising CNN backbone, three residual blocks and dual-attention mechanisms (SE Attention and Spatial Attention).

Algorithm 4 Replay Sample Generation

Input: Samples $SGM_{offspring}$, $MPC_{offspring}$, $DE_{offspring}$,CAE models $M \in \{\ell$: model$\}$,Target counts $Q \in \mathbb{N}$, Thresholds $T \in \{\ell$: $\tau\}$,Labels L

Output: Replay samples G

Initialize: $G \leftarrow \emptyset$, $C \leftarrow \{\ell : 0 \mid \forall \ell \in L\}$;
while $\exists \ell \in L$ where $C[\ell] < Q$ **do**
  Generate candidates $O \leftarrow$ stack($SGM_{offspring}$, $MPC_{offspring}$, $DE_{offspring}$) ;
  foreach *sample* $o \in O$ **do**
    x $\leftarrow$ tensor($o_{features}$); $\mathcal{L} \leftarrow \{\}$;
    foreach $\ell \in L$ **do**
      $\hat{x} \leftarrow M[\ell]$(x);
      $\mathcal{L}[\ell] \leftarrow \| \hat{x} - \text{x} \|^2$;
  (ℓ_{min}, τ_{min}) $\leftarrow$ ($\mathcal{L}$);
  if $\tau_{min} < T[\ell_{min}]$ **and** $C[\ell_{min}] < Q$ **then**
    $G \leftarrow G \cup \{(o_{features}, \ell_{min})\}$;
    $C[\ell_{min}] \leftarrow C[\ell_{min}] + 1$;
return G

Initial Convolution Module. This module consists of convolutional layers, batch normalization, and $ReLU$ activation. It performs preliminary feature extraction and dimensional transformation, mapping heterogeneous data into a unified feature space for downstream processing.

Residual Blocks. Residual connections mitigate feature information loss and gradient vanishing, enabling efficient feature propagation. By directly passing original features to deeper layers, the network retains critical information while supporting depth scaling. The block structure is defined as:

$$H(x) = Conv1D(ReLU(BN(Conv1D(X_{in})))) \tag{2}$$

$$X_{sut} = ReLU(H(x) + Shortcut(X_{in})) \epsilon \mathbb{R}^{B \times C_{out} \times L/stride} \tag{3}$$

$$Shortcut(x) = \begin{cases} x, & if\ C_{in}=C_{out}\ and\ stride=1 \\ Conv1D(x), & otherwise \end{cases} \quad (4)$$

Attention Mechanisms. We propose the use of SE Attention and Spatial Attention to enhance feature representations from the channel and spatial dimensions, respectively, thereby improving the model's capability to detect cyber-attack variants. Specifically, SE Attention adaptively learns the importance weights of channels through global average pooling and fully connected layers, employing a squeeze-and-excitation mechanism to enable the model to focus on critical features. Spatial Attention learns spatial position weights by concatenating average-pooled and max-pooled features, thereby enhancing the model's representation of significant spatial regions. The specific descriptions of the modules are as follows:

$$Squeeze: z_c = \frac{1}{L}\sum_{l=1}^{L} X_{c,l} \quad (5)$$

$$Excitation: s = \sigma(W_2 . ReLU(W_1 \cdot z))\ (W_1 \epsilon \mathbb{R}^{C/r \times C}, W_2 \epsilon \mathbb{R}^{C \times C/r}) \quad (6)$$

$$SpatialWeight = \sigma(Conv1D(Concat)) \epsilon \mathbb{R}^{B \times 1 \times L} \quad (7)$$

b. Local Model Training. Each client operates independently when training its local model. Upon the arrival of Task T, the client receives new training data, categorizes it by class labels y, and trains a CAE for each class to filter replay samples. Additionally, the client selects exemplar samples from the current task and stores them in the exemplar memory bank for subsequent replay sample generation. Finally, the local model is trained using the architecture described earlier. The detailed implementation process is outlined in **Algorithm 5**.

Algorithm 5 Local Model Training

Input: Client ID *id*, Training data *D*, Global epoch *g*
Output: Updated local model parameters ω_t

Initialize:
Exemplar samples bank $\mathcal{M} \leftarrow \emptyset$; Extract features *X*, labels *Y* from *D*
if current Task T = 0 **then**
  if g = 0 **then**
    Train CAE; Select exemplar samples and store to $\mathcal{M}_{id}$
    $\omega_t \leftarrow$ Train_model(*X*, *Y*)
  else
    $\omega_t \leftarrow$ Train_model(*X*, *Y*)
  end if
  if g = 0 **then**
    Select exemplar samples and store to $\mathcal{M}_{id}$
    Gene Evolution: SGM, MPC, DE
    $X_{replay} \leftarrow D_k^{SGM} \cup D_k^{MPC} \cup D_k^{DE}$
    $X'_{replay} \leftarrow select\_replay\_samples(X_{replay})$
    $X' \leftarrow X \cup X'_{replay}, Y' \leftarrow Y \cup Y'_{replay}$
    $\omega_t \leftarrow$ Train_model(X', Y')
  else
    $\omega_t \leftarrow$ Train_model(X′, Y′)
  end if
end if

26: return ω_t

(3) Federated Training.

a. Key Generation.

In federated learning, to protect the privacy of client data, homomorphic encryption is typically used to secure model parameters, allowing the server to aggregate parameters in encrypted form. Since we assume both the server and clients are trustworthy, the server generates a key pair. The key generation process is as follows: let p and q be two large prime numbers satisfying $gcd(pq, (p-1)(q-1)) = 1, n = pq$. The public key is $\mathcal{PK} = (n, g)$, and the private key is $\mathcal{SK} = (\lambda, \mu)$,where $\lambda = lcm(p-1, q-1), \mu = (L(g^{\lambda} \bmod n^2))^{-1} \bmod n$. At the start of federated training, the server distributes the public key to all clients, while the private key is retained solely by the server.

b. Client Parameter Encryption.

After completing local model training, the client encrypts its local model parameters. Let ω_k^r denote the local parameters of client k, and r be a random number satisfying $r < N$. The client encrypts ω_k^r using the public key, generating the ciphertext, which is then uploaded to the server. The encryption formula is:

$$\widetilde{\omega_k^r} = E_{\mathcal{PK}}(\omega_k^r) = g^{\omega_k^r} r^N \bmod N^2 \tag{8}$$

c. Sever Parameter Aggregation.

The server collects and aggregates the parameters uploaded by all clients. Under encryption, the global model parameters are computed and distributed to all clients. The aggregation formula is:

$$\begin{aligned} S &= \prod_{i=1}^{K} = E_{\mathcal{PK}}(\widetilde{\omega_k^r}) = g^{\widetilde{\omega_k^r}} r_1^N \cdot g^{\widetilde{\omega_k^r}} r_2^N \cdot \ldots \cdot g^{\widetilde{\omega_k^r}} r_k^N \bmod N^2 \\ &= g^{\sum_{i=1}^{K} \widetilde{\omega_k^r}} \cdot \prod_{i=1}^{K} r_i^N \bmod N^2 \end{aligned} \tag{9}$$

After completing the aggregation, the server distributes the global model parameters to all clients, concluding one round of federated communication.

d. Client Parameter Update.

Upon receiving the encrypted global parameters from the server, each client decrypts them and updates its local model using the formula:

$$\begin{aligned} \omega_{sum}^r &= L(\widetilde{\omega_{avg}^r} \bmod N^2) \cdot \mu \bmod N \\ &= \frac{L(g^{\sum_{i=1}^{K} \widetilde{\omega_{avg}^r}} \cdot \prod_{i=1}^{K} r_i^N \bmod N^2)}{L(g^{\lambda} \bmod N^2)} \bmod N \\ &= \sum_{i=1}^{K} \widetilde{\omega_{avg}^r} \bmod N \end{aligned} \tag{10}$$

This marks the completion of one round of federated training interaction between the clients and the server. Each task undergoes R rounds of this process, ultimately yielding an optimal global cyber-attack detection model.

(4) Local Model Training at Task $T+1$. To mitigate catastrophic forgetting when the next task arrives, we generate replay samples using genetic evolution strategies. First, we retrieve the exemplar samples stored in the memory bank from previous tasks

and load the pre-trained CAEs for each historical class. Then, through genetic evolution strategies, a large number of replay samples are generated. These samples are filtered using the CAEs of each class to retain those that conform to the original class characteristics. Subsequently, the generated replay samples are combined with the newly received task data to train the local model. Finally, we preserve a certain number of exemplar samples from the current task for each class and update the exemplar memory bank. After completing the local model training, the federated training process described in step (3) is executed to update the global model.

4 Experiment

In this section, we detail the datasets, evaluation metrics, and experimental settings required for the experiments. We have evaluated the performance of our proposed method through extensive experiments. First, experiments were conducted on different datasets to validate the effectiveness of cyberattack detection in federated class-incremental scenarios. Next, we compared the experimental results under different replay methods, contrasting our proposed evolutionary strategy with random replay, GAN-based replay, and VAE-based replay.

4.1 Datasets

We evaluate our proposed method using the UNSW-NB15 [14] dataset and NLS-KDD [15] dataset. The UNSW-NB15 dataset is a public dataset specifically designed for research and evaluation of network intrusion detection systems. It covers common attack types in modern network environments, including 9 classes of attack traffic in addition to normal traffic. To mitigate the interference of class imbalance issues in experiments, we select the 6 most populous classes and divide them into 3 tasks with 2 classes per task. Finally, we partition the dataset into training and test sets at a 4:1 ratio. Specific setting results for the UNSW-NB15 dataset are shown in **Table 1.**

Table 1. Detailed information about the UNSW-NB15 dataset and its settings

Task	Type	Training Set	Testing Set
Task 1	Normal	74,400	18,600
	Generic	47,096	11,774
Task 2	Exploits	35,620	8,905
	Fuzzers	19,397	4,849
Task 3	DoS	13,081	3,270
	Reconnaissance	11,190	2,797

The NLS - KDD dataset is a classic dataset in the field of network intrusion detection. Similarly, we selected the 6 most populous classes, and divided them into 3 tasks with 2 classes per task. Due to the highly imbalanced class distribution in the NLS-KDD dataset, we applied downsampling: for the Normal and Neptune classes, we randomly

sampled $1/10$ of the data, while using all data for the remaining classes. The configuration results for the NLS-KDD dataset are presented in **Table 2.**

For each data set, we divided it into three CIL. In the federated learning framework, we initialized three local clients and set each task to train for 10 global rounds. In each global round, the local model is trained for 20 epochs using an Adam optimizer with a learning rate of 0.001 When training the class-specific CAE for each category, we set the number of iterations to 20. In the first global round of each incremental task, we store 500 samples in the prototypical sample library and generate replay samples for old tasks through the gene evolution strategy, with the number of generated replay samples matching the size of the largest class in the current task.

Table 2. Detailed information about the NLS-KDD dataset and its settings

Task	Type	Training Set	Testing Set
Task 1	Normal	6,734	1,347
	Neptune	4,121	824
Task 2	Satan	3,633	727
	Ipsweep	3,599	141
Task 3	Portsweep	2,931	156
	Sumrf	2,646	627

Since our research scenario assumes that data across local clients are independently and identically distributed (i.i.d.), we evenly distribute the data of each task among the clients to ensure that each client receives different data, thereby realizing data heterogeneity across local clients. During testing, however, we use a test set containing all classes to evaluate the globally aggregated model after federated aggregation, to check whether the model meets the requirements of incremental learning.

4.2 Experimental environment and evaluation criteria

Our experimental environment is summarized as follows: The processor is a 12th Gen Intel(R) Core(TM) i7-12700K, and the GPU is an NVIDIA[3] GeForce RTX 3090 Ti with 24GB RAM. The programming language used is Python[4] 3.9 for Windows 11, the neural network framework is PyTorch[5] version 2.5.0, and the federated learning framework is also built on PyTorch.

We primarily use the following five metrics to evaluate our proposed method: Accuracy, Precision, Recall, False Positive Rate (FPR), and F1 (the weighted average of precision and recall). These metrics comprehensively assess the model's performance on the test set from different perspectives

[3] https://www.nvidia.cn/

[4] https://www.python.org/

[5] https://www.pytorch.org/

4.3 Experimental result

(1) Experimental Results on Different Datasets

Table 3 and **Table 4** present the performance of our method on the UNSW and NLS-KDD datasets. The experimental results show that: First, during Task 1, without generating replay samples and relying solely on local models and federated learning for global model training, the model achieves an accuracy exceeding 99% on both datasets. As Task 2 proceeds, the model begins to forget previous tasks, leading to a decrease in accuracy. However, our approach of generating replay samples through the gene evolution strategy ensures a high detection accuracy at the end of Task 2. For the UNSW dataset, the detection accuracy is 88.78%, and the F1 score is 0.8944. For the NLS-KDD dataset, the accuracy is 90.16%, with an F1 score of 0.9058. By Task 3, the model's forgetting intensifies, causing a further drop in accuracy. Nonetheless, our method effectively mitigates catastrophic forgetting, achieving F1 scores of 0.7655 and 0.8563 for the UNSW and NLS-KDD datasets, respectively.

Table 3. Experimental results on the UNSW dataset

Task	R	Accuracy↑	Precision(%)↑	Recall(%)↑	F1↑	FPR↓
	2	0.9980	99.80	99.79	0.9979	0.0021
	4	0.9975	99.74	99.74	0.9974	0.0026
1	6	0.9988	99.90	99.85	0.9988	0.0015
	8	0.9950	99.41	99.54	0.9947	0.0046
	10	**0.9950**	99.41	99.54	**0.9947**	0.0046
	2	0.8770	87.03	92.87	0.8857	0.0386
	4	0.8873	87.88	93.46	0.8940	0.0350
2	6	0.8953	88.34	93.86	0.8994	0.0322
	8	0.8878	87.88	93.51	0.8944	0.0349
	10	**0.8878**	87.88	93.51	**0.8944**	0.0349
	2	0.8047	74.83	85.86	0.7638	0.0368
	4	0.8041	74.65	85.56	0.7628	0.0371
3	6	0.7893	73.67	84.25	0.7473	0.0399
	8	0.8084	74.92	85.94	0.7655	0.0361
	10	**0.8084**	74.92	85.94	**0.7655**	0.0361

(2) Compare the experimental results under different replay methods

On the NLS-KDD dataset, we compared the proposed method with three traditional generative replay approaches: random replay, GAN-based generative replay, and VAE-based generative replay.

Random replay [16] is the most basic strategy, which involves directly storing a small number of real samples from old tasks and randomly selecting the required quantity to mix with the current task's data during new task training, thereby helping the model

retain memory of old knowledge. However, its drawbacks include requiring sufficient storage space and potential issues with insufficient samples from old tasks.

GAN-based generative replay [17] involves designing a Conditional Generative Adversarial Network (CGAN) to generate replay samples, which primarily consists of a generator and a discriminator. The generator takes as input a noise vector with $noise\_dim = 130$ and the one-hot encoding of class labels. During adversarial training, the discriminator uses binary cross-entropy loss (BCELoss) with label smoothing, and the Adam optimizer with hyperparameters $learning\ rate = 0.0002$, $\beta_1 = 0.5$, and $\beta_2 = 0.99$. In the generation phase, the model generates a specified number of samples for each old class, ultimately outputting the feature data and their corresponding class labels.

Table 4. Experimental results on the NLS-KDD dataset

Task	R	Accuracy↑	Precision(%)↑	Recall(%)↑	F1↑	FPR↓
	2	0.9926	99.22	99.22	0.9922	0.0078
	4	0.9926	99.24	99.19	0.9922	0.0081
1	6	0.9945	99.39	99.44	0.9941	0.0056
	8	0.9940	99.35	99.38	0.9936	0.0062
	10	**0.9940**	99.35	99.38	**0.9936**	0.0062
	2	0.8861	84.61	93.16	0.8868	0.0366
	4	0.8736	84.01	92.41	0.8801	0.0409
2	6	0.8819	85.31	92.94	0.8896	0.0382
	8	0.8967	86.58	93.68	0.8999	0.0335
	10	**0.9016**	87.42	93.99	**0.9058**	0.0318
	2	0.8619	81.82	91.99	0.8661	0.0273
	4	0.8634	82.21	92.25	0.8694	0.0272
3	6	0.8561	80.14	91.52	0.8545	0.0285
	8	0.8590	80.35	91.66	0.8563	0.0278
	10	**0.8590**	80.35	91.66	**0.8563**	0.0278

VAE-based generative replay [18] involves designing a Variational Autoencoder (VAE) model to perform data distribution modeling and generate samples from old tasks through latent space sampling. This model mainly consists of an encoder and a decoder. During the training phase, we use an Adam optimizer with a $learning\ rate = 0.0001$ and a $weight\ decay = 1e-5$. The loss function is defined as the weighted sum of the reconstruction loss and the KL divergence: $Loss = BCE(recon_x, x) + 0.5 \times KL(mean, log_{var})$, In the generation phase, we randomly sample from the latent distributions corresponding to each old class and decode them to generate replay samples. This method explicitly models the probabilistic characteristics of the data distribution and combines the conditional generation mechanism to ensure that the generated samples are highly consistent with the samples of the old classes in both the feature space and the class distribution.

The experimental results are shown in **Table 5.** The results indicate that the method we proposed consistently maintains the highest accuracy. Specifically, in Task 3, our proposed method outperforms the random replay method by 4.66%, the GAN-based replay method by 6.91%, and the VAE-based replay method by 22.32%. This difference demonstrates the effectiveness of the prototypical samples we retained, which can preserve the spatial distribution of the old tasks. In addition, due to its unique biological evolution mechanism, by simulating gene mutations and recombination's in the process of natural selection, it retains the most discriminative features of the old tasks and has a strong ability for variant evolution.

Table 5. The experimental results are compared with different generation replay methods on the NLS-KDD dataset

T	R	Acc.	Pre.(%)	Rec.(%)	F1	Acc.	Pre.(%)	Rec.(%)	F1
		Random				**CGAN**			
	2	0.9862	98.87	98.18	0.9852	0.9724	97.77	96.43	0.9703
	4	0.9940	99.35	99.38	0.9936	0.9931	99.28	99.25	0.9927
1	6	0.9945	99.44	99.39	0.9941	0.9936	99.29	99.34	0.9932
	8	0.9940	99.38	99.35	0.9936	0.9936	99.32	99.32	0.9932
	10	0.9945	99.39	99.44	0.9941	0.9940	99.33	99.40	0.9936
	2	0.9945	99.39	99.44	0.9941	0.9940	99.33	99.40	0.9936
	4	0.8690	83.09	87.58	0.8527	0.8473	85.97	90.95	0.8635
2	6	0.8486	85.96	86.65	0.8630	0.7792	82.41	87.10	0.8067
	8	0.8628	86.92	87.79	0.8735	0.7838	83.90	87.41	0.8184
	10	0.8602	86.51	87.45	0.8698	0.7897	84.22	87.72	0.8235
	2	0.8644	87.01	87.92	0.8746	0.7828	83.63	87.32	0.8160
	4	0.8644	87.01	87.92	0.8746	0.7828	8363	87.32	0.8160
3	6	0.8048	79.93	86.06	0.8288	0.8014	80.50	89.52	0.8154
	8	0.8140	79.92	87.31	0.8345	0.7988	80.19	89.56	0.8089
	10	0.8187	80.64	87.76	0.8405	0.7729	77.86	88.25	0.7778
		VAE				**Our Method**			
	2	0.9562	96.67	94.26	0.9524	**0.9802**	**98.36**	**97.39**	**0.9787**
	4	0.9931	99.28	99.25	0.9927	**0.9926**	**99.22**	**99.22**	**0.9922**
1	6	0.9931	99.28	99.25	0.9927	**0.9926**	**99.24**	**99.19**	**0.9922**
	8	0.9936	99.32	99.32	0.9932	**0.9945**	**99.39**	**99.44**	**0.9941**
	10	0.9926	99.24	99.19	0.9922	**0.9940**	**99.35**	**99.38**	**0.9936**
	2	0.9926	99.24	99.19	0.9922	**0.9940**	**99.35**	**99.38**	**0.9936**
	4	0.8006	80.52	88.27	0.8060	**0.8776**	**81.05**	**92.58**	**0.8643**
2	6	0.7723	81.05	86.69	0.7943	**0.8861**	**84.61**	**93.16**	**0.8868**
	8	0.7845	82.81	87.33	0.8124	**0.8736**	**84.01**	**92.41**	**0.8801**
	10	0.7805	81.54	87.13	0.8021	**0.8819**	**85.31**	**92.94**	**0.8896**

	2	0.7887	82.17	87.59	0.8105	**0.8967**	**86.58**	**93.68**	**0.8999**
	4	0.7887	82.17	87.59	0.8105	**0.9016**	**87.42**	**93.99**	**0.9058**
3	6	0.7745	78.02	87.17	0.7857	**0.8587**	**80.12**	**92.11**	**0.8570**
	8	0.5667	72.50	74.05	0.5969	**0.8619**	**81.82**	**91.99**	**0.8661**
	10	0.5633	72.43	73.56	0.5916	**0.8634**	**82.21**	**9225**	**0.8694**

It is worth noting that the False Positive Rate (FPR) of the gene evolution method remains at the lowest level in all tasks, indicating that the replay samples it generates can accurately maintain the boundary characteristics between normal behavior and network attacks. This consistent and stable performance shows that the replay method based on the evolutionary strategy can effectively alleviate the problem of catastrophic forgetting of the model in incremental learning.

5 Conclusion

To mitigate the issues of catastrophic forgetting in models and the leakage of private data, we propose evolutionary replay-driven federated class-incremental learning for cyber-attack detection. The experimental results show that in the incremental tasks on different datasets, the F1 scores of our proposed method reach 0.8944, 0.7655, 0.9058, and 0.8563 respectively. Compared with the random replay, CGAN, and VAE methods, in Task 3, our method outperforms the random replay method by 4.66%, the GAN-based replay method by 6.91%, and the VAE-based replay method by 22.32%. Although we have achieved the detection of cyber-attack variants in the federated class-incremental scenario, there are still the following limitations and drawbacks. For example: (1) In the federated process, we assume that both the clients and the server are honest. However, in practical applications, the clients and the server may be subject to malicious attacks and exhibit dishonest behaviors. (2) When dividing the task data for each client, we assume by default that the data of each client is independently and identically distributed. Therefore, our future research directions will focus on the scenarios of federated learning poisoning and non-independently and identically distributed client data.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China (No. U24A20239, No. 62032002, No. 62402300, No. 62366052); in part by the Sichuan Provincial Science and Technology Department regional innovation cooperation key project (No. 2025YFHZ0265); in part by the Youth Science Foundation of Sichuan (No. 2025ZNSFSC1474); in part by the Sichuan Province Science and Technology Innovation Seedling Project (No. MZGC20240056)

References

1. Check Point Software.https://www.globenewswire.com/news-release/2025/01/14/3009378/0/en/Check-Point-Software-s-2025-Security-Report-Finds-Alarming-44-Increase-in-Cyber-Attacks-Amid-Maturing-Cyber-Threat-Ecosystem.html.Accessed Jan 2025.
2. Chen, H., Wang, Y., Hu, Q.: Multi-Granularity Regularized Re-Balancing for Class Incremental Learning. IEEE Trans. Knowl. Data Eng. **35**(7), 7263-7277(2023)
3. Yang, Y., Sun, Z., Zhu, H., Fu, Y., Zhou, Y., Xiong, H., Yang, J.: Learning Adaptive Embedding Considering Incremental Class. IEEE Trans. Knowl. Data Eng. **35**(3), 2736-2749 (2023)
4. Mehrnoosh, M., Mohammad, R., Aram, G.: History Repeats: Overcoming Catastrophic Forgetting for Event-Centric Temporal Knowledge Graph Completion. In: Proceedings of the Association for Computational Linguistics (ACL), Toronto, Canada, pp. 7740-7755(2023)
5. Dusan, V., Ondrej, B.: Unsupervised Pretraining for Neural Machine Translation Using Elastic Weight Consolidation. In: 2019 57th Association for Computational Linguistics (ACL), Florence, Italy, pp. 130–135(2019)
6. Sylvestre, A. R., Alexander, K., Georg, S. Christoph, H. L.: Icarl: Incremental Classifier and Representation Learning. In: Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Hawaii, USA, pp. 5533-5542(2017)
7. H. Brendan, M., Eider, M., Daniel, R., Seth, H., Blaise, A. Y. A.: Communication-efficient learning of deep networks from decentralized data. In: Proceeding the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 1273–1282(2017)
8. Mao, J., Wei, Z., Li, B., Zhang, R., Song, L.: Toward Ever-Evolution Network Threats: A Hierarchical Federated Class-Incremental Learning Approach for Network Intrusion Detection in IIoT. IEEE Internet Things J., 11(18), 29864-29877(2024)
9. Chen, Y., Alysa, Z. T., Feng, S., Yu, H., Deng, T., Zhao, L.: General Federated Class-Incremental Learning With Lightweight Generative Replay. IEEE Internet Things J., **11**(20), 33927-33939(2024)
10. Dong, J., Li, H., Cong, Y., Sun, G., Zhang, Y., Luc, V. G.: No One Left Behind: Real-World Federated Class-Incremental Learning. IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI), **46**(4), 2054-2070(2024)
11. Lu, Y., Yang, L., Chen, H. R., Cao, J., Lin, W., Long, S.: Federated Class-Incremental Learning With Dynamic Feature Extractor Fusion. IEEE Trans. Mob. Comput., **23**(12), 12969-12982(2024)
12. Zhang, Z., Guo, B., Sun, W., Liu, Y., Yu, Z., Cross-FCL: Toward a Cross-Edge Federated Continual Learning Framework in Mobile Edge Computing Systems, IEEE Trans. Mob. Comput., **23**(1), 313-326(2024)
13. Li, Y., Wang, H., Qi, Y., Liu, W., Li, R.: Re-Fed+: A Better Replay Strategy for Federated Incremental Learning, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI), (Early Access)
14. UNSW-NB15 Dataset: https://research.unsw.edu.au/projects/unsw-nb15-dataset
15. NSL-KDD Dataset: https://www.unb.ca/cic/datasets/nsl.html
16. Ahmet, Y., Ozlem D. I.: Class-Incremental Continual Learning for Human Activity Recognition with Motion Sensors. In: Proceeding of the 32nd Signal Processing and Communications Applications Conference (SIU), Mersin, Turkiye, pp.1-4(2024)
17. Aishwarya, A., B., Biplab B, Fabio, C., Subhasis, C.: Semantics-Driven Generative Replay for Few-Shot Class Incremental Learning. In: Proceeding of the 30th ACM International Conference on Multimedia (ACM'MM), pp.5246-5254(2022)

18. Gabriela, S., Karla, S.: Feedback-Driven Incremental Imitation Learning Using Sequential VAE. In: 2022 IEEE International Conference on Development and Learning (ICDL), London, United Kingdom, pp.238-243(2022)