Incremental Learning in Network Traffic Management: A Fixed-Representation Rehearsal Approach

Francisco Rau*‡, Petia Georgieva†, Carlos Herranz*, Iñaki Val*, Joaquin Perez‡
*MaxLinear, Inc., Valencia, Spain
†Instituto de Telecomunicações, Institute of Electronics and Informatics Eng. of Aveiro (IEETA), University of Aveiro, Portugal
‡Electronic Engineering Department, Universitat de València, Valencia, Spain
Corresponding Author: frau@maxlinear.com

***Abstract*—Network traffic management systems must continuously evolve to new applications without degrading the performance of previously learned categories. This paper evaluates a class-incremental learning strategy that combines rehearsal-based replay with fixed feature representation. Starting from a Fully Connected Neural Network (FCNN) trained on six traffic classes using flow-level features, we freeze all feature-extraction layers and retrain only a newly initialized classification head. Incremental updates are performed using all samples from the incoming class(es) plus a compact, stratified rehearsal buffer containing 10% of the original training set. Experiments on 46,729 labeled flows assess two scenarios: adding one class and adding two classes, and compare against full retraining as an upper bound. The proposed approach reaches 87.90% and 87.72% test accuracy (weighted F1 of 0.877 and 0.875) with minimal forgetting (gaps up to 1.13%). Incremental updates reduce training time by 89-92% and decrease training-data storage requirements by up to 76.88%, supporting deployment in dynamic network environments.**

***Index Terms*—Incremental Learning, Network Traffic Management, Deep Learning.**

I. Introduction

Network Traffic Classification (NTC) based on Machine Learning and Deep Learning (ML/DL) is a key enabler for modern network operation and management, supporting the automated identification of applications and services from traffic-flow observations [1]. In operational environments, however, traffic is highly non-stationary: new applications appear, existing ones evolve, and usage patterns drift over time. As a result, models trained on a closed set of classes progressively become outdated and must be updated to incorporate novel traffic categories while preserving performance on previously learned ones. A straightforward solution is full retraining with all new and historical data, which usually provides an upper bound in precision, but is often impractical due to recurring training cost and the need to retain and access old data at scale [2], [3].

Class-Incremental Learning (CIL) addresses the challenge of updating models sequentially as new classes arrive, without requiring full access to past data. However, CIL remains difficult in practice due to catastrophic forgetting and the stability–plasticity trade-off: simple fine-tuning on new-class data typically degrades performance on old classes, as the optimization process overwrites previous knowledge [4]. Recent studies focusing on NTC indicate that performance often lags behind training from-scratch, prompting the adoption of complex mechanisms to reduce forgetting [2]. Consequently, the state-of-the-art in traffic classification adapts general CIL ideas through sophisticated training recipes, including knowledge distillation, rectification/bias-correction [1], [5], and even generative replay using Generative Adversarial Network (GAN)-based models [3]. Recent directions explore federated frameworks [6], however these methods generally prioritize accuracy at the cost of high system complexity. Despite this progress, there remains a disconnect with operational requirements: network operators prioritize lightweight procedures that bound computational complexity and memory. Some studies, [2], [6], propose strategies that keep the feature extractor fixed (i.e., Fixed-Representation) combined with rehearsal as lower-bound baselines (denoted as FZ-Mem). While Bovenzi et al. [2] include FZ-Mem as a lower-bound baseline within a broader CIL benchmark using packet-level deep learning models, we present, to the best of our knowledge, the first dedicated evaluation of this fixed-representation rehearsal strategy applied to flow-level features, reframing it from a baseline to a viable operational solution where training cost and memory footprint, rather than peak accuracy, are the dominant deployment criteria. Concretely, and unlike complex fine-tuning pipelines, we build on a Fully Connected Neural Network (FCNN), freeze all feature-extraction layers, and update only the classification head using a rehearsal buffer (10% of the base training set), a size previously shown to offer an optimal trade-off between retention and storage cost [2]. This design challenges the trend toward complexity by explicitly prioritizing representational stability and training speed. We validate the approach in two scenarios (6+1 and 6+2), comparing with full retraining. Our results demonstrate that this streamlined strategy achieves competitive accuracy with minimal forgetting, while reducing incremental training time by more than 89% and memory requirements by up to 76.88%, proving that fixed-representation with rehearsal methods is sufficient for network environments.

The remainder of the article is organized as follows: Section II describes the methodology that includes data representation, model architecture, and incremental learning strategy; Section III presents the experimental setup; Section IV discusses the results; and Section V concludes with future directions.

II. Methodology

A. Data Representation and Preprocessing

1) Traffic Windowing: The windowing mechanism segments network traffic into fixed 5-second intervals, a duration optimized to capture burst patterns typical of applications like Roblox and YouTube without introducing the latency associated with longer windows [7], [8]. This windowing enables bidirectional tracking and aggregation of packet-level data into coherent flow[1]-level features. For each incoming packet, the system updates an existing flow or initiates a new one. When a flow exceeds the duration of the current time window, a new window begins while retaining the same flow identifier. This behavior is illustrated by the case of "Flow 5" in Figure 1.

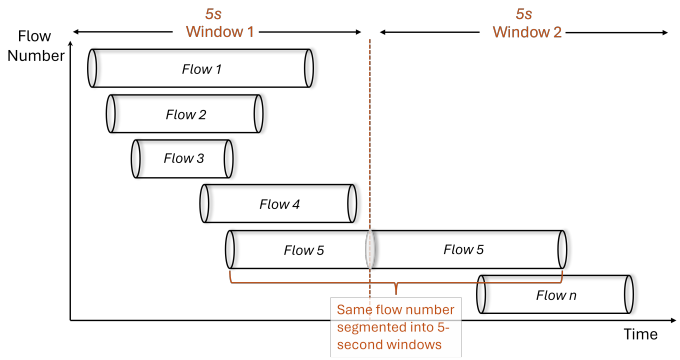

Fig. 1. Flow-based packet segmentation and temporal windowing for feature extraction.

2) Feature Extraction: A set of 29 features is extracted at the flow level within each 5-second time window, organized into categories designed to capture traffic behavior patterns without inspecting payload content. These features encompass basic metrics (duration, byte/packet counts, protocol), packet size statistics (minimum, maximum, mean, standard deviation), and temporal attributes derived from Inter-Arrival Times (IAT). Additionally, the model incorporates flow dynamics metrics (such as burstiness and switching ratio) and rate-based features (such as data transmission intensity), complemented by values smoothed via an Exponential Moving Average (EMA) to ensure temporal stability against short-term fluctuations. The mathematical formulations and details of these features are thoroughly described in [7], where their effectiveness in improving classification accuracy is validated using XGBoost [9], [10]. We utilize the same 29 flow-level features; however, we transition to an FCNN architecture to facilitate CIL. This approach is necessary because XGBoost lacks native support for incrementally adding classes without complete model retraining [11].

B. Base Model Architecture

The base classifier employs an FCNN architecture designed to process 29-dimensional feature vectors extracted from traffic flows. The network utilizes a pyramidal structure with progressively decreasing dimensions to compress the input space and extract abstract features. While tree-based models typically outperform neural networks on tabular data due to their ability to handle irregular patterns and uninformative features [12], this architecture is optimized to challenge that performance gap. The specific layer configuration is:

- Input Layer: Receives the 29-dimensional standardized feature vector.
- Hidden Layer 1: 2048 neurons, ReLU, L2 regularization, ($\lambda = 10^{-2}$), Batch Normalization, Dropout (0.30).
- Hidden Layer 2: 1024 neurons, ReLU activation, L2 regularization, Batch Normalization, Dropout (0.25).
- Hidden Layer 3: 768 neurons, ReLU activation, L2 regularization, Batch Normalization, Dropout (0.20).
- Hidden Layer 4: 512 neurons, ReLU activation, L2 regularization, Batch Normalization, Dropout (0.15).
- Hidden Layer 5: 256 neurons, ReLU activation, L2 regularization, Batch Normalization, Dropout (0.10).
- Output Layer: C neurons (number of classes) with softmax activation function.

The model employs different regularization techniques: L2 regularization penalizes large weights to promote simpler models; Batch Normalization normalizes layer inputs to accelerate training; and Dropout rates decrease progressively across layers, applying stronger regularization in early layers where overfitting risks are higher. The network is optimized using the AdamW optimizer [13] with experimentally defined learning rate 10^{-4} and weight decay 10^{-4}, using sparse categorical cross-entropy as the loss function. The class imbalance is handled through class weighted loss function. Training is controlled by two callbacks: Early Stopping (monitoring validation loss with patience of 15 epochs, restoring best weights) and ReduceLROnPlateau (reducing the learning rate by factor 0.5 with patience of 10 epochs to the minimum 10^{-7}). The model is trained for a maximum of 200 epochs with batch size 128 and 20% validation split.

C. Incremental Learning Strategy

The incremental learning approach combines two complementary techniques to mitigate catastrophic forgetting: rehearsal-based replay and fixed backbone representation. This hybrid strategy balances the stability-plasticity trade-off by preserving learned feature representations while allowing the classification head to adapt to new classes. Figure 2 illustrates the complete pipeline of the incremental learning process.

1) Rehearsal Buffer: The rehearsal mechanism maintains a compact memory buffer $\mathcal{M}$ containing representative samples from previously learned classes. Throughout the paper, the terms "backbone" refer to the feature extraction layers of the FCNN, and "fixed representation" refers to the strategy of freezing these layers during incremental updates. When transitioning from the base model to an incremental update, we construct the rehearsal buffer by randomly sampling 10% of the original training data with stratified selection to preserve the class distribution. Formally, let $\mathcal{D}_{base}$ denote the base-class training set with N samples, $\mathcal{D}_{new}$ the training samples of the newly introduced class(es), and $\mathcal{M}$ the stratified rehearsal buffer from $\mathcal{D}_{base}$, with $|\mathcal{M}| = 0.1 \times N$ samples.

During incremental training, the new training set $\mathcal{D}_{inc}$ is formed by combining the rehearsal buffer ($\mathcal{M}$) with the complete training data of the new class(es):

$$\mathcal{D}_{inc} = \mathcal{M} \cup \mathcal{D}_{new} \tag{1}$$

[1]Flow refers to a sequence of packets between two endpoints (e.g., between a client and a server), identified by the 5-tuple: source/destination IP address, source/destination ports, and protocol.

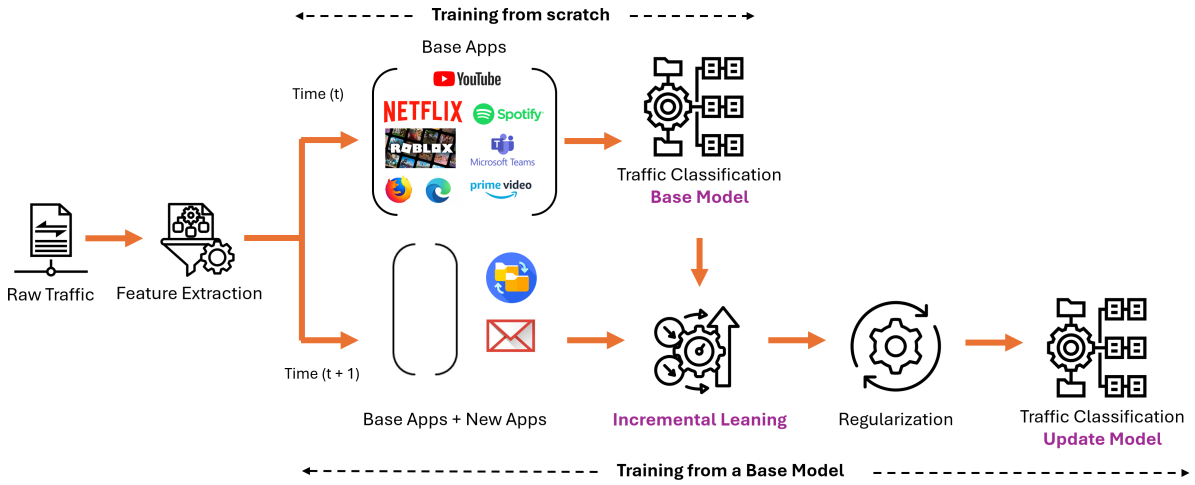

Fig. 2. Stages of the evaluated Incremental Learning Model. The regularization stage includes a rehearsal buffer and a fixed backbone

where $\mathcal{D}_{new}$ represents the full training samples of the newly introduced class(es). This configuration ensures that the model is exposed to exemplars from all classes during incremental updates, preventing complete forgetting of previous knowledge while dedicating sufficient capacity to learn the new categories.

2) Fixed Backbone: A key component of our evaluation is maintaining a fixed feature extraction backbone during incremental updates. The backbone comprises all hidden layers of the FCNN (layers 1-5 with their associated batch normalization and dropout operations), which have been trained in the base classes to extract discriminative traffic representations.

When creating the incremental model, we clone the architecture and weights of all hidden layers from the trained base model. Let $f_\theta : \mathbb{R}^{29} \to \mathbb{R}^{256}$ denote the feature extraction function parametrized by weights θ, mapping the input features to the 256-dimensional representation space of the final hidden layer. Here θ refers strictly to the backbone parameters (hidden layers 1–5) and is distinct from the classification-head parameters ϕ introduced below. During incremental training, θ is frozen.

This freezing strategy offers several advantages: (1) it prevents catastrophic forgetting of the feature representations learned for base classes; (2) it significantly reduces computational cost since gradients need not be computed for the backbone parameters; and (3) it provides training stability by avoiding drastic representational changes that could disrupt classification of old classes.

3) Head Retraining: While the backbone remains frozen, the classification head is replaced and retrained to accommodate the expanded label space. For an incremental update that introduces k new classes to a model previously trained in C classes, a new output layer with $C + k$ neurons is instantiated with random initialization.

The new classification head $g_\phi : \mathbb{R}^{256} \to \mathbb{R}^{C+k}$ is trained on the combined data set $\mathcal{D}_{inc}$ using the same optimization configuration as the base model. The training objective minimizes the cross-entropy loss over all classes:

$$\mathcal{L} = -\frac{1}{|\mathcal{D}_{inc}|} \sum_{(x,y)\in\mathcal{D}_{inc}} \log p\big(y \mid g_\phi(f_\theta(x))\big) \qquad (2)$$

where θ (backbone) remains frozen and only ϕ (classification head) is retrained. The stratified validation split (20% of $\mathcal{D}_{inc}$) ensures that all classes, including those represented only in the rehearsal buffer, are present in both training and validation sets. This approach enables the classifier to learn the appropriate decision boundaries for the new classes while maintaining discrimination capability for previously learned categories through the rehearsal samples.

III. Experimental Setup

A. Dataset Description

In order to test the proposed methodology, we generated network traffic data that contain labeled flows from various applications. Specifically, training and testing data are aggregated from raw files within VLC Data [14]. To expand the evaluation to more applications, we also use the VNAT dataset [15]. Traffic encompasses applications such as Video Streaming (Netflix and Prime Video), Roblox, Web, Teams, Youtube, Spotify, C2 (Command & Control) and File Transfer. Following the feature extraction process, the combined dataset comprises 46,729 flow samples. As shown in Table I, the final distribution shows a moderate class imbalance.

TABLE I
Class Distribution in the Full Dataset

Label	Count
Video_Stream	6,870
Roblox	6,159
File_Transfer	6,112
C2	6,017
Spotify	5,794
Teams	5,438
Youtube	5,175
Web	5,164
Total	**46,729**

Data preprocessing involves removing metadata columns (IP addresses, ports, flow identifiers, window numbers) that do not contribute to traffic pattern recognition. Features are standardized using z-score normalization, the scaler is fitted

on the base training set, and applied consistently across all incremental updates to ensure feature comparability.

B. Evaluation Scenarios

We design two evaluation scenarios to assess the incremental learning approach under different conditions:

- **Scenario A** (6+1): The base model is trained on six classes (Video Streaming, Roblox, Spotify, Teams, Youtube, Web). These six classes correspond to the multimedia, real-time and web-browsing applications originally captured in the VLC dataset [14], which span a heterogeneous mix of streaming, interactive, and bursty traffic patterns and therefore provide a sufficiently diverse base to train a discriminative backbone. File Transfer and C2, drawn from VNAT [15], are reserved as the incremental classes precisely because their traffic patterns (bulk-elephant flows and low-rate control flows) differ from the base set, providing a stricter test of backbone generalization than re-using classes already represented in the base distribution. Subsequently, the model is incrementally updated to incorporate File Transfer as the seventh class. This scenario evaluates the system's ability to integrate a new application category while preserving performance on the original six classes.
- **Scenario B** (6+2): Starting from the same six-class base model, this scenario simultaneously incorporates both File Transfer and C2 as new classes in a single incremental update. This scenario tests the model's capacity to handle a larger knowledge expansion in one step, assessing whether the fixed backbone can accommodate two new classes while maintaining performance on the original categories.

For both scenarios, the full dataset is partitioned using stratified sampling with an 80/20% train/test split, ensuring proportional class representation. The rehearsal buffer is constructed by randomly sampling 10% of the base training data, maintaining the class proportions. Table II summarizes the composition of the training set for each scenario.

TABLE II
DATASET COMPOSITION: TRAINING AND TESTING SAMPLES

Class	Base Model	6+1 Training	6+2 Training	Testing
File Transfer	–	4,890	4,890	1,222
C2	–	–	4,814	1,203
Video Streaming	5,496	550	550	1,374
Roblox	4,927	493	493	1,232
Spotify	4,635	464	464	1,159
Teams	4,350	435	435	1,088
Youtube	4,140	414	414	1,035
Web	4,131	413	413	1,033
Total	**27,679**	**7,659**	**12,473**	**9,346**

C. Implementation Details

- The implementation is developed using TensorFlow 2.19 with Keras API [16], using a computer with AMD Ryzen 5600H with 32 GB RAM. The base model training uses the complete training set of the six base classes with balanced class weights computed using scikit-learn function called *compute_class_weight*.
- For incremental model, hidden layers are cloned from the base model by extracting layer configurations and weights, and then reconstructing identical layers. The same callback configuration (Early Stopping, ReduceLROnPlateau) is applied during incremental training to maintain consistency with base model training.

D. Evaluation Metrics

Model performance is assessed through the following metrics [17]:

- *Accuracy*: measures the proportion of correctly classified instances, TP - True Positives, TN - True Negatives, FP - False Positives, FN - False Negatives:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- Weighted F1-Score: The harmonic mean of precision and recall, weighted by class support to account for class imbalance:

$$\text{F1}_{\text{weighted}} = \sum_{c=1}^{C} \frac{n_c}{N} \cdot \frac{2 \cdot P_c \cdot R_c}{P_c + R_c} \tag{4}$$

where P_c and R_c denote precision and recall for class c, n_c is the number of samples in class c, and N is the total sample count.

Additionally, we measure *Forgetting* as the performance gap on old classes between the upper bound (full retraining) and the incremental model, and *Intransigence* as the performance gap on new classes [18].

IV. RESULTS

A. Base Model Performance

The FCNN base model, trained on six classes, provides a solid foundation for incremental updates. On the test set, the model achieves 88.02% accuracy and 0.8800 weighted F1-score. The model exhibits highest performance on Roblox (F1: 0.9505) and Teams (F1: 0.9406), while Web (F1: 0.8077) and Youtube (F1: 0.8034) show relatively lower scores.

B. Scenario 6+1: Single Class Addition

After incorporating File Transfer as the seventh class, the incremental model demonstrates effective knowledge retention and new class integration. Table III presents the performance comparison between the incremental model and the upper bound (model retrained from scratch on all 7 classes).

TABLE III
PERFORMANCE COMPARISON – SCENARIO A (6+1)

Model	Old Classes	New Classes	Overall
Accuracy			
Incremental (6+1)	0.8581	0.9975	0.8790
Upper Bound	0.8637	0.9975	0.8838
F1-Score			
Incremental (6+1)	0.8682	0.9277	0.8771
Upper Bound	0.8640	0.9943	0.8834

The incremental model achieves 87.90% overall test accuracy and 0.8771 weighted F1-score. Notably, File Transfer achieves excellent classification with 0.9277 F1-score, demonstrating that the fixed backbone produces features that are highly discriminative for File Transfer traffic patterns. Among base classes, Roblox maintains strong performance

(F1: 0.9448), while Web (F1: 0.7756) and Youtube (F1: 0.7873) show somewhat lower scores. The complete per-class report for this scenario is provided in Table VII.

This trend is consistent with the base-model results (Table VI in the Appendix) and is attributable to the intrinsic heterogeneity of these two categories at the flow level: Web aggregates browsing sessions that mix short request/response transactions with long-lived connections to highly diverse content delivery infrastructures, while YouTube traffic is dominated by adaptive-bitrate streaming whose burstiness and inter-arrival patterns overlap with other multimedia classes (e.g., Video Streaming). The 29 aggregated flow-level features therefore yield less separable representations for Web and YouTube than for protocol-stable applications such as Roblox or Teams, and the fixed backbone preserves rather than amplifies this base model behavior.

The forgetting and intransigence analysis reveals:

- *Forgetting (Old Classes):* Minimal impact on performance, with an accuracy gap of 0.56% in accuracy and a marginal positive backward transfer in F1 (incremental F1 slightly exceeds the upper bound, +0.42 pp).
- *Intransigence (New Class):* Performance remained stable for new classes, showing a 0% accuracy gap and a 6.66% F1-score gap.

C. Scenario 6+2: Two Class Addition

The second scenario simultaneously adds File Transfer and C2 as new classes. Despite the increased complexity of distinguishing eight categories, the model maintains competitive performance. Table IV summarizes the results.

TABLE IV
PERFORMANCE COMPARISON – SCENARIO B (6+2)

Model	Old Classes	New Classes	Overall
Accuracy			
Incremental (6+2)	0.8486	0.9588	0.8772
Upper Bound	0.8599	0.9951	0.8949
F1-Score			
Incremental (6+2)	0.8640	0.9067	0.8750
Upper Bound	0.8602	0.9918	0.8943

The incremental model achieves 87.72% test accuracy and 0.8750 weighted F1-score. Both new classes are learned effectively: C2 achieves 0.9101 F1-score, and File Transfer achieves 0.9033 F1-score. The forgetting and intransigence gaps reported below are computed directly from the values in Table IV (overall and new-class accuracy/F1 of the incremental model versus the upper bound), with per-class F1 scores taken from the classification reports in Tables VIII and X:

- *Forgetting (Old Classes):* A minimal accuracy gap of 1.13%, with the incremental F1 again marginally exceeding the upper bound (+0.38 pp), demonstrating robustness against catastrophic forgetting.
- *Intransigence (New Classes):* Limited impact on plasticity, with a decrease in accuracy of 3.63% and an F1-score gap of 8.51%.

D. Computational Cost Analysis

Table V compares the training time between the incremental approach and full retraining (upper bound). The main advantage of the class-incremental approach is that the base model training is a one-time cost. For each subsequent class update, only head retraining is required. In Scenario 6+1, the incremental update takes only 133.04 sec. compared to 1696.39 sec. for full retraining, which means a reduction of 92.16% of the training time. Similarly, Scenario 6+2 achieves an 89.72% reduction.

TABLE V
TRAINING TIME COMPARISON

Stage	Time (s)	Reduction (%)
Base Model (6 classes)	1,085.01	-
Scenario A (6+1)		
Upper Bound (7 classes)	1,696.39	-
Incremental Update (6+1 class)	133.04	92.16
Scenario B (6+2)		
Upper Bound (8 classes)	1,871.50	-
Incremental Update (6+2 classes)	192.43	89.72

E. Memory Overhead Analysis

The rehearsal buffer introduces a bounded memory overhead. For Scenario 6+1, the incremental training set contains 7,659 samples (2.13 MB), compared to 32,569 samples (9.22 MB) for the full 7-class training. This represents a 76.88% reduction in memory requirements. For Scenario 6+2, the incremental set contains 12,473 samples (3.43 MB) versus 37,383 samples (10.56 MB) for full 8-class training, achieving a reduction of 67.50%. The rehearsal buffer itself requires only 0.78 MB (2,769 samples), which represents 10% of the original 6-class training set.

V. CONCLUSION

This paper evaluated a fixed-representation rehearsal approach for class-incremental learning in network traffic classification, motivated by the operational need for bounded training time and limited storage of historical data. Instead of adopting increasingly complex continuous learning pipelines, we froze the FCNN backbone learned in six base classes and retrained only the classification head using a small rehearsal buffer (10% of the base training set) together with all samples from newly introduced classes.

Experimental validation across two incremental scenarios A and B revealed several key findings. First, the approach successfully mitigates catastrophic forgetting, maintaining performance in previously learned classes with accuracy gaps of up to 1.13% compared to complete retraining. Second, the method effectively integrates new classes, achieving F1-scores above 0.90 for newly introduced categories in both scenarios. Third, the computational benefits are substantial: training time reductions exceed 89%, and memory requirements decrease by up to 76.88%, making the approach viable for resource-constrained operational environments.

Future work will focus on evaluating this architecture in continuous learning scenarios with a longer sequence of incremental updates to assess long-term backbone stability, and extending the taxonomy beyond eight classes with more imbalanced and temporally evolving arrivals.

ACKNOWLEDGMENT

This work was supported by the EU Horizon Europe Program in the frame of Marie Skłodowska-Curie Actions

Doctoral Networks (MSCA-DN) project OWIN6G (Optical and wireless sensors networks for 6G scenarios) under Agreement No. 101119624, and by the Grant PID2023-148671OB-I00 funded by MICIU/AEI/10.13039/501100011033 and by ERDF/EU. It was further supported by the European Union - NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project No. BG-RRP-2.004-0005.

APPENDIX

TABLE VI
CLASSIFICATION REPORT TEST - BASE MODEL 6 CLASSES

Class	Precision	Recall	F1-score	Support
Roblox	0.9355	0.9659	0.9505	1232
Spotify	0.8576	0.9249	0.8900	1159
Teams	0.9494	0.9320	0.9406	1088
Video_Stream	0.9079	0.8392	0.8722	1374
Web	0.8210	0.7948	0.8077	1033
Youtube	0.7936	0.8135	0.8034	1035
Accuracy	–	–	**0.8802**	6921
Macro Avg	**0.8775**	**0.8784**	**0.8774**	6921
Weighted Avg	**0.8809**	**0.8802**	**0.8800**	6921

TABLE VII
CLASSIFICATION REPORT TEST - SCENARIO A (6+1)

Class	Precision	Recall	F1-score	Support
File_Transfer	0.8670	0.9975	0.9277	1222
Roblox	0.9253	0.9651	0.9448	1232
Spotify	0.8640	0.9103	0.8866	1159
Teams	0.9395	0.9283	0.9339	1088
Video_Stream	0.9022	0.8261	0.8625	1374
Web	0.8414	0.7193	0.7756	1033
Youtube	0.7951	0.7797	0.7873	1035
Accuracy	–	–	**0.8790**	8143
Macro Avg	**0.8764**	**0.8752**	**0.8740**	8143
Weighted Avg	**0.8787**	**0.8790**	**0.8771**	8143

TABLE VIII
CLASSIFICATION REPORT TEST - SCENARIO B (6+2)

Class	Precision	Recall	F1-score	Support
C2	0.8373	0.9967	0.9101	1203
File_Transfer	0.8859	0.9214	0.9033	1222
Roblox	0.9297	0.9667	0.9479	1232
Spotify	0.8639	0.9094	0.8861	1159
Teams	0.9367	0.9246	0.9306	1088
Video_Stream	0.9056	0.8028	0.8511	1374
Web	0.8456	0.6999	0.7659	1033
Youtube	0.8000	0.7691	0.7842	1035
Accuracy	–	–	**0.8772**	9346
Macro Avg	**0.8756**	**0.8738**	**0.8724**	9346
Weighted Avg	**0.8775**	**0.8772**	**0.8750**	9346

TABLE IX
CLASSIFICATION REPORT - UPPER BOUND (7 CLASSES)

Class	Precision	Recall	F1-score	Support
File_Transfer	0.9911	0.9975	0.9943	1222
Roblox	0.9237	0.9635	0.9432	1232
Spotify	0.8565	0.8861	0.8711	1159
Teams	0.9353	0.9301	0.9327	1088
Video_Stream	0.8891	0.8399	0.8638	1374
Web	0.7735	0.7735	0.7735	1033
Youtube	0.7864	0.7720	0.7791	1035
Accuracy	–	–	**0.8838**	8143
Macro Avg	**0.8794**	**0.8804**	**0.8797**	8143
Weighted Avg	**0.8835**	**0.8838**	**0.8834**	8143

TABLE X
CLASSIFICATION REPORT - UPPER BOUND (8 CLASSES)

Class	Precision	Recall	F1-score	Support
C2	0.9845	1.0000	0.9922	1203
File_Transfer	0.9926	0.9902	0.9914	1222
Roblox	0.9206	0.9692	0.9442	1232
Spotify	0.8535	0.8896	0.8711	1159
Teams	0.9376	0.9256	0.9315	1088
Video_Stream	0.8764	0.8261	0.8505	1374
Web	0.7859	0.7638	0.7747	1033
Youtube	0.7741	0.7681	0.7711	1035
Accuracy	–	–	**0.8949**	9346
Macro Avg	**0.8906**	**0.8916**	**0.8908**	9346
Weighted Avg	**0.8943**	**0.8949**	**0.8943**	9346

REFERENCES

[1] Z. Wu, Y. ning Dong, X. Qiu, and J. Jin, "Online multimedia traffic classification from the qos perspective using deep learning," *Computer Networks*, vol. 204, 2 2022.

[2] G. Bovenzi, A. Nascita, L. Yang, A. Finamore, G. Aceto, D. Ciuonzo, A. Pescape, and D. Rossi, "Benchmarking class incremental learning in deep learning traffic classification," *IEEE Transactions on Network and Service Management*, vol. 21, pp. 51–69, 2 2024.

[3] W. Zhu, X. Ma, Y. Jin, and R. Wang, "Iletc: Incremental learning for encrypted traffic classification using generative replay and exemplar," *Computer Networks*, vol. 224, 4 2023.

[4] D. W. Zhou, Q. W. Wang, Z. H. Qi, H. J. Ye, D. C. Zhan, and Z. Liu, "Class-incremental learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, pp. 9851–9873, 2024.

[5] F. Cerasuolo, A. Nascita, G. Bovenzi, G. Aceto, D. Ciuonzo, A. Pescapè, and D. Rossi, "Memento: A novel approach for class incremental learning of encrypted traffic," *Computer Networks*, vol. 245, 5 2024.

[6] R. Carillo, F. Cerasuolo, G. Bovenzi, D. Ciuonzo, and A. Pescapè, "Explainable federated class incremental learning for encrypted network traffic classification," *Computer Networks*, vol. 269, 9 2025.

[7] F. Rau, C. Herranz, I. Val, P. Georgieva, and J. Perez, "A novel flow-based online network traffic classification using ml," *TechRxiv*, 8 2025.

[8] F. Rau, L. M. Giraldo, C. Herranz, J. Perez, I. Val, and R. Garcia, "Implementing a traffic classifier on embedded soc systems with deep learning processor unit," in *2025 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 9 2025, pp. 1–6.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2016, pp. 785–794.

[10] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, J. Yuan, and D. Cortes, "xgboost: Extreme gradient boosting," 2025.

[11] C. Zhang, Y. Zhang, X. Shi, G. Almpanidis, G. Fan, and X. Shen, "On incremental learning for gradient boosting decision trees," *Neural Processing Letters*, vol. 50, no. 1, pp. 957–987, aug 2019.

[12] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *arXiv*, 2022.

[13] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv*, 1 2019.

[14] F. Rau, C. H. Claveras, I. Val, and J. Perez, "Vlc data: A multi-class network traffic dataset covering diverse applications and platforms," 4 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15121418

[15] MIT Lincoln Laboratory, "Vpn/non-vpn network application traffic dataset (vnat)," 2025, accessed: 2025-03-18. [Online]. Available: https://www.ll.mit.edu/r-d/datasets/vpnnonvpn-network-application-traffic-dataset-vnat

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," *arXiv preprint arXiv:1605.08695*, 2016.

[17] P. Christen, *Evaluation of Matching Quality and Complexity*. Springer Berlin Heidelberg, 2012, pp. 163–184.

[18] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Proc of the Eur. Conf on Computer Vision (ECCV)*, 2018.