COMPUTER ENGINEERING

DCAD: A Dynamic Continual Anomaly Detection Framework for Adaptive Network Traffic Analysis under Concept Drift

Muhammad Mustapha Maihaja, Nyangwarimam Obadiah Ali

Nile University of Nigeria, Abuja, Nigeria

Published November 2026

Abstract

Machine-learning-based network anomaly detectors are usually trained once on historical traffic, yet real traffic keeps changing as applications, software, and attack techniques evolve, a problem known as concept drift. Detectors that ignore drift raise false alarms and miss novel attacks, and periodic manual retraining is costly. This paper proposes the Dynamic Continual Anomaly Detection (DCAD) framework, which couples a preprocessing layer, an ADWIN-based concept-drift monitor, an incremental learning engine, a replay-based knowledge-retention layer, and a threshold-based decision layer. An Autoencoder trained only on normal traffic serves as the detection engine and flags flows whose reconstruction error exceeds a validation-selected threshold. On the NSL-KDD benchmark, the Autoencoder detector is compared with SVM, Random Forest, and Isolation Forest. It achieved the highest accuracy (83.4% versus 78.1%, 77.1%, and 70.6%), a false positive rate of 3.7%, an attack recall of 73.7%, and the lowest per-sample inference time. Over ten consecutive evaluation segments its accuracy fell by 1.8 percentage points, against 3.2 and 3.1 for SVM and Random Forest. A streaming experiment with ADWIN-triggered incremental updates illustrates the adaptation mechanism. The limitations of the evaluation, including a single dataset and the absence of an end-to-end drift-adaptation test of the Autoencoder, are discussed.

Keywords: Anomaly detection; autoencoder; concept drift; continual learning; network intrusion detection; NSL-KDD

1 Introduction

The rapid growth of connected devices, cloud services, and internet-based applications has increased both the volume of network traffic and the exposure of networks to cyber threats such as denial-of-service attacks, malware, botnets, and ransomware [1]. Signature-based defenses detect only known attacks, so anomaly detection, which models normal behavior and flags deviations, has become a cornerstone of modern intrusion detection [2]. Machine learning (ML) is the main tool for this task because it can learn complex patterns from large volumes of traffic [3].

A fundamental difficulty is that network environments are not stationary. Network expansion, software updates, new applications, and evolving attack techniques change the statistical properties of traffic over time, so that what is “normal” today may be anomalous tomorrow and vice versa. This phenomenon is known as concept drift [4, 5]. Models trained in a static, batch setting degrade under drift: they misclassify legitimate behavior as malicious, which produces high false positive rates and analyst fatigue, and they fail to detect novel attacks. The usual remedy, periodic manual retraining, is resource-intensive and slow, and it leaves a window of vulnerability between the moment drift occurs and the next retraining.

This paper proposes the Dynamic Continual Anomaly Detection (DCAD) framework, an adaptive architecture that integrates anomaly detection, concept-drift detection, incremental learning, and knowledge retention. An Autoencoder trained on normal traffic is used as the detection engine. The contributions are as follows. First, we characterize the limitations of three widely used static detectors (SVM, Random Forest, and Isolation Forest) on the NSL-KDD benchmark. Second, we present the modular DCAD architecture, in which an ADWIN drift monitor triggers incremental model updates while a replay buffer preserves earlier knowledge. Third, we implement an Autoencoder detector with a validation-selected decision threshold and compare it with the static baselines in terms of accuracy, false positive rate, response time, and stability over successive evaluation segments. Fourth, we demonstrate ADWIN-triggered incremental updating on a data stream and state clearly which parts of the framework have and have not been evaluated end to end.

The remainder of the paper is organized as follows. Related work is reviewed next, followed by a description of the DCAD framework and the experimental methodology. The results are then reported and discussed, and the final section concludes the paper.

2 Related work

2.1 Machine learning for network anomaly detection

Supervised classifiers such as Support Vector Machines (SVM) [6] and Random Forests (RF) [7] are common in intrusion detection and perform well when labeled attack data are available, but they depend on labels and tend to struggle with attack types absent from training. Unsupervised methods such as Isolation Forest [8] instead model the normal class and require no attack labels. Autoencoders, which learn a compact representation of the input and reconstruct it [9], are attractive for anomaly detection because inputs that differ from the training distribution reconstruct poorly [10]. Deep sequence models such as recurrent networks have also been applied to intrusion detection [11], and ensembles of autoencoders have been used for online network intrusion detection [12].

Evaluation practice has been criticized in this field. Axelsson showed that, because attacks are rare relative to normal traffic, even a low false-alarm rate can make a detector impractical, and that accuracy alone is a misleading metric [13]. Sommer and Paxson argued that anomaly detection in networks differs from other ML applications and that studies must consider operational realities [3]. Benchmark datasets are also a concern: NSL-KDD was derived from KDD'99 to remove redundant records [14], but it remains an older, simulated data set, and the literature has surveyed more recent alternatives [15].

2.2 Concept drift and continual learning

Surveys of concept drift describe how changes in the data distribution degrade predictive models and how drift can be detected and handled [4, 5]. ADWIN (adaptive windowing) is a widely used detector that maintains a variable-length window over a data stream and signals drift when two sub-windows have significantly different means [16]. When a neural model is updated on new data, it may forget earlier knowledge, a failure known as catastrophic forgetting [17, 18]; experience replay, which mixes stored earlier samples into updates, is a standard mitigation [19].

Recent work combines these ideas for anomaly detection. Drift detectors built on autoencoder reconstruction error have been proposed [20], continual-learning processes have been designed to detect both previously learned and newly emerging attacks [21], and anomaly detectors that learn

continuously from in-field data have been explored in other safety-critical settings [22]. State-transition-aware detectors reuse earlier models when a previously seen concept recurs [23].

2.3 Research gap

Most network anomaly detectors are still evaluated as static models, and the cost of keeping them current is rarely measured. There is a need for a lightweight, drift-aware architecture that detects drift automatically, updates incrementally instead of retraining from scratch, retains useful earlier knowledge, and keeps the per-flow response time low. The DCAD framework addresses this need.

3 The DCAD framework

3.1 Architecture overview

The DCAD framework (Figure 1) processes network traffic through five connected modules, with a feedback loop that supplies updated knowledge to the drift-detection and learning modules. The design objectives are sustained detection accuracy, a low false positive rate (FPR), fast incremental updates compared with full retraining, a low response time per flow, adaptability to drift, retention of earlier knowledge, and scalability to large traffic volumes.

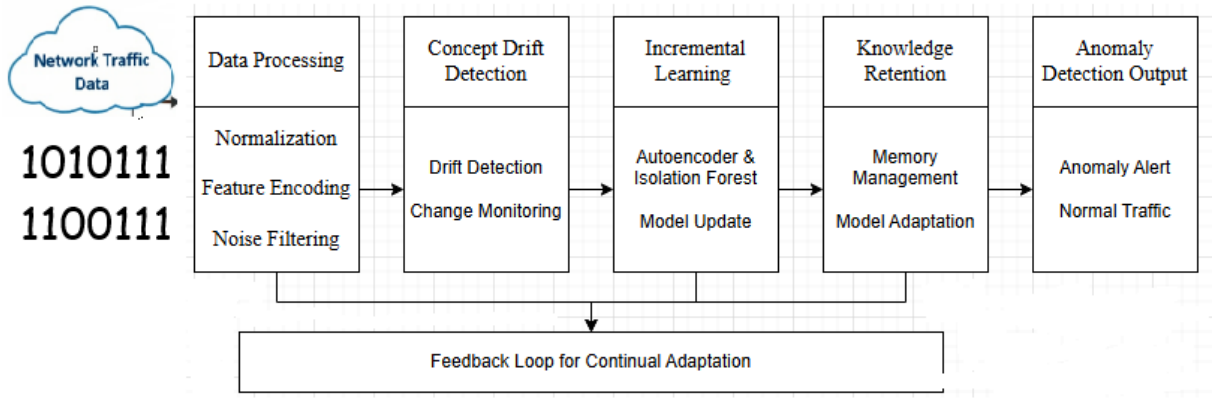


Figure 1. Architecture of the proposed DCAD framework.

3.2 Modules

Data preprocessing layer. Raw traffic records are cleaned, categorical attributes are encoded numerically, and features are normalized so that they can be consumed by the learning models.

Concept-drift detection module. The module monitors the incoming stream with sliding windows. In our implementation, the binary prediction error of the current model is fed to ADWIN [16], using the River library [24]; a detected change in the error rate is interpreted as drift. When the detector is applied to the Autoencoder, the reconstruction-error trend can be monitored in the same way.

Incremental learning engine. When drift is signaled, the engine updates the model with recently observed traffic instead of retraining from scratch. In the streaming experiment reported below, the update used the most recent 1,000 samples.

Knowledge retention and update layer. To avoid catastrophic forgetting [17], the design keeps a replay buffer of previously learned normal patterns that is mixed into updates [19], so that the model integrates new behavior without discarding the old.

Decision layer. The detector outputs an alert or a normal label by comparing the anomaly score with a threshold chosen to balance detection accuracy and FPR. For the Autoencoder, the score is the reconstruction error defined in Section IV-C.

4 Experimental methodology

4.1 Dataset

Experiments used the NSL-KDD data set [14], a refined version of KDD'99 in which duplicate records were removed. Each record describes a network connection with 41 features (for example, protocol type, service, duration, and byte counts) and a label that is either normal or one of four attack categories: denial of service, probe, remote-to-local, and user-to-root. All attack categories were merged into a single attack class for binary detection. The training file (KDDTrain+) holds 125,973 records. The test file (KDDTest+) holds 22,544 records (9,711 normal and 12,833 attack) and was processed as a stream to emulate continuous monitoring. KDDTest+ contains attack types that do not occur in the training file, so the train–test split embeds a distribution shift.

4.2 Preprocessing

The three categorical attributes (protocol type, service, and flag) were encoded as integers, the class label was converted to a binary normal/attack indicator, and all features were standardized to zero mean and unit variance using statistics computed on the training data only:

$$x'_j = \frac{x_j - \mu_j}{\sigma_j} \tag{1}$$

where μ_j and σ_j are the mean and standard deviation of feature j, computed on the training data.

4.3 Autoencoder detection engine

The detector is a fully connected Autoencoder (Figure 2) with an encoder that compresses the input to a low-dimensional latent representation and a symmetric decoder that reconstructs it. Its configuration is given in Table 1; it is referred to as DCAD-AE in the results. The model was implemented in PyTorch [25] and trained with the Adam optimizer [26] on the normal records of the training file only, so it learns the structure of legitimate traffic. For an input vector x with d features and reconstruction $\hat{x}$, the anomaly score is the mean squared reconstruction error, i.e., the squared Euclidean distance between x and $\hat{x}$ divided by d:

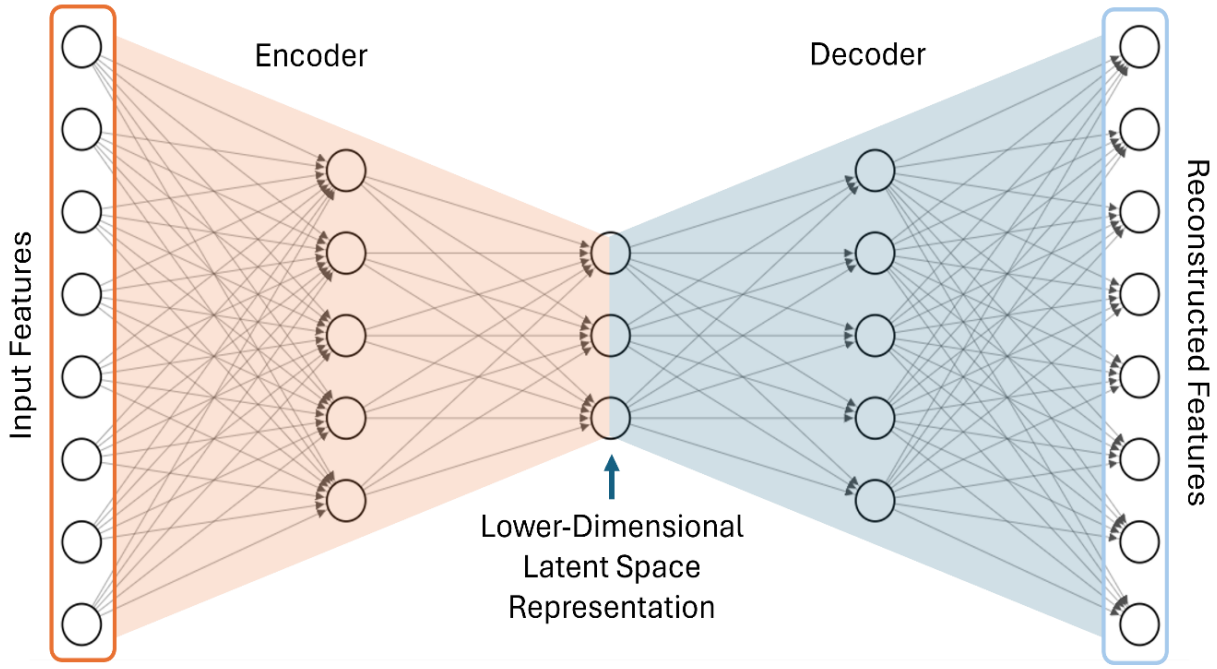


Figure 2. Structure of the Autoencoder used as the detection engine.

$$e(x) = \frac{\| x - \hat{x} \|^2}{d} \tag{2}$$

A flow is labeled anomalous when e(x) exceeds a threshold τ:

$$\hat{y}(x) = \begin{cases} 1 \text{ if } e(x) > \tau \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

The threshold τ was selected by sweeping candidate values on a held-out validation split (the last 20% of the training file) and keeping the value that maximizes the F1-score. The validation split was never used to train the Autoencoder. This choice replaced an earlier configuration with a fixed threshold (the mean plus three standard deviations of the training reconstruction error) and only 50 full-batch training steps. That configuration underfit the model and classified almost all traffic, including most attacks, as normal, which shows how sensitive the detector is to threshold selection.

Table 1. Autoencoder configuration.

Parameter	Value
Architecture	41–32–16–8–16–32–41, fully connected
Activation	ReLU (hidden layers), linear output
Loss function	Mean squared error (MSE)
Optimizer	Adam, learning rate 10^{-3}, weight decay 10^{-5}
Batch size / epochs	256 / 60
Training data	Normal records of KDDTrain+ only
Threshold τ	F1-maximizing value on validation split
Random seed	42

4.4 Baselines and implementation

Three static baselines were trained once and never updated: an SVM, a Random Forest with 100 trees, and an Isolation Forest, all implemented with scikit-learn [27]. SVM and Random Forest were trained on the labeled training file; Isolation Forest is unsupervised. The experiments ran in Python on a personal computer with an Intel Core i7 processor and 8 GB of RAM, using Google Colab.

4.5 Evaluation metrics

Let TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, with the attack class as positive. Because intrusion data are imbalanced, several metrics are reported [13]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

$$\text{FPR} = \frac{FP}{FP + TN} \tag{8}$$

The response time is the inference time per sample, obtained by dividing the total inference time on the test set by the number of records. Stability was assessed by splitting the test stream into ten consecutive segments of 2,254 records each and measuring accuracy on every segment with the models fixed.

5 Results and discussion

5.1 Performance of the static baselines

Table 2 summarizes performance on KDDTest+. SVM achieved the highest accuracy among the baselines (78.14%) and a precision of 97.74%, but its recall was only 63.05% and it was the slowest to train (93.94 s) and to run (9.61 s on the full test set). Random Forest was nearly as accurate (77.08%) and much faster (11.45 s training, 0.19 s inference). Isolation Forest was the fastest (1.06 s training, 0.15 s inference) but the least accurate (70.64%) and had the highest false positive rate (0.083). Precision, recall, and F1 were not computed for Isolation Forest.

The gap between in-distribution and test performance is informative. In a check on a held-out portion of the training data, Random Forest reached about 99.9% accuracy, whereas on KDDTest+ it reached 77.1%. This large drop is consistent with the distribution shift between the two files and illustrates why static models are fragile when traffic changes.

Table 2. Detection performance on KDDTest+.

Model	Acc.	Prec.	Rec.	F1	FPR	Time
SVM	78.14	**97.74**	63.05	76.65	**0.019**	2.1
Random Forest	77.08	96.66	61.86	75.44	0.028	0.43
Isolation Forest	70.64	—	—	—	0.083	0.05

Model	Acc.	Prec.	Rec.	F1	FPR	Time
DCAD-AE	**83.40**	96.32†	**73.65**†	**83.48**†	0.037	**0.0017**

Note: Acc., Prec., Rec., and F1 in %; Time is the inference time in ms per sample. Best value per column in bold. DCAD-AE is the Autoencoder detection engine of the framework. †Computed from the confusion matrix in Figure 4. "—": not computed.

5.2 Performance of the Autoencoder detection engine

The training loss of the Autoencoder (Figure 3) decreased from about 0.65 in the first epoch to about 0.06 after 60 epochs, indicating that the model learned to reconstruct normal traffic. On KDDTest+, the Autoencoder correctly identified 9,350 normal and 9,452 attack records, with 361 false alarms and 3,381 missed attacks (Figure 4).

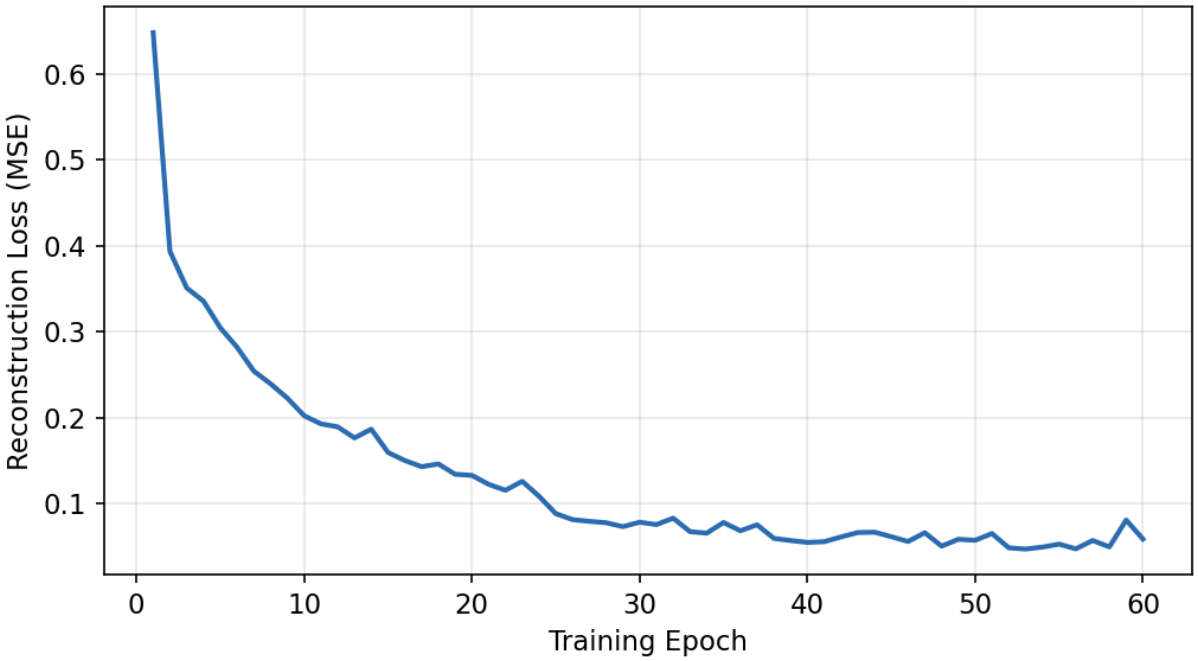


Figure 3. Training loss (MSE) of the Autoencoder over 60 epochs.

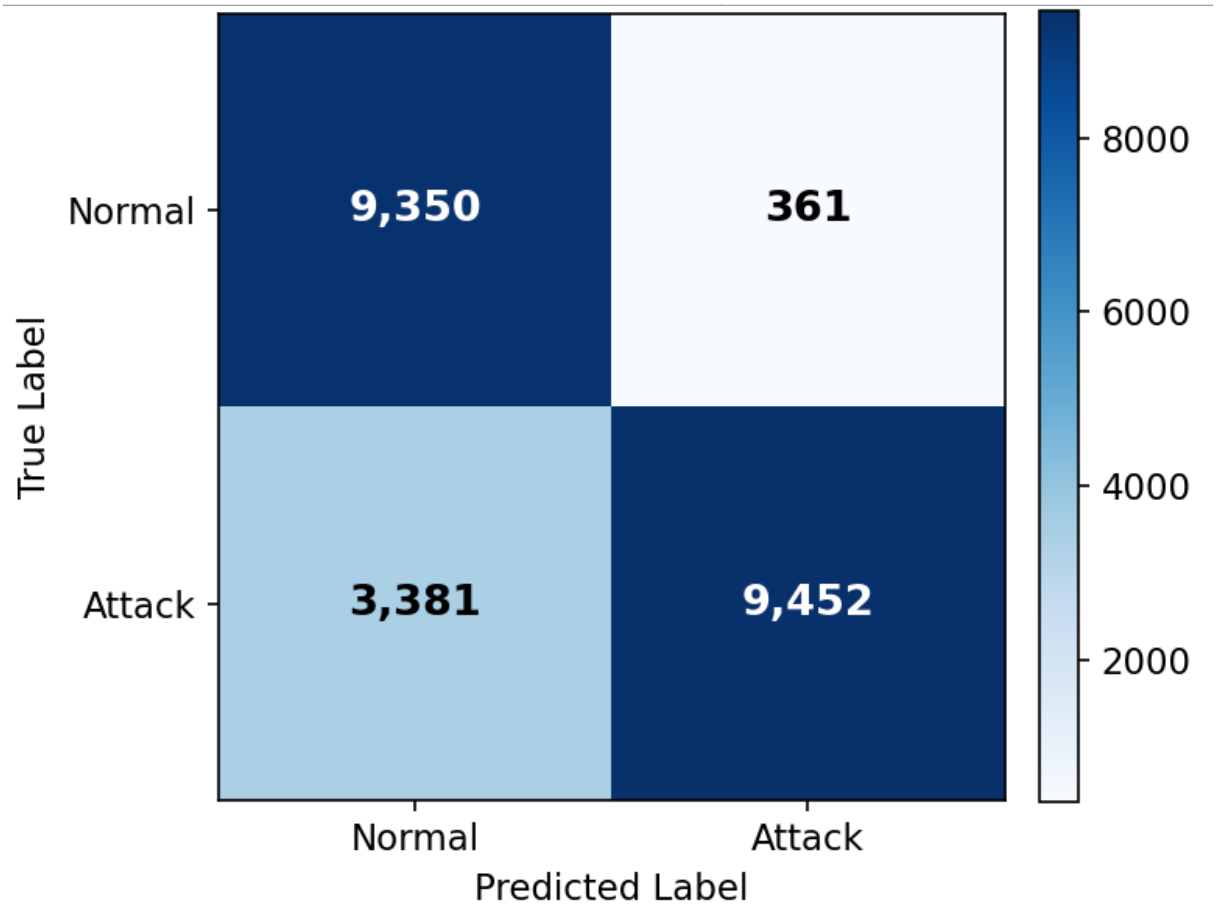


Figure 4. Confusion matrix of the Autoencoder detector on KDDTest+ (positive class: attack).

From these counts the Autoencoder reaches an accuracy of 83.4%, a precision of 96.3%, an attack recall of 73.7%, and an FPR of 3.7%. It has the highest accuracy and recall of the four models (Table 2): its accuracy is 5.3 percentage points above that of SVM and its recall is about 10–12 points higher than that of SVM and Random Forest. Its FPR is higher than those of SVM (0.019) and Random Forest (0.028) but less than half that of Isolation Forest (0.083). Because the model needs only one forward pass per record, its inference time (about 1.7 µs per sample, measured in batch mode) is the lowest of the four, which is attractive for high-speed monitoring.

5.3 ADWIN-triggered incremental updating on a stream

To illustrate the adaptation mechanism, an online logistic classifier was initialized on the first 5,000 training records and then applied to the 22,544 test records one at a time. The prediction error of each record was passed to ADWIN [16]; whenever drift was signaled, the classifier was updated on the 1,000 most recent samples with their labels. Figure 5 shows that the cumulative accuracy is volatile at the start of the stream, as expected from the small sample size, and then settles at about 0.77–0.78 without a sustained decline. This experiment demonstrates the detect-and-update loop, but it should be read with care: cumulative accuracy changes slowly, no non-updating control was run on the same stream, and the update assumed that true labels become available shortly after prediction.

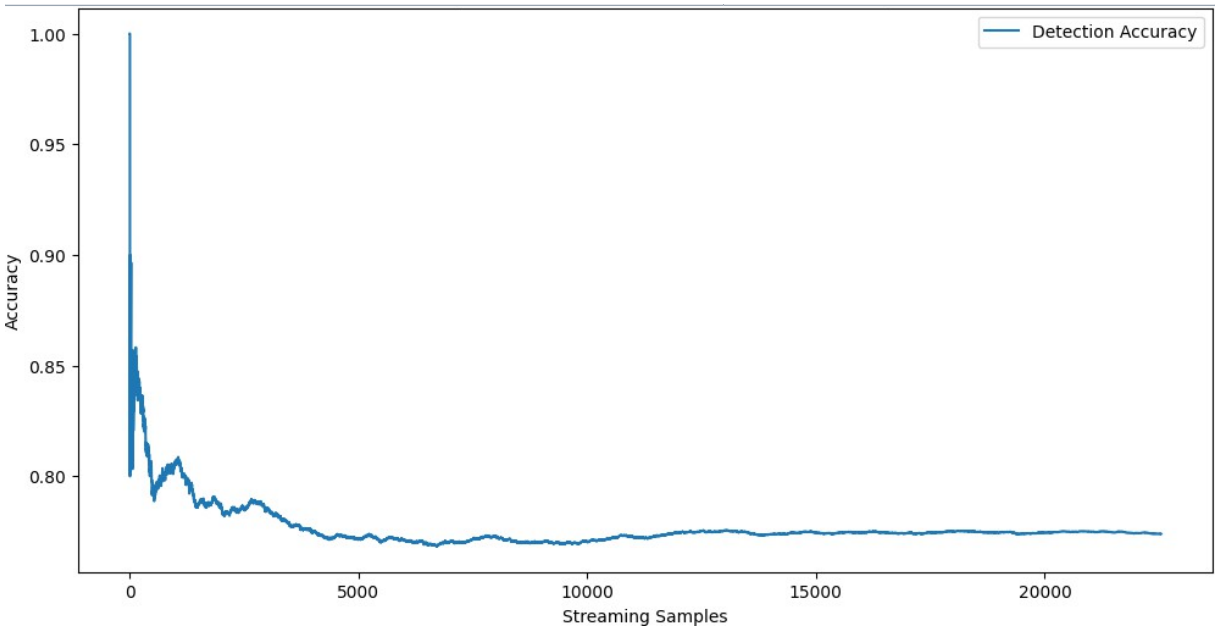


Figure 5. Cumulative detection accuracy of the ADWIN-monitored streaming classifier over the test stream.

5.4 Stability over successive evaluation segments

Figure 6 and Table 3 report the accuracy of each model over the ten consecutive segments of the test stream. SVM and Random Forest declined by 3.2 and 3.1 percentage points, respectively. The Autoencoder started from the highest accuracy (84.9%), ended at 83.1%, and lost only 1.8 points, and it remained within a band of roughly 82–85% across the segments. Isolation Forest lost the least (1.2 points) but at a much lower accuracy level (about 69–73%).

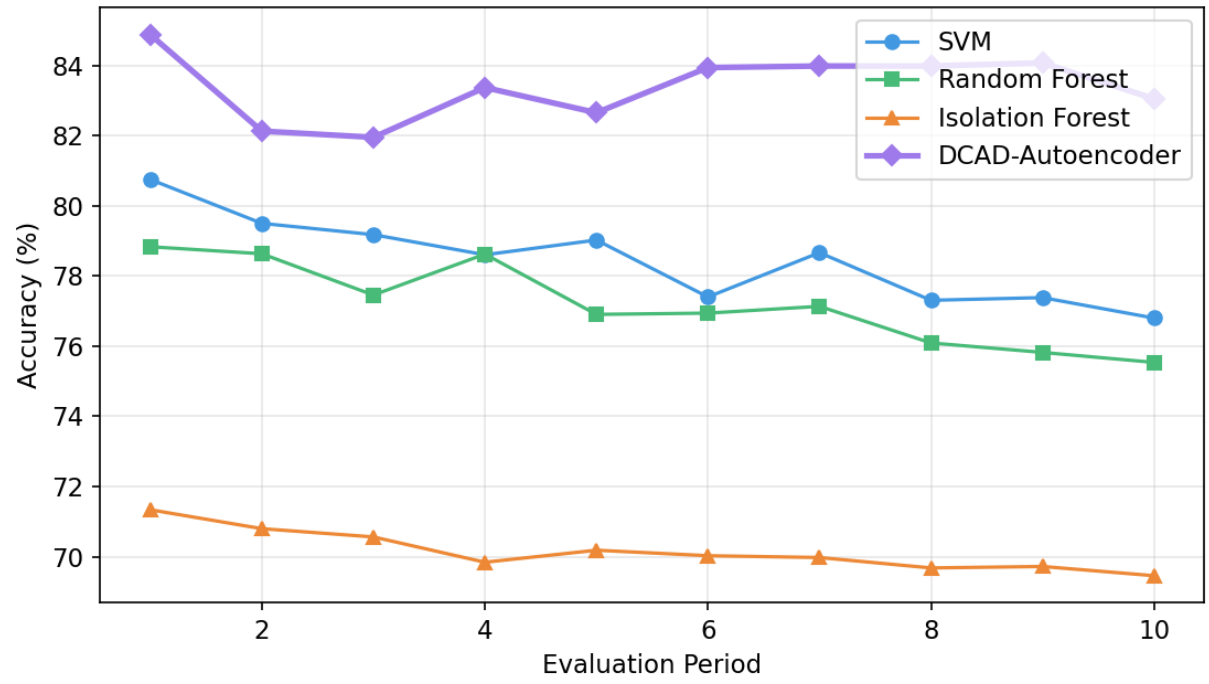


Figure 6. Accuracy of each model over ten consecutive segments of the test stream.

Table 3. Accuracy retention over ten segments.

Model	Initial (%)	Final (%)	Loss (pp)
SVM	80.1	76.9	3.2
Random Forest	79.0	75.9	3.1
Isolation Forest	70.8	69.6	**1.2**
DCAD-AE	**84.9**	**83.1**	1.8

Note: Initial and final: accuracy on the first and tenth segment. Loss in percentage points (pp). Best value per column in bold.

5.5 Discussion and limitations

The results show that an Autoencoder trained only on normal traffic, with a threshold tuned on held-out data, can match or exceed supervised baselines on NSL-KDD in accuracy and recall without seeing attack records during training, while being very fast at inference. Several limitations must be stated clearly.

First, the Autoencoder results in Table II and Table III were obtained with fixed weights: the drift-triggered incremental update and the replay buffer were not applied to the Autoencoder in these experiments, so the smaller accuracy loss reflects the robustness of the one-class representation and not active adaptation. The adaptation loop was demonstrated separately on an online classifier (in the streaming experiment). An end-to-end evaluation of ADWIN-triggered Autoencoder updating with replay is the most important next step, together with measuring the update latency against full retraining.

Second, the ten segments are consecutive slices of KDDTest+, which is not an ordered record of real traffic, so the experiments approximate temporal drift and do not capture the drift of a live network. Third, only NSL-KDD, a dated and simulated data set, was used [14, 15]. Fourth, the threshold was chosen using labeled validation data, so the detector is semi-supervised in practice. Finally, all results come from a single run with a fixed random seed, no confidence intervals or statistical significance tests were computed, and response times were measured in batch mode on a CPU, so they should be compared only qualitatively.

6 Conclusion and future work

This paper presented the DCAD framework for adaptive network anomaly detection under concept drift, built around an Autoencoder trained on normal traffic, an ADWIN drift monitor, incremental updating, and replay-based knowledge retention. On NSL-KDD, the Autoencoder detector achieved the highest accuracy (83.4%) and attack recall (73.7%) among the evaluated models, kept the FPR at 3.7%, had the lowest inference time, and lost only 1.8 percentage points of accuracy across ten successive segments, compared with more than 3 points for SVM and Random Forest. A streaming experiment demonstrated the ADWIN-triggered update loop.

Future work will (i) evaluate the complete framework with drift-triggered Autoencoder updates and a replay buffer, including update latency; (ii) use recent and more diverse data sets such as CIC-IDS2017 [28] and UNSW-NB15 [29], and temporally ordered traffic; (iii) explore LSTM, Transformer, and hybrid detectors and alternative drift detectors; (iv) add explainability and lightweight variants for edge and IoT devices; and (v) investigate federated, privacy-preserving deployment.

References

[1] Aslan Ö, Aktuğ SS, Ozkan-Okay M, Yilmaz AA, Akin E. A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. *Electronics*. 2023;12(6):1333.

[2] Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey. *ACM Comput. Surv*. 2009;41(3):1–58.

[3] Sommer R, Paxson V. Outside the closed world: On using machine learning for network intrusion detection. In: *Proc*. IEEE Symp. Secur. Privacy, Oakland, CA, USA. 2010. p. Secur.

[4] Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A. A survey on concept drift adaptation. *ACM Comput. Surv*. 2014;46(4):1–37.

[5] Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G. Learning under concept drift: A review. *IEEE Trans. Knowl. Data Eng*. 2019;31(12):2346–2363.

[6] Cortes C, Vapnik V. Support-vector networks. *Mach. Learn*. 1995;20(3):273–297.

[7] Breiman L. Random forests. *Mach. Learn*. 2001;45(1):5–32.

[8] Liu FT, Ting KM, Zhou ZH. Isolation forest. In: *Proc*. 8th IEEE Int. Conf. Data Mining, Pisa, Italy. 2008. p. 413–422.

[9] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. *Science*. 2006;313(5786):504–507.

[10] Sakurada M, Yairi T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: *Proc*. MLSDA Workshop Mach. Learn. Sensory Data Anal. 2014. p. 4–11.

[11] Yin C, Zhu Y, Fei J, He X. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*. 2017;5:21954–21961.

[12] Mirsky Y, Doitshman T, Elovici Y, Shabtai A. Kitsune: An ensemble of autoencoders for online network intrusion detection. In: *Proc*. Netw. Distrib. Syst. Secur. Symp. (NDSS), San Diego, CA, USA. 2018.

[13] Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur*. 2000;3(3):186–205.

[14] Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. In: *Proc*. IEEE Symp. Comput. Intell. Secur. Defense Appl., Ottawa, ON, Canada. 2009. p. Comput.

[15] Ring M, Wunderlich S, Scheuring D, Landes D, Hotho A. A survey of network-based intrusion detection data sets. *Comput. Secur*. 2019;86:147–167.

[16] Bifet A, Gavaldà R. Learning from time-changing data with adaptive windowing. In: *Proc*. SIAM Int. Conf. Data Mining, Minneapolis, MN, USA. 2007. p. 443–448.

[17] Kirkpatrick J, et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. U.S.A*. 2017;114(13):3521–3526.

[18] Parisi GI, Kemker R, Part JL, Kanan C, Wermter S. Continual lifelong learning with neural networks: A review. *Neural Netw*. 2019;113:54–71.

[19] Rolnick D, Ahuja A, Schwarz J, Lillicrap T, Wayne G. Experience replay for continual learning. In: *Proc*. Adv. Neural Inf. Process. Syst. (NeurIPS). 2019.

[20] Hu L, Lu Y, Feng Y. Concept drift detection based on deep neural networks and autoencoders. *Appl Sci*. 2025;15(6):3056.

[21] Park H, Kim T, Lee H, Shin D, Shin D, Park M. A continual learning process to detect both previously learned and newly emerging attack. *Appl Sci*. 2025;15(18):10034.

[22] Stocco A, Tonella P. Towards anomaly detectors that learn continuously. In: *Proc IEEE Int Symp Softw Reliab Eng Workshops (ISSREW)*. 2020. p. 201–208.

[23] Li B, Gupta S, Müller E. State-transition-aware anomaly detection under concept drifts. *Data Knowl. Eng*. 2024;154:102365.

[24] Montiel J, et al. River: Machine learning for streaming data in Python. *J. Mach. Learn. Res*. 2021;22(110):1–8.

[25] Paszke A, et al. PyTorch: An imperative style, high-performance deep learning library. In: *Proc*. Adv. Neural Inf. Process. Syst. (NeurIPS). 2019.

[26] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: *Proc*. 3rd Int. Conf. Learn. Represent. (ICLR), San Diego, CA, USA. 2015.

[27] Pedregosa F, et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res*. 2011;12:2825–2830.

[28] Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: *Proc*. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP), Funchal, Portugal. 2018. p. 108–116.

[29] Moustafa N, Slay J. UNSW-NB15: A comprehensive data set for network intrusion detection systems. In: *Proc*. Mil. Commun. Inf. Syst. Conf. (MilCIS), Canberra, ACT, Australia. 2015. p. 1–6.