SPIDER: A Semi-Supervised Continual Learning-based Network Intrusion Detection System

Suresh Kumar Amalapuram
Indian Institute of Technology Hyderabad
cs19resch11001@iith.ac.in

Bheemarjuna Reddy Tamma
Indian Institute of Technology Hyderabad
tbr@cse.iith.ac.in

Sumohana S. Channappayya
Indian Institute of Technology Hyderabad
sumohana@ee.iith.ac.in

***Abstract*—Network intrusion detection (NID) aims to identify unusual network traffic patterns (distribution shifts) that require NID systems to evolve continuously. While prior art emphasizes fully supervised annotated data-intensive continual learning methods for NID, semi-supervised continual learning (SSCL) methods require only limited annotated data. However, the inherent class imbalance (CI) in network traffic can significantly impact the performance of SSCL approaches. Previous approaches to tackle CI issues require storing a subset of labeled training samples from all past tasks in the memory for an extended duration, potentially raising privacy concerns. The proposed Semisupervised Privacy-preserving Intrusion detection with Drift-aware continual LEaRning (SPIDER) is a novel method that combines gradient projection memory (GPM) with SSCL to handle CI effectively without the requirement to store labeled samples from all of the previous tasks. We assess SPIDER's performance against baselines on six intrusion detection benchmarks formed over a short period and the Anoshift benchmark spanning ten years, which includes natural distribution shifts. Additionally, we validate our approach on standard continual learning image classification benchmarks known for frequent distribution shifts compared to NID benchmarks. SPIDER achieves comparable performance to fully supervised and semisupervised baseline methods, while requiring a maximum of 20% annotated data and reducing the total training time by 2X.**

***Index Terms*—network intrusion detection, continual learning, gradient projection memory, class imbalance, distribution shift**

I. Introduction

Distribution shift, also known as concept drift (CD) [1], is a natural phenomenon in many real-world problems, including cybersecurity [2]. Often, the cause for such distribution shifts is the changes in the hidden context of the concept class not known apriori [3]. In a nutshell, real-world data is not always independent and identically distributed (*i.i.d.*), and concept drift happens gradually over time [4]. Specifically, we are interested in CD of network intrusion detection data in this work. Network intrusion detection (NID) is a form of data drift detection that represents anomalous (abnormal) patterns in the network traffic [5]. This abnormality may include unauthorized access and penetrating activities on computing systems over the network [6]. CD arises in network traffic due to changes in the user/intruder behavior pattern, software updates, etc [2]. Consequently, our desiderata become building a network intrusion detection system (NIDS) that evolves *continuously* to preserve old abnormal patterns while adapting to new knowledge. These desiderata can be realized using *continual learning* (CL) [7] for building NIDS. CL is defined as a class of machine learning (ML) algorithms that mimic the human way of learning as new tasks arrive, each with distribution shifts [7, 8]. Each task may represent a subset of training data containing a distribution shift to be learned. A major performance bottleneck in the CL framework is *catastrophic forgetting* (CF): an abrupt loss in the performance of the previously learned task. Today, the CL framework solves various problems in domains like computer vision, natural language processing, etc. However, its application to NID is under-explored.

NID as a binary classification problem (BCP): NID is typically trained on normal data using zero positive learning [9], which makes it immune to malicious behavior drift. However, its performance can suffer when the distribution of *normality shifts* [10]. Here, normality is the benign (majority) class traffic. Normality shifts can happen with new patches, software updates, devices, or protocols [2, 10]. So, we pose NID as a CL-based BCP to adapt to normality shifts and identify known anomalies/attacks, which is particularly helpful in differentiating out-of-distribution normal samples from known intrusions. However, posing NID as a supervised BCP has two main challenges: *class imbalance* (CI) and the requirement of the large volume of *annotated data*. To reduce the amount of labeled data required, we focus on the CL-based semi-supervised binary classification problem in this work. In this context, CI refers to a situation where each class in a task has a varying number of training examples (n_t). The majority class contains the highest number of training examples, while the minority classes are the last few classes when ranked based on n_t.

Previous research [11, 12] on CL-based supervised binary NID problem has shown the detrimental effects of class imbalance on the CF of intrusion detection tasks and advocates the usage of memory replay (MR) based algorithms to mitigate CF. However, [11] requires access to large volumes of annotated data. Despite the best efforts to automate the annotation process, it still requires human intervention to prevent incorrect and missing annotations. These annotation errors will make the learning system performance biased/impacted [13]. Under such scenarios, semi-supervised training methods could advance the learning with limited annotated data [14] utilizing the abundant unlabeled data.

While memory-replay (MR) based techniques handle the adverse effects of the CI on catastrophic forgetting, they require storing a subset of training samples from *all of the previous tasks* for an extended duration, which causes privacy preservation issues. In NID, data privacy refers to storing a subset of *labeled training samples of all past tasks* for a long time in memory, potentially exposing the diversity of real-world traffic to *adversarial attacks*. For instance, adversarial attacks like *gradient revision* [15] on the buffer memory (of CL method) lead to an increase in forgetting past task knowledge. However, incorporating adversarial training to combat adversarial attacks into the memory-based CL methods may lead to accelerated forgetting [16] problem. Unlike images, crafting an adversarial example on NID data with imperceptible changes is difficult without the class label. Based on this intuition and motivated by the previous attacks on buffer memory, our work stores only past task samples without labels in the memory. Besides privacy concerns, the MR approaches require a Memory Reorganization Policy (MRP) to accommodate newly arriving training samples by replacing some old samples in a fixed-size buffer memory. Previous work [17] has shown the effectiveness of such an MRP on the detection performance

of the NIDS. So, careful attention is needed to choose an appropriate policy.

TABLE I: Comparing the time taken by the memory reorganization policy (MRP Time) with the total training time using CBRS algorithm [18] on different NID benchmark datasets. Reported timing values are in seconds measured using the wall clock time.

Dataset	MRP Time	Total Train Time	MRP Time Proportion
CICIDS-2017 [19]	29	371	7.8%
KDDCUP'99	24	428	5.6%
UNSW-NB15 [20]	19	502	3.7%
CSE-CICIDS-2018 [19]	75	1388	5.4%
AnoShift [2]	84	1448	5.8%

Further, MRP may induce additional computational complexity for large-scale training, as illustrated in Table I for different NID datasets with class balanced reservoir sampling (CBRS [18]). CBRS is a fully supervised algorithm that uses a finite-size buffer memory to store a subset of all the past tasks and replaces the old samples with newly arriving ones depending upon the class imbalance ratio. The additional time required for MRP is nearly in the range of 4 to 8%, which may increase further with the size of the training dataset.

Motivated by data privacy and scaling issues of MRP, the proposed method (SPIDER) uses gradient projection memory (GPM) [21] as a substitute to represent all previous task exemplars and a finite buffer that stores only the *unlabeled* training samples from the previous task. GPM [21] contains the bases of the representations (neural network activations) learned for each task. We will take gradient update directions orthogonal to GPM at each training step to preserve the previously learned knowledge. After finishing the training with the task 't,' we will *replace* the entire buffer memory with the subset of randomly chosen unlabeled samples from the task 't', which does not require any MRP.

In summary, our key contributions are as follows:

- To the best of our knowledge, this is the first work that studies challenges when formulating the NID problem in the SSCL setting. These challenges include adapting to concept drift, training with limited annotated data, data privacy, and class imbalance.
- We introduce a novel method dubbed Semisupervised Privacy-preserving Intrusion detection with Drift-aware continual lEaRning (SPIDER) that leverages gradient projection memory [21] in conjunction with finite buffer memory that uses only unlabeled training samples from the previous task.
- We evaluate and compare the proposed method with five baseline methods covering diverse families of different CL methods using six standard network intrusion detection benchmarks (KDDCUP'99, NSL-KDD, CICIDS-2017 [19], UNSW-NB15 [20], CSE-CICIDS-2018 [19], and AnoShift [2]) and three standard CL image classification benchmarks (MNIST, CIFAR-10 [22], and CIFAR-100 [22]). We find that the SPIDER's performance on all these benchmarks is consistent and is on par with that of the baselines, with a maximum of 20% annotated data ensuring improved data privacy with reduced training time.

The rest of this paper is structured as follows: Section II presents the related work, while Section III explains the relevant mathematical foundations and notations used in the paper. In Section IV, we delve into the problem formulation, the end-to-end training process, and the pseudo-code of the proposed SPIDER method. The experimental setup necessary for conducting the experiments is detailed in Section V, and then Section VI discusses the performance results, computation complexity, and hyperparameter sensitivity on the robustness of the SPIDER through ablation study followed by the conclusion remarks.

II. Related work

Intrusion detection (ID): Network intrusion differs from normal system behavior [5], so NID can be characterized by the distribution drift in the network traffic [23] as the malicious attacks are continuously evolving [24]. This research aims to develop a NIDS that can adapt to changes in the distribution of normal and attack traffic. Network intrusions constitute only a small portion of overall network traffic [25]; consequently, an inherent class imbalance is present in their representative datasets. Various sampling (and algorithmic) approaches have been proposed in the literature, but this work does not use them.

Continual learning (CL): Learning from a sequence of tasks to mimic human learning is the hallmark of CL [7]. Aimed at mitigating the CF, these methods are broadly categorized into regularization-based methods [26], expansion-based methods, memory/rehearsal-based approaches [27], and class imbalance-based methods [18]. The orthogonal projection (OP) based approach updates the model parameters in the direction orthogonal to the past tasks gradients. Specifically, the gradient projection memory (GPM) [21] method stores the bases of representations of past tasks and updates the model orthogonal to the bases. Unlike prior works, we use GPM [21] and buffer memory to handle class imbalance in this work.

Pseudo labeling in semi-supervised learning: Lee et al. [28] introduced a pseudo-labeling approach based on the *entrophy regularization* to benefit from unlabeled data. The basis for such a strategy relies on the *cluster assumption* or equivalently *low-density separation*. The low-density separation between the classes minimizes the class labels' conditional entropy on the unlabeled data.

Semi-supervised continual learning (SSCL) methods generally exploit the unlabeled data. In particular, ORDisCo [29] uses conditional GAN to generate samples for replay and avoids CF of unlabeled data by stabilizing the parameters of the discriminator. Distillmatch [30] is built on data augmentation, pseudo-labeling, and consistency regularization. PlCIL [31] uses a model trained on a previous task to generate pseudo-labels for newly arriving unlabeled data and avoids CF using buffer memory. In stark contrast, our work focuses on *class imbalance* and *data privacy* in the SSCL setting, which the prior works ignored.

Continual learning in security: CL is applied to malware classification tasks in [32] and studied its suitability using 11 different CL algorithms on two real-world malware traces. CL for phishing attack detection is explored in [33] on real-world benign and phishing traces collected over three periods. Concerning our work, [11] has shown that the memory-based class of CL approaches is more suitable for NID tasks and suggests operating in domain incremental learning settings (DILS) for better performance. Similarly, we conduct the experiments in the DILS in this work, in which label space across all tasks is fixed to $benign(0)$ or $attack(1)$, while data space may change.

III. Background

This section provides the mathematical foundation necessary to understand the proposed method, SPIDER. It covers the singular value decomposition algorithm, a preliminary method used in the SPIDER, and various notations used in this work.

A. Preliminaries

Singular value decomposition (SVD) is also known as the matrix factorization algorithm. Given a rectangular matrix $A \in \mathbf{R}^{m \times n}$, it is factorized into product of three matrices $U \in \mathbf{R}^{m \times m}$, $V \in \mathbf{R}^{n \times n}$ and $\Sigma \in \mathbf{R}^{m \times n}$ which contain the sorted singular values along the main diagonal. In a nutshell, $A = U\Sigma V^T$[1]. Given the rank (r) of the matrix, a k-rank approximation to the matrix A is given by

[1]for any given matrix A, A^T is the transpose operation on the matrix A.

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T \tag{1}$$

where $k \leq r$, $u_i \in U, v_i \in V$ are the left and right singular vectors and $\sigma_i \in diag(\Sigma)$ are singular values. The value of k can be the smallest value that satisfies $||A_k||_F^2 \geq \delta ||A||_F^2$, where $||.||_F$ is a Frobenius norm of the matrix and δ is threshold hyperparameter ($0 < \delta \leq 1$).

B. Notations

In this section, we present the notations and background required to understand the problem formulation. Specifically, we outline a general training strategy in the continual learning paradigm.

Without loss of generality, we assume that the entire training data can be represented as a series of tasks to mimic human-like learning, a common practice in CL literature [8, 18, 27]. Each task is typically a non-overlapping subset of training samples which assumes *locally task-level iid*. Let $\mathcal{D} = \{\mathcal{D}^1, \mathcal{D}^2, \cdots, \mathcal{D}^t\}$ denote the full training dataset, where $\mathcal{D}^t = \{\mathcal{D}_l^t, \mathcal{D}_u^t\}$ represents the training exemplars of the task 't'. Specifically, $\mathcal{D}_l^t, \mathcal{D}_u^t$ are the respective $labeled$ and $unlabeled$ exemplars of the task 't', where $\mathcal{D}_l^t = \{x_{l_i}^t, y_{l_i}^t\}_{i=1}^{n_l}$ being the feature vector and corresponding label of sample i and $\mathcal{D}_u^t = \{x_{u_j}^t\}_{j=1}^{n_u}$ is a set of unlabeled samples ($n_u \gg n_l$). The label space of $\mathcal{D}_l^t$ consists of two classes (y_0 and y_1), with our focus being on binary classification. Assuming the general continual learning setting, the solver function (predictor) f_θ (parameterized by θ) will observe the training data as an ordered sequence of tasks from $\mathcal{D}$. Each task t (> 1) consists of samples taken from the unlabeled data $\mathcal{D}_u^t$, and the corresponding labels are generated using a pseudo-label generator function. Thus, the learning process for each task 't' consists of two steps: 1) generate pseudo-label ($\hat{y}_u^t$) for the unlabeled data x_u^t taken from $\mathcal{D}_u^t$ and 2) train the predictor f_θ with the data of task 't' ($\hat{\mathcal{D}}_u^t$ and $\mathcal{D}_l^t$) using the loss function $\mathcal{L}(\cdot)$. The objective function over all the tasks can be formulated as follows:

$$\theta^* = \arg\min_\theta \frac{1}{|\mathcal{D}|} \sum_t \mathcal{L}_t(f_\theta^t) \tag{2}$$

where $\mathcal{L}_t(f_\theta^t) = \mathcal{L}_t(f_\theta^t(\mathcal{D}^t))$. The objective function defined in Eqn(2) may not be adequate to handle *catastrophic forgetting* (CF) due to *distribution shifts*. Various families of approaches were proposed to handle CF. In this work, the focus is on memory-based approaches in which a buffer *memory* $\mathcal{M}$ is used to store the samples from all the previous tasks ($\{\mathcal{D}_l^i\}_{i=1}^t$). During training with task 't', $\mathcal{D}_l^t$ will be augmented with data sampled from $\mathcal{M}$ (represented as $\mathcal{D}^m$). As a result, the objective function over all the tasks is reformulated as follows:

$$\begin{aligned} \theta^* &= \arg\min_\theta \frac{1}{|\mathcal{D}_*^t|} \sum_t \mathcal{L}_t(f_\theta^t) + \mathcal{L}_t'(f_\theta^t) \\ \text{where} \quad \mathcal{L}_t(f_\theta^t) &= \mathcal{L}_t(f_\theta^t, \{\mathcal{D}_l^t \cup \hat{\mathcal{D}}_u^t\}), \\ \mathcal{L}_t'(f_\theta^t) &= \mathcal{L}_t(f_\theta^t, \mathcal{D}^m), \quad \text{and} \\ \mathcal{D}_*^t &= \{\mathcal{D}_l^t \cup \hat{\mathcal{D}}_u^t \cup \mathcal{D}^m\} \end{aligned} \tag{3}$$

IV. The Proposed Method: SPIDER

In this section, we describe the problem formulation of the network intrusion detection in the SSCL setting and details of the training process of the proposed SPIDER method for each task.

A. Problem formulation

The objective function in Eqn(3) requires the following: 1) buffer memory ($\mathcal{M}$) to store a subset of all previous tasks training samples to avoid catastrophic forgetting of the learned tasks and 2) label–generating function to generate pseudo labels ($\hat{\mathcal{D}}_u^t$) for the unlabeled data in each task. Whenever training on a longer sequence of tasks, methods that rely on using $\mathcal{M}$ require storing all previous tasks' training samples for longer periods, which may not be feasible when users are concerned about data privacy. In stark contrast, we store only *unlabeled* training samples ($\mathcal{D}_u^m$) from the previous task in $\mathcal{M}$ to ensure privacy in the proposed method. Similar to [31], the model trained on previous task f_θ^{t-1} is used as a pseudo label generating function in this work. So, f_θ^{t-1} will generate pseudo labels for the unlabeled data in memory ($\hat{\mathcal{D}}_u^m$), and in each task ($\hat{\mathcal{D}}_u^t$). However, storing unlabeled samples in $\mathcal{M}$ instead of labeled samples may not effectively handle the detrimental effect of CI on detection accuracy. This is because gradient updates $\nabla_\theta \mathcal{L}_t(\theta)$ will be influenced by the class imbalance ratio [34]. Thus, unlabeled samples in $\mathcal{M}$ bring a newer challenge when privacy is considered alone. This issue can be resolved based on our empirical finding that projecting the current gradient update orthogonal to the directions of the previous tasks' gradients ($\nabla_\theta \mathcal{L}_1(\theta), \nabla_\theta \mathcal{L}_2(\theta), ...$) will reduce the effect of CI. The intuition is that the stochastic gradient descent (SGD) updates lie in the span of the input space, and taking the direction orthogonal to such previous tasks' gradient space will introduce minimum interference from the previously learned tasks and promotes positive backward transfer. Thus, the learning objective becomes the following:

$$\begin{aligned} \theta^* &= \arg\min_\theta \frac{1}{|\mathcal{D}_*^t|} \sum_t \mathcal{L}_t(f_\theta^t, f_\theta^{t-1}) + \mathcal{L}_t'(f_\theta^t, f_\theta^{t-1}) \\ \text{subject to} \quad & \nabla_\theta \mathcal{L}_t(\theta) \perp \text{span}\{\nabla_\theta \mathcal{L}_1(\theta), \nabla_\theta \mathcal{L}_2(\theta), ..., \nabla_\theta \mathcal{L}_{t-1}(\theta)\} \\ \text{where} \quad & \hat{\mathcal{D}}_u^t = f_\theta^{t-1}(\mathcal{D}_u^t), \\ & \hat{\mathcal{D}}_u^m = f_\theta^{t-1}(\mathcal{D}_u^m), \\ & \mathcal{L}_t(f_\theta^t, f_\theta^{t-1}) = \mathcal{L}_t(f_\theta^t, f_\theta^{t-1}, \{\mathcal{D}_l^t \cup \hat{\mathcal{D}}_u^t\}), \\ & \mathcal{L}_t'(f_\theta^t, f_\theta^{t-1}) = \mathcal{L}_t(f_\theta^t, , f_\theta^{t-1}, \hat{\mathcal{D}}_u^m), \\ & \mathcal{D}_*^t = \{\mathcal{D}_l^t \cup \hat{\mathcal{D}}_u^t \cup \hat{\mathcal{D}}_u^m\} \end{aligned} \tag{4}$$

The span $\{\nabla_\theta \mathcal{L}_1(\theta), \nabla_\theta \mathcal{L}_2(\theta), ..., \nabla_\theta \mathcal{L}_{t-1}(\theta)\}$ represents the space of the gradients. We will leverage the concept of gradient projection memory to construct an efficient gradient span. The procedure to construct the gradient space is explained in detail in the following subsection.

B. Constructing the gradient span via GPM

Gradient projection memory (GPM) [21] is built on the intuition that the stochastic gradient descent (SGD) update lies in the input space. When learning a new task, the gradient update step is orthogonal to the gradient subspace of the past tasks to minimize the interference from the past tasks. Based on the prior intuition, to find the gradient subspace of the past tasks, it is sufficient to find the input subspace spanning all the past tasks. The bases of such a subspace are constructed using learned representations of the past task using the singular value decomposition (SVD) algorithm. The union of such bases of all the past tasks stored in memory is known as *gradient projection memory*.

Collating the representations: The representations are the layerwise activation function values corresponding to input, whereas an activation function can be a differentiable non-linear function (*e.g.,* Rectified linear unit-ReLU). For instance, consider the input as x, using P-layered neural network parameterized by θ as f_θ, and the activation function $\sigma(.)$, then the representations at layer p are represented as x^p.

$$x^p = \sigma(f_\theta(x^{p-1})) \tag{5}$$

whereas x^{p-1} is the representation of the input at the layer $p-1$. After learning the first task using $\mathcal{D}^1$, for each layer 'p', a representation matrix $R_1^p = [x_1^p, x_1^p, \cdots, x_{2n_s}^p]$ concatenating '$2n_s$' representations along the column dimension obtained by the forward pass of n_s randomly chosen exemplars per each class (total $2n_s$) from $\mathcal{D}^1$.

Constructing the bases of gradient span: Now, the SVD factorization is applied on $R_1^p = U_1^p \Sigma_1^p (V_1^p)^T$ to obtain its $k-$rank approximation $(R_1^p)_k$ using the following criteria:

$$||(R_1^p)_k||_F^2 \geq \delta^p ||R_1^p||_F^2 \tag{6}$$

Eventually, the first $k-$left singular vectors will become the bases for layer l concerning the first task. Subsequently, bases of the remaining layers are constructed thereafter, all layerwise bases of the first task are stored in gradient projection memory ($\mathcal{M}_{gpm}$). In other words, $\mathcal{M}_{gpm} = \{(\mathcal{M}^p)_{p=1}^P\}$, where $\mathcal{M}^p = [u_{1,1}^p, u_{2,1}^p, \cdots, u_{k,1}^p]$.

For the subsequent tasks (from 2 to t), new gradients (say $\nabla_{\theta^p \mathcal{L}_2}$- read it as the gradient of the loss value of the second task $\mathcal{L}_2$ for the parameters θ at the layer p) are projected onto $\mathcal{M}_{gpm}$ and subtract the projection component. Hence, the residual gradient component lies in the space orthogonal to the core gradient space.

$$\nabla_{\theta^p \mathcal{L}_2} = \nabla_{\theta^p \mathcal{L}_2} - (\nabla_{\theta^p \mathcal{L}_2}) \mathcal{M}^1 (\mathcal{M}^1)^T \tag{7}$$

where $\mathcal{M}^1$ is the gradient projection memory of the first task. Now, before applying SVD to find the bases, we need to eliminate common bases so that newly added bases are unique in GPM [21] using the following equation:

$$\hat{R_2^p} = R_2^p - \mathcal{M}^1 (\mathcal{M}^1)^T R_2^p \tag{8}$$

where, R_2^l is the representations of layer l for second task. SVD is applied to find the $k-$rank approximations bases and eventually added to $\mathcal{M}_{gpm}$. Eventually, to conclude, the span of the gradient space will be the gradient projection memory.

$$\text{span}\{\nabla_\theta \mathcal{L}_1(\theta), \nabla_\theta \mathcal{L}_2(\theta), ..., \nabla_\theta \mathcal{L}_{t-1}(\theta)\} = \mathcal{M}_{gpm} \tag{9}$$

C. Training process

In this section, the training process of the proposed SPIDER method is described. The training process for each task is split into four steps, as outlined in Fig. 1. For task 't', during the first step, the labeled train data x_l^{t-1} of the predecessor task '$t-1$' is forwarded through the current model f_θ^{t-1} results in creating the activation values at each layer. To simplify the notations, we denote all layers-wise activation values $\{R_{t-1}^p\}_{p=1}^P$ as A^{t-1}. After collating A^{t-1}, the corresponding lower (k)-rank representations A_k^{t-1} is computed using the singular value decomposition algorithm (refer to Eqn(6)). Subsequently, redundant bases of A_k^{t-1} concerning the prior tasks bases are removed using the Gram-Schmidt algorithm, and then unique bases will be added to the bases of the past tasks (gradient projection memory $\mathcal{M}_{gpm}$). The remaining step involves training with the exemplars from the task 't'. Specifically, in the second step, training samples for each batch $\mathcal{B}$ are received from three data sources; labeled data (x_l^t, y_l^t), unlabeled (x_u^t) data of the task 't' denoted as $\mathcal{B}_l, \mathcal{B}_u$ and the unlabeled data (x_u^{t-1}) of the predecessor task ('$t-1$') from memory $\mathcal{M}$ denoted as $\mathcal{B}_u^m$. During the third step, pseudo labels are generated using f_θ^{t-1} for $\mathcal{B}_u$ and $\mathcal{B}_u^m$ and denoted as $\hat{\mathcal{B}_u}, \hat{\mathcal{B}_u^m}$. Eventually, during the fourth step, each batch gradient ∇f_B^t is projected orthogonally to $\mathcal{M}_{gpm}$ to compute the newer gradient $\nabla' f_B^t$, which is propagated through all the layers using the standard backpropagation algorithm. The schematic training process of SPIDER is given in Algorithm 1.

V. Preparing Experimental Setup

This section contains the details of the datasets, data preprocessing, task formulation, and all the experiments' facets. These include hyperparameters selection, architecture details, baseline method selection criteria, evaluation metrics, implementation, and hardware details.

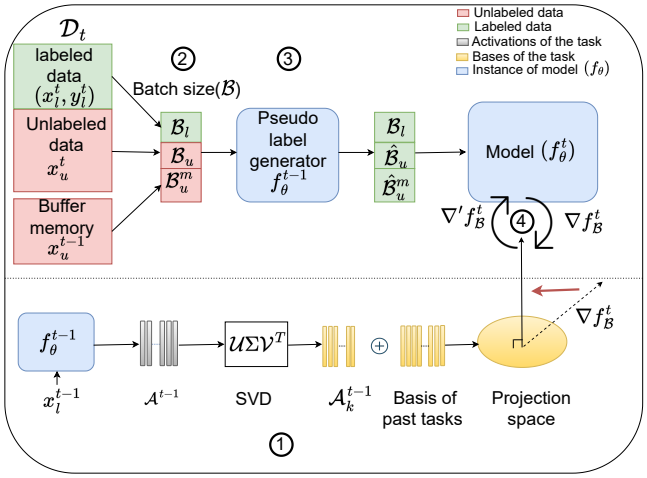


Fig. 1: Graphical illustration of the end-to-end training process of the proposed SPIDER method for each task.

Algorithm 1 Pseudo code of the proposed SPIDER method

Input: sequence of tasks $\{1, 2, \cdots, t-1, t\}$, dataset of task 't' is $\mathcal{D}^t = \{\mathcal{D}_l^t, \mathcal{D}_u^t\}$ (labeled and unlabeled portions), size of $\mathcal{D}^t$ is $|\mathcal{D}^t|$, batch size (b), buffer memory $\mathcal{M}$, no.of samples drawn from $\mathcal{M}$ is b_m ($\leq b$), labeled data ratio r, model f_θ, f_θ trained until task 't' is f_θ^t, gradient projection memory $\mathcal{M}_{gpm}$
for each task 't' **do**
  if 't' is the first task **then**
    while $\mathcal{D}_l^t$ is non-empty **do**
      $\mathcal{B}^t \sim \mathcal{D}_l^t$, where $|\mathcal{B}^t| = b$
      compute gradient $\nabla f_\mathcal{B}^t$ of loss $\mathcal{L}_t$ on $\mathcal{B}$
      update f_θ^t using the $\nabla f_\mathcal{B}^t$
    end while
  else
    sample n_s exemplars per each class from $\mathcal{D}_l^{t-1}$
    compute activations A^{t-1} of $2n_s$ samples using f_θ^{t-1}
    compute A_k^{t-1} using SVD, add it to $\mathcal{M}_{gpm}$
    while $\mathcal{D}^t$ is non-empty **do**
      sample $\mathcal{B}_u^m \sim \mathcal{M}$, where $|\mathcal{B}_u^m| = b_m$
      $b_{rem} = b - b_m$
      sample $\mathcal{B}_l \sim \mathcal{D}_l^t$, where $|\mathcal{B}_l| = b_l = b_{rem} \times r$
      $b_u = b_{rem} - b_l$
      $\mathcal{B}_u \sim \mathcal{D}_u^t$, where $|\mathcal{B}_u| = b_u$
      generate pseudo labels $\hat{\mathcal{B}}_u$ and $\hat{\mathcal{B}}_u^m$ using f_θ^{t-1}
      $\mathcal{B} = \hat{\mathcal{B}}_u^m \cup \mathcal{B}_l \cup \hat{\mathcal{B}}_u$
      compute gradient $\nabla f_\mathcal{B}^t$ of loss $\mathcal{L}_t$ on $\mathcal{B}$
      compute $\nabla' f_\mathcal{B}^t$, orthogonal projection of $\nabla f_\mathcal{B}^t$ on $\mathcal{M}_{gpm}$
      update f_θ^t using $\nabla' f_\mathcal{B}^t$
    end while
  end if
  replace $\mathcal{M}$ with randomly selected samples from $\mathcal{D}_u^t$
end for

A. Datasets

Network intrusion detection datasets: KDDCUP'99 is a widely used dataset that contains 4.9 million samples with 41 features each. Its training set includes 24 types of attacks but has been criticized for its redundancy and lack of representation of real-world network traffic. NSL-KDD is a newer version of KDDCUP'99 that addresses some of these issues but still suffers from the same problems. CICIDS-2017 [35] and CSE-CICIDS-2018 [35] are two multi-class NID datasets initially curated by the Canadian Institute for Cybersecurity [19]. However, the authors of [35] found serious flaws in the original datasets and released the corrected version. It contains 14 attack classes and one benign class and, in total, 2.1 million and

63.2 million samples, respectively. UNSW-NB15 [20] is a multi-class NID dataset with nine attack classes and one benign class. It has 2.5 million samples, and approximately 87% of the samples belong to the benign class. AnoShift [2] is a new unsupervised anomaly detection benchmark that builds upon Kyoto2006+. It spans over ten years with natural temporal variations and almost 90% attack traffic. However, in this work, we used a subset of the AnoShift benchmark.

While the AnoShift benchmark comprehends distribution shifts over a decade of network traffic, other NID benchmarks may lack concept drift as they are artificially generated in a short time frame [2]. Consequently, we validate our proposed approaches using standard continual learning image classification benchmarks. The visual nature of these image classification benchmarks facilitates the observation of distribution shifts more effectively.

Distribution shift in NID datasets: Most of the publicly available NID benchmark datasets are created in a controlled environment by injecting synthetic (artificial) attacks. Furthermore, these datasets are curated over a short period. Specifically, CICIDS2017 and CICIDS2018 datasets contain 5 and 11 days of traffic, which may not be suitable for validating the proposed continual learning method. Our experimentation with NID datasets shows that concept drift (distribution shift) is minimal when datasets are curated over a short time. We hypothesize our claim by measuring the DS between the tasks using optimal transport dataset distance (OTDD). OTDD quantifies the similarity (dissimilarity) between the two tasks using the transportation cost incurred by moving the probability distribution of one task to another task. OTDD computes the transport cost, capturing the underlying data density even when the label set is disjoint between two tasks. The intuition is that the lower the transportation cost, the higher the similarity between the two tasks; thus, minimal DS is present.

TABLE II: Comparing the time taken by dataset curations and the corresponding distribution shift present in the datasets using OTDD values. The Avg. OTDD value is the mean of the OTDD values computed between the two adjacent tasks of a given sequence of tasks for a particular dataset.

Dataset	Dataset curation time	Avg. OTDD value
CICIDS-2017	5 days	0.15
CSE-CICIDS-2018	11 days	0.05
AnoShift	**10 years**	**7189**
CIFAR-10	-	205
CIFAR-100	-	3139

The relationship between the time taken for dataset curation and the corresponding number of DS is outlined in Table II. The DS is measured using the average (Avg.) OTDD values, representing the mean of OTDD values calculated for two successive tasks within the given task sequence. In contrast to curated NID datasets, the AnoShift dataset, spanning a decade, exhibits significantly higher DS than curated NID datasets, making it uniquely suitable for validating the proposed CL-based NID method (SPIDER). As a result, to ensure a thorough evaluation of the proposed approach, we incorporated standard CL benchmark datasets (MNIST, CIFAR-10/100) in our experiments by transforming the image classification problem into an image-based intrusion detection problem. It is worth noting that CIFAR-10 and CIFAR-100 exhibit higher DS compared to existing NID benchmarks, excluding AnoShift.

Image classification datasets: MNIST and CIFAR-10/100 datasets are widely used for image classification tasks. MNIST consists of handwritten digits from 0 to 9, whereas CIFAR-10/100 has 10 and 100 different classes of objects. In our experiments, digit nine of MNIST is considered as the attack class. In CIFAR-10 and CIFAR-100, the attack classes are the truck and the superclass vehicle two, respectively.

B. Data preprocessing

For the KDDCUP'99 and NSL-KDD datasets, three insignificant columns (columns with more zero entries) were removed, resulting in 38 remaining columns. These columns were then normalized using the standard scalar from the $sklearn$ library. Regarding the CICIDS2017 and CICIDS2018 datasets, they were initially spread across multiple CSV files. The data from these files were concatenated into a single CSV file comprising over 90 features. By removing flow-specific identifiers and employing feature engineering based on the Pearson correlation coefficient with a threshold of 90%, the dataset was streamlined, resulting in the retention of approximately 51 features following the min-max normalization. Similarly, for the UNSW-NB15 dataset, four different CSV files were combined, duplicates were removed, followed by the normalization process. The preprocessed AnoShift subset (Kyoto 2006+) is available here [36], and we utilized it for our experiments. MNIST, CIFAR-10, and CIFAR-100 datasets are normalized across the three colored channels using the mean and standard deviation computed over the entire dataset. Each benchmark dataset is divided into three segments: 70% for training, 5% for validation, and 25% for testing.

C. Tasks formulation

Existing CL literature excels at creating tasks from vision datasets. However, NIDS datasets lack such procedures. Based on our experience, we advocate for creating imbalanced tasks with more benign samples (and fewer attack samples) that closely resemble real-world network traffic and incorporating measurable distribution shifts with longer task sequences. For KDDCUP'99 and NSL-KDD, we created five tasks each, and for CICIDS-2017 and CICIDS-2018, we generated ten tasks. Furthermore, nine tasks were created for UNSW-NB and ten for AnoShift. For the computer vision benchmarks MNIST and CIFAR-10, we randomly selected one class as an attack class and divided and distributed the attack class data among the remaining benign classes. This ensured that the experiments formulated using vision benchmarks were similar to the network intrusion detection experiments. Specifically, for MNIST and CIFAR-10, we created nine tasks, each containing one of the nine benign classes along with a chunk of attack class samples randomly chosen as the attack class. Similarly, for the CIFAR-100 benchmark, we created 19 tasks by selecting one randomly chosen superclass label as the attack class.

D. Configuring the experiments

Architecture: For NID and MNIST datasets, we utilize a multi-layer, fully connected (FC) neural network. The AlexNet architecture is a backbone for CIFAR-10 and CIFAR-100, followed by a network of FC layers.

TABLE III: Hyperparameter details of different benchmark datasets

Dataset	#tasks	Batch size	input size	Architecture	l_r	w_d
NSL-KDD	5	500	38	FC:100,500,250,50,1	10^{-5}	10^{-4}
KDDCUP'99	5	1024	38	FC:100,500,250,50,1	10^{-3}	10^{-3}
CICIDS-2017	10	1024	70	FC:100,250,50,1	10^{-3}	10^{-3}
CICIDS-2018	10	1024	70	FC:100,500,250,50,1	10^{-2}	10^{-4}
UNSW-NB15	9	1024	202	FC:100,250,500,150,50,1	10^{-3}	10^{-3}
AnoShift	10	1024	18	FC:100,500,250,50,1	10^{-4}	10^{-5}
MNIST	9	128	$1 \times 32 \times 32$	FC:100,150,50,10,1	10^{-1}	10^{-8}
CIFAR-10	9	128	$3 \times 32 \times 32$	AlexNet,FC:100,50,1	10^{-1}	10^{-5}
CIFAR-100	19	128	$3 \times 32 \times 32$	AlexNet,FC:100,50,1	10^{-1}	10^{-5}

Hyperparamaters: We use stochastic gradient descent (SGD) optimizer in our experiments with a Nesterov momentum value of 0.9. The learning rate decay multistep LR is used at each epoch step with $\gamma = 0.96$ for all the datasets. The learning rate (l_r) and weight

decay (w_d) of SGD are fine tuned using the grid search over the validation set. The early stopping strategy, with a patience value of 3 and a delta error of 0.01, is used on the validation set to improve the generalization performance. The batch size varies across datasets: 128 for CIFAR-10/00 and 1024 for NID datasets, except for NSL-KDD, where the batch size is 500. The number of epochs is set to a maximum value of 100 for all the experiments. More comprehensive hyperparameter information is available in Table III.

Baselines selection: After thoroughly reviewing the CL literature, we have carefully selected baseline methods encompassing a diverse range of CL approaches. In the supervised CL context, we have opted for elastic weight consolidation (EWC [26]), which is a regularization-based approach, average gradient episodic memory (A-GEM [8]), a memory-based gradient projection approach, maximal interfere retrieval (MIR [27]), which belongs to the buffer memory-based CL category, class imbalance family of CL methods (CBRS [18]), and gradient projection memory (GPM [21]), an orthogonal projection based CL approach). In the case of semi-supervised CL, we have included pseudo labeling in class incremental learning (PlCIL [31]), a buffer memory-based method. However, we have decided not to utilize other SSCL methods, such as Distillmatch [30] and ORDisCo [29], as they involve image data augmentation techniques (e.g., flip, crop, blur) that are primarily designed for image classification tasks and are less relevant to non-image datasets like NID. Hence, we have omitted them from our selection.

Evaluation in SSCL setting: Comparing the efficacy of the SSCL setting in NIDS performance involves assessing the utilization of unlabeled data. This assessment is achieved by computing performance bounds for NIDS using the lower and the upper-performance bounds. The lower bound (ℓ) reflects the performance of the NID systems where the ML model is trained with limited annotated data, whereas the upper bound (u) uses fully labeled training data for training. Based on these performance bounds, the evaluation protocol quantifies the performance advantages of employing SSCL in the NID problem. Specifically, Naive methods serve as ℓ, whereas EWC, A-GEM, MIR, and CBRS serve as u.

Evaluation metrics: The metrics selected in this study closely resemble those used in previous works. Specifically, we adopt the receiver operating characteristic area under the curve (ROC-AUC) and the precision-recall area under the curve (PR-AUC) metrics to evaluate both benign and attack classes, which are denoted as PR-AUC (B) and PR-AUC (A), respectively.

Implementation and hardware details: We utilize the open-source continual learning library avalanche (version 0.2.1) for baseline methods such as EWC and A-GEM. However, employing these implementations directly pose certain technical challenges, particularly for NID benchmarks. Therefore, we customize them to suit the requirements of the benchmarks. Additionally, we implement MIR and CBRS using the PyTorch library (version 1.13.0). We conducted our experiments on a system with the following specs: 376 GB of memory, 104 cores (Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz), and 2 Nvidia Quadro RTX 5000 GPUs.

VI. Performance results

In Table IV, we present a quantitative analysis by comparing the proposed SPIDER method with the baseline methods discussed in the previous section. The following key observations are made.

Firstly, the proposed SPIDER method consistently outperforms all baseline methods across all benchmark datasets except for A-GEM, a supervised CL algorithm. However, the performance difference between SPIDER and A-GEM is relatively small. The consistency of A-GEM can be attributed to its retention of all past task-wise representative labeled samples in memory, which helps reduce catastrophic forgetting through gradient projections. To validate the efficacy of SPIDER similarly to A-GEM, we experiment with a memory $\mathcal{M}^*$ containing a subset of labeled samples from all past tasks, denoted as **SPIDER**+$\mathcal{M}^*$. This variation improves the performance of the minority class in large-size datasets exhibiting distribution shift, such as AnoShift (PR-AUC (B) improves to 0.85). Notably, **SPIDER**+$\mathcal{M}^*$ becomes the best method on CIFAR-10 and CIFAR-100 datasets (which has frequent distribution shifts between the tasks). Specifically, the PR-AUC (A) values are increased from 0.44 to 0.59 on CIFAR-10 and 0.18 to 0.29 on CIFAR-100 datasets, respectively. Regarding other supervised baselines, CBRS outperforms MIR and EWC on larger NID datasets concerning PR-AUC on minority classes (particularly CICIDS-2017, CICIDS-2018, and AnoShift). Conversely, MIR outperforms EWC and CBRS on image classification datasets (CIAFR-10 and CIFAR-100). MIR's performance could be because it maintains the most informative samples in the buffer memory and handles higher distribution shifts well.

The second observation is that PLCIL, an SSCL memory-based method containing a subset of labeled samples from all past tasks, performs poorly on all datasets compared to supervised baselines and the proposed SPIDER method. One pragmatic reason for PLCIL's performance besides maintaining the labeled samples in memory could be that its memory organization policy does not consider the class imbalance in the NID datasets. As a result, PLCIL performance is comparable to the naive approach, an SSCL method without memory, acting as a lower bound for any SSCL baseline. This suggests the need for careful attention to class imbalance under the SSCL setting when designing novel methods.

Lastly, using GPM [21] alone to construct representations of past tasks is also affected by class imbalance, leading to a significant performance drop for the minority class. SPIDER addresses this issue, and its performance on the minority class of NID benchmark datasets outperforms GPM [21] utilizing memory that only stores unlabeled samples from the previous task. This phenomenon is empirically validated and found to be consistent on all the NID and image benchmark datasets, as shown in Table V. In conclusion, SPIDER improves the performance of the minority class (typically the attack class) under the SSCL setting in class imbalance compared to baseline methods.

Computational complexity: We assess the computational complexity regarding required training time, measured in wall clock time seconds. Specifically, we consider supervised baselines (EWC, MIR, and A-GEM) while excluding CBRS, as it is solely a reservoir sampling-based memory reorganization policy. The wall clock training times for all methods are presented in Table VI. It can be observed that using SPIDER method reduces the total training time to at least half that of the baseline methods. This shows that SPIDER is more training time-friendly; as a result, it reduces the total training budget.

A. Ablation study

In this section, we evaluate the robustness of SPIDER's performance by investigating its sensitivity to different hyperparameters. Specifically, we conduct experiments to explore the impact of the amount of labeled data used, the effect of memory, and the number of samples utilized to construct the gradient projection memory.

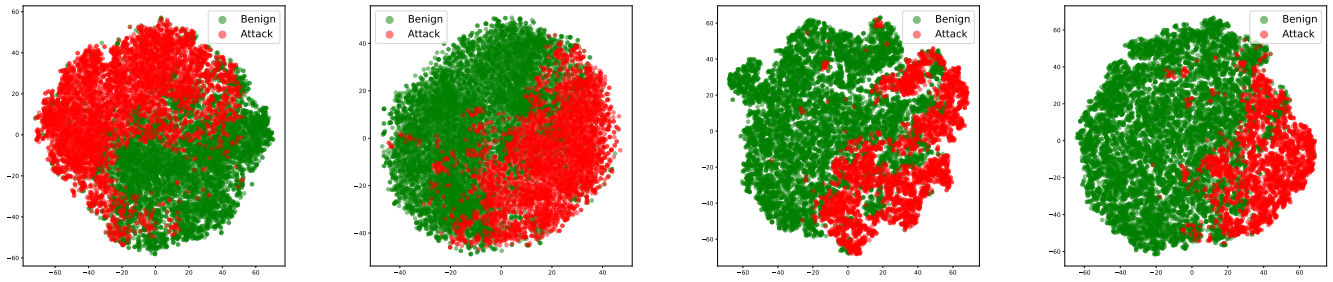
Performance trends with labeled data: In this study, we analyze the influence of varying amounts of labeled data on performance. The corresponding performance trends for NID and image benchmarks are depicted in Fig. 3, and the following observation can be made. We notice a substantial enhancement in the performance of the minority class (measured by the PR-AUC value) during the initial increase in the labeled data used for training, specifically in the range of 2% to 10%. However, after this point, the rate of improvement becomes more gradual. We also explored the performance trend based on the model's learnability. Specifically, we visualized the model's learning capacity by plotting the evolution of the t-SNE visualization of the parameter values of the last layer of the AlexNet network. This was done with varying labeled data on the CIFAR-10 dataset's train set, as shown in Fig. 2. The results reveal that as the amount of labeled

TABLE IV: Performance comparison of the proposed SPIDER method with baseline methods on different NID and image classification datasets. Each experiment is repeated with five different task orders using different seed values, and the arithmetic mean of the ROC-AUC, PR-AUC (B), and PR-AUC (A) values are reported. Red means ranking first, green means ranking second, and blue means ranking third.

	KDDCUP'99			NSL-KDD			CICIDS-2017		
Method	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC
Naive	0.81	0.47	0.51	0.72	0.71	0.70	0.65	0.95	0.87
EWC [26]	**0.99**	**0.96**	**0.99**	**0.95**	**0.93**	**0.94**	0.85	0.98	0.96
A-GEM [8]	**0.99**	**0.98**	**0.99**	**0.95**	**0.93**	**0.94**	**0.99**	**0.99**	**0.99**
MIR [27]	0.90	0.90	0.89	0.88	0.85	0.82	0.90	0.99	0.97
CBRS [18]	0.93	0.72	0.66	0.82	0.70	0.67	**0.91**	**0.99**	**0.97**
PLCIL [31]	0.46	0.59	0.49	0.73	0.75	0.73	0.65	0.95	0.87
SPIDER (ours)	**0.99**	**0.97**	**0.97**	**0.94**	**0.90**	**0.93**	**0.95**	**0.98**	**0.98**
SPIEDER + $\mathcal{M}^*$	**0.99**	**0.97**	**0.98**	**0.93**	**0.90**	**0.93**	0.88	0.99	0.97

	CSE-CICIDS-2018			UNSW-NB15			AnoShift		
Method	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC
Naive	0.42	0.97	0.73	0.71	0.98	0.93	0.78	0.57	0.65
EWC	0.56	0.95	0.75	**0.99**	**1.00**	**0.99**	0.90	0.55	0.76
A-GEM	0.53	0.97	0.50	**0.99**	**1.00**	**0.99**	**0.96**	**0.85**	**0.92**
MIR	0.86	0.99	0.98	0.92	0.99	0.99	0.82	0.43	0.67
CBRS	**0.95**	**0.99**	**0.99**	0.93	0.99	0.99	0.90	0.76	0.83
PLCIL	0.42	0.97	0.73	0.71	0.98	0.91	0.78	0.57	0.65
SPIDER (ours)	**0.98**	**0.99**	**0.99**	**0.97**	**0.99**	**0.99**	**0.91**	**0.83**	**0.88**
SPIEDER + $\mathcal{M}^*$	**0.99**	**0.99**	**0.99**	0.93	0.99	0.99	**0.92**	**0.85**	**0.90**

	MNIST			CIFAR-10			CIFAR-100		
Method	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC	PR-AUC (A)	PR-AUC (B)	ROC-AUC
Naive	0.40	0.98	0.86	0.20	0.95	0.71	0.04	0.97	0.42
EWC	**0.86**	**0.99**	**0.97**	0.31	0.97	0.84	0.13	0.98	0.75
A-GEM	**0.93**	**0.99**	**0.98**	**0.43**	**0.98**	**0.88**	**0.15**	**0.98**	**0.80**
MIR	0.78	0.99	0.96	0.42	0.98	0.87	0.14	0.97	0.73
CBRS	0.77	0.99	0.96	0.27	0.96	0.78	0.08	0.97	0.66
PLCIL	0.40	0.98	0.86	0.21	0.95	0.71	0.08	0.97	0.64
SPIDER (ours)	0.81	0.99	0.95	**0.44**	**0.98**	**0.87**	**0.18**	**0.98**	**0.75**
SPIEDER + $\mathcal{M}^*$	**0.82**	**0.99**	**0.96**	**0.59**	**0.98**	**0.91**	**0.29**	**0.98**	**0.83**



(a) r = 2%, PR-AUC(A) = 0.17 (b) r = 10%, PR-AUC(A) = 0.35 (c) r = 20%, PR-AUC(A) = 0.50 (d) r = 40%, PR-AUC(A) = 0.54

Fig. 2: The t-SNE visualization shows model learnability on CIFAR-10 using AlexNet's last layer activations. The PR-AUC value of the attack class improves with more labeled data (r).

data increases, the PR-AUC value of the minority class improves, and the corresponding t-SNE visualizations show a convergence towards creating a distinct class separation boundary.

Effect of buffer memory ($\mathcal{M}$): This study focuses on the impact of memory on the SPIDER performance using NID datasets, and the results are presented in Table VII. The following observations can be made: The presence of $\mathcal{M}$ does not affect the majority class since, during backpropagation, most of the gradients are dominated by the majority class. This resulted in the suppression of minority class gradients, leading to the problem of minority performance drop. This is a direct consequence of the class imbalance ratio (CIR) affecting gradient updates, supported by prior work [34]. However, this effect can be mitigated by replaying minority class samples stored in $\mathcal{M}$. As a result, the batch-level CIR is reduced, which reduces the gradient dominance of the majority class and leads to an improvement in the performance of the minority class.

Effect of varying the projection memory samples (n_s): In this study, we investigate the impact of the number of samples used per

TABLE V: Performance comparison between SPIDER and GPM using PR-AUC values computed on minority class of different NID datasets. Each experiment is repeated with five task orders, and the arithmetic mean values are reported. The best values are marked in bold.

Method	KDDCUP'99	NSL-KDD	CICIDS-2017	CICIDS-2018
GPM [21]	0.59	0.75	0.95	0.89
SPIDER	**0.97**	**0.90**	0.95	**0.98**

Method	AnoShift	MNIST	CIFAR-10	CIFAR-100
GPM [21]	0.76	0.60	0.41	0.16
SPIDER	**0.83**	**0.81**	**0.44**	**0.18**

TABLE VI: The proposed SPIDER method's training time is compared with the baseline methods in seconds, measured using wall clock time. Each experiment is repeated with five task orders, and mean values are reported. The best values are marked in bold.

Methods	KDDCUP'99	UNSW-NB15	CICIDS-2017	CSE-CICIDS-2018	AnoShift
EWC	823	630	370	11914	2144
A-GEM	1397	848	318	9448	3087
MIR	1427	948	804	11752	3086
SPIDER	**268**	**419**	**231**	**4420**	**1560**

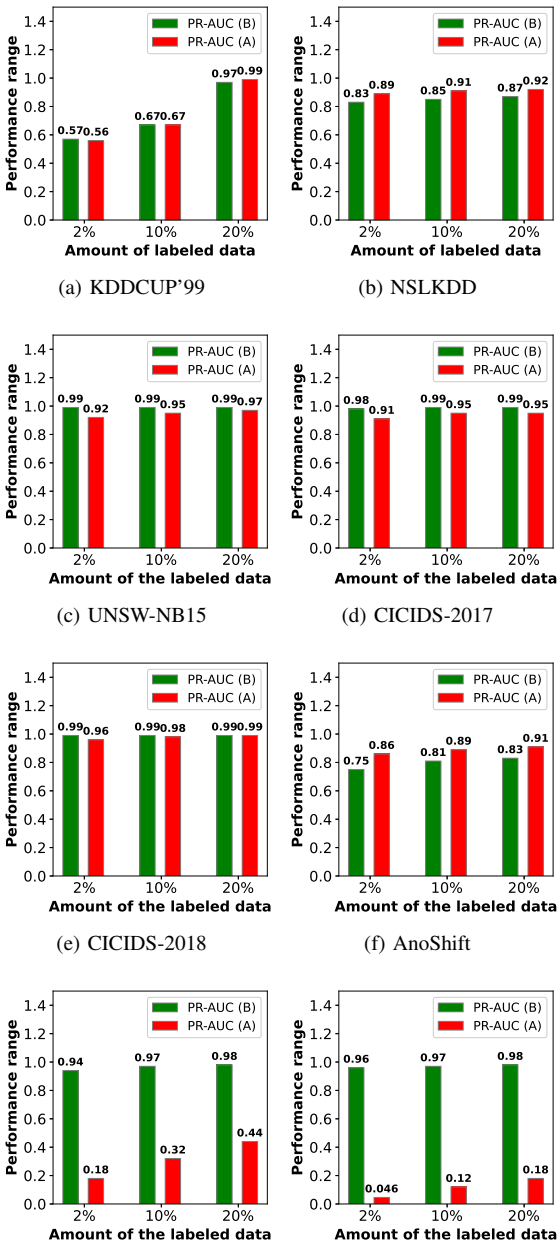
TABLE VII: The effect of buffer memory on SPIDER's performance is assessed using PR-AUC (A) and PR-AUC (B) values on different NID benchmark datasets. Performance improvements are highlighted with a red arrow (↑).

	PR-AUC (A)			
Method	KDDCUP'99	CICIDS-2017	CICIDS-2018	AnoShift
SPIDER - $\mathcal{M}$	0.99	0.95	0.91	0.90
SPIDER	0.99	0.91 (↓)	0.98(↑)	0.91
	PR-AUC (B)			
Method	KDDCUP'99	CICIDS-2017	CICIDS-2018	AnoShift
SPIDER - $\mathcal{M}$	0.97	0.99	0.99	0.76
SPIDER	0.97	0.99	0.99	0.83 (↑)

TABLE VIII: The influence of the number of gradient projection memory samples on SPIDER's performance is assessed using the PR-AUC value of the minority class in each NID benchmark dataset. Performance improvements are denoted by red arrows (↑). Each experiment is repeated five times with different task orders, and mean values are reported.

	PR-AUC			
n_s	KDDCUP'99	CICIDS-2017	CICIDS-2018	AnoShift
10^2	0.97	0.95	0.99	0.82
10^3	0.97	0.95	0.98 (↓)	0.82
10^4	0.97	0.95	0.98	0.82
10^5	0.97	0.95	0.98	0.83 (↑)

each task class (refer to line 12 of the Algorithm 1) to construct the corresponding task bases before adding them to the gradient projection memory. It is important to note that when the number of class samples in each task (n_c) is lower than n_s ($n_c \leq n_s$), we always pick the minimum of n_c and n_s. The results of this study on the performance of the minority class in NID benchmark datasets



(a) KDDCUP'99 (b) NSLKDD

(c) UNSW-NB15 (d) CICIDS-2017

(e) CICIDS-2018 (f) AnoShift

(g) CIFAR-10 (h) CIFAR-100

Fig. 3: Variation in the performance with varying amounts of labeled data on different NID datasets. Each experiment is conducted with different task orders five times, and the mean values are plotted.

are presented in Table VIII. The following observations can be made: First, there is a slight or no improvement in the performance of the minority class, which we attribute to the greater challenge in enhancing the performance of the minority class compared to the majority class, as explained by the gradient dominance phenomenon in the previous study (effect of the buffer memory). Second, we observe that n_s and the size of the datasets are empirically independent on NID benchmarks. In other words, the minority class detection performance is not improving by increasing n_s on NID datasets. Specifically, the PR-AUC metric value of the minority class of the KDDCUP'99, CICIDS-2017, CICIDS-2018, and AnoShift is stable at n_s=10^2, does

not improve with increasing n_s.

B. Limitations of the SPIDER and Scope of the Future Work

Our work is motivated to reduce the requirement of annotated data and the need to eliminate the storage of labeled data in buffer memory aid to handle the CI of the network data in the continual learning setting. In light of extensive experiments conducted on NID and image benchmark datasets, we observe the following limitations of the proposed SPIDER method.

Closed-world assumption: The goal of the CL framework is to sequentially learn new tasks without forgetting (catastrophic forgetting) the knowledge of the past tasks. Thus, the CL framework naturally assumes the closed world assumption. In contrast, cybersecurity defense systems (like NIDS) are meant to be deployed to identify unseen attacks (zero-day attacks). Thus, an open-world setting is a natural choice for security applications. This is our initial attempt to understand the application of SSCL to NID tasks, so in this work, we consider a closed-world learning setting. As a part of future work, we will include an open-world learning setting as a primary objective.

Amount of labeled data: In our work, we conducted experiments varying the amount of labeled data to examine the efficacy of the proposed SPIDER method. However, for real-world ML-based NIDS developers, it is not easy to define the amount of labeled data to consider given terabytes of network traffic generated in an enterprise-scale environment. Further, annotating a small fraction of network traffic requires a lot of domain expertise and manual effort (time and cost). In other words, assuming a labeling oracle is impractical in a real-world setting.

Evaluation datasets: Evaluating the SPIDER on the public datasets remains a common constraint in NIDS research. Specifically, we know that KDDCUP'99 is nearly 2.5 decades old, but we intend to use it to validate how our proposed approach works in the CL setting without requiring access to past task data while training on current task.

Potential extensions to this work involves exploring self-supervised techniques with continual learning to enhance detection accuracy. Further, one could develop an unsupervised anomaly detection method that will continually learn without the need for labels and identify unseen attacks near real-time at an enterprise-level scale.

C. Concluding remarks

This work was motivated by the limitations of existing approaches that do not adequately address the issue of adapting to distribution shifts in the network traffic of benign and attack data in network intrusion detection. While unsupervised learning methods do not require capturing the concept drift of attack classes, their effectiveness can deteriorate when normality shifts occur. To overcome this problem, we formulated the NID as a binary classification problem to adapt to both distribution shifts. However, such a formulation demands access to large amounts of annotated data and the ability to handle class imbalance. To solve these challenges, we introduced a novel semisupervised, continual learning-based method called SPIDER for NID tasks. The key novelty lies in leveraging gradient projection memory (GPM) [21] combined with semisupervised continual learning (SSCL) techniques to handle class imbalance and data privacy concerns effectively. Unlike existing methods that store all of the past tasks samples in memory for an extended duration, potentially compromising data privacy, the proposed SPIDER method requires storing only unlabeled data in the memory from the previous task.

We conducted an extensive empirical evaluation of SPIDER on six standard network intrusion detection datasets, including the recent AnoShift dataset, which exhibits natural distribution shifts over a decade of network traffic. We also evaluated SPIDER on three image classification datasets. Our findings demonstrate that SPIDER achieves a comparable level of performance with baselines (four supervised CL and one SSCL method), requiring only a maximum of 20% labeled data, thus ensuring a reasonable level of data privacy. Furthermore, SPIDER significantly reduced (by 2x) the training time on NID benchmarks, showcasing its scalability and effectiveness in handling class imbalance and data privacy in network intrusion detection datasets. The authors have provided public access to their code and/or data at [37].

ACKNOWLEDGEMENTS

We sincerely thank all the anonymous reviewers for their constructive feedback and invaluable suggestions, which helped greatly improve this work. SKA thanks the LRN Foundation and COMSNETS association, and the Ministry of Education, Government of India for the financial support.

REFERENCES

[1] A. Tsymbal, "The problem of concept drift: Definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, p. 58, 2004.

[2] M. Dragoi, E. Burceanu, E. Haller, A. Manolache, and F. Brad, "Anoshift: A distribution shift benchmark for unsupervised anomaly detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 854–32 867, 2022.

[3] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, pp. 69–101, 1996.

[4] W. Ren, P. Wang, X. Li, C. E. Hughes, and Y. Fu, "Semi-supervised drifted stream learning with short lookback," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1504–1513.

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

[6] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[7] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[8] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-GEM," in *International Conference on Learning Representations*, 2019.

[9] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proceedings of the 2019 ACM SIGSAC Conference on CCS*, 2019, pp. 1283–1297.

[10] D. Han *et al.*, "Anomaly detection in the open world: Normality shift detection, explanation, and adaptation," in *30th Annual Network and Distributed System Security Symposium (NDSS)*, 2023.

[11] S. K. Amalapuram, A. Tadwai, R. Vinta, S. S. Channappayya, and B. R. Tamma, "Continual learning for anomaly based network intrusion detection," in *14th International Conference on COMSNETS*, 2022, pp. 497–505.

[12] S. kumar Amalapuram, S. S. Channappayya, and B. Tamma, "Augmented memory replay-based continual learning approaches for network intrusion detection," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[13] A. Koksal, K. G. Ince, and A. Alatan, "Effect of annotation errors on drone detection with yolov3," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 1030–1031.

[14] S.-A. Rebuffi, S. Ehrhardt, K. Han, A. Vedaldi, and A. Zisserman, "Semi-supervised learning with scarce annotations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition workshops*, 2020, pp. 762–763.

[15] Y. Guo, M. Liu, Y. Li, L. Wang, T. Yang, and T. Rosing, "Attacking lifelong learning models with gradient reversion," 2019.

[16] X. Mi *et al.*, *Adversarial robust memory-based continual learner*, 2023. arXiv: 2311.17608 [cs.CV].

[17] S. K. Amalapuram, T. T. Reddy, S. S. Channappayya, and B. R. Tamma, "On handling class imbalance in continual learning based network intrusion detection systems," in *The First International Conference on AI-ML-Systems*, 2021, pp. 1–7.

[18] A. Chrysakis and M.-F. Moens, "Online continual learning from imbalanced data," in *International Conference on Machine Learning*, PMLR, 2020, pp. 1952–1961.

[19] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSP*, vol. 1, pp. 108–116, 2018.

[20] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, IEEE, 2015, pp. 1–6.

[21] G. Saha, I. Garg, and K. Roy, "Gradient projection memory for continual learning," in *International Conference on Learning Representations*, 2021.

[22] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[23] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *Proceedings of the 14th ACM workshop on AI and security*, 2021, pp. 111–122.

[24] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee, "Polymorphic blending attacks.," in *USENIX security symposium*, 2006, pp. 241–256.

[25] D. Cieslak, N. Chawla, and A. Striegel, "Combating imbalance in network intrusion datasets," in *2006 IEEE International Conference on Granular Computing*, 2006, pp. 732–737.

[26] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the NAS*, vol. 114, no. 13, pp. 3521–3526, 2017.

[27] R. Aljundi *et al.*, "Online continual learning with maximal interfered retrieval," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[28] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on challenges in representation learning, ICML*, vol. 3, 2013, p. 896.

[29] L. Wang, K. Yang, C. Li, L. Hong, Z. Li, and J. Zhu, "Ordisco: Effective and efficient usage of incremental unlabeled data for semi-supervised continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5383–5392.

[30] J. Smith, J. Balloch, Y.-C. Hsu, and Z. Kira, "Memory-efficient semi-supervised continual learning: The world is its own replay buffer," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[31] A. Lechat, S. Herbin, and F. Jurie, "Pseudo-labeling for class incremental learning," in *BMVC 2021: The British Machine Vision Conference*, 2021.

[32] M. S. Rahman, S. Coull, and M. Wright, "On the limitations of continual learning for malware classification," in *Conference on Lifelong Learning Agents*, PMLR, 2022, pp. 564–582.

[33] A. Ejaz, A. N. Mian, and S. Manzoor, "Life-long phishing attack detection using continual learning," *Scientific Reports*, vol. 13, no. 1, p. 11 488, 2023.

[34] E. Francazi, M. Baity-Jesi, and A. Lucchi, "A theoretical analysis of the learning dynamics under class imbalance," in *International Conference on Machine Learning*, PMLR, 2023, pp. 10 285–10 322.

[35] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018," in *2022 IEEE Conference on Communications and Network Security (CNS)*, 2022, pp. 254–262.

[36] *Anoshift subset*, https://github.com/bit-ml/AnoShift, Accessed: June 2023.

[37] *Spider*, https://github.com/amalapuram/spider, Accessed: Jan 2024.