2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress

2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics | 979-8-3503-5163-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00054

Unsupervised Real-Time Flow Data Drift Detection Based on Model Logits for Internet of Things Network Traffic Classification

1st Pan Wang
School of Modern Posts
Nanjing University of Posts and Telecommunications
Nanjing, Jiangsu, China
wangpan@njupt.edu.cn

2nd Minyao Liu
School of Modern Posts
Nanjing University of Posts and Telecommunications
Nanjing, Jiangsu, China
1222097606@njupt.edu.cn

3rd Zeyi Li
School of Computer Science
Nanjing University of Posts and Telecommunications
Nanjing, Jiangsu, China
2022040506@njupt.edu.cn

4th Zixuan Wang
School of Modern Posts
Nanjing University of Posts and Telecommunications
Nanjing, Jiangsu, China
2020070135@njupt.edu.cn

5th Xuejiao Chen
School of Communications
Nanjing Vocational College of Information Technology
Nanjing, Jiangsu, China
chenxj@njcit.cn

***Abstract*—In recent years, numerous Internet of Things (IoT) traffic classification techniques based on deep learning have been proposed, demonstrating impressive classification performance. With ubiquitous access to massive heterogeneous IoT endpoints, networks exhibit high levels of dynamism, heterogeneity, and complexity, leading to potential data distribution drift (DDD), which will significantly degrade the performance of IoT network traffic classification. Although numerous scholars have explored various drift detection methods, most studies require real-time stream data labeling. Meanwhile, some unsupervised methods incur high resource consumption and latency when applied to network traffic. To address this issue, we propose an unsupervised real-time drift detection method based on Model Logits (ML-UDD). Based on the Logit scores from the deep learning model's traffic classification/inference process, the model's Maximum Logit Score (MLS) will be calculated as the key metric for detection DDD. This method does not need to rely on the data labels, thus it can explore in an unsupervised manner, which is superior to many other supervised methods in terms of expert experience and labor. Furthermore, an adaptive weighted CUSUM-type rapid detection approach is proposed to detect the DDD based on a sliding time window during the traffic flow proceeding. In addition, this method can further locate the 'Drift Flow' where DDD happened, namely Flow Drift Location. This offers a real-time and efficient method to monitor and identify DDD with minimal resource consumption. The experiments have been conducted on two public and private network traffic datasets. The comparative experiments with other SOTA DDD detection methods have demonstrated that the proposed ML-UDD method can surpass these methods in terms of different performance metrics.**

***Index Terms*—drift detection, data distribution drift, network traffic classification, Internet of Things**

I. Introduction

With the rapid advancement of Internet of Things, cloud computing, big data, and particularly deep learning and high-performance computing technologies, feature learning from massive IoT traffic data has become feasible, opening new possibilities for enhancing IoT traffic classification performance. Deep learning is characterized by three outstanding features: automatic feature extraction, exploration of deep nonlinear features, and reusable classic models in domains such as computer vision, images, text, and speech, making it the mainstream method for traffic classification. In recent years, many IoT traffic classification techniques based on deep learning have been introduced, including methods utilizing CNNs, LSTMs, and GANs, all demonstrating excellent classification capabilities. However, these methods are derived from static models and are unable to adapt to data that changes over time. With pervasive access to massive heterogeneous IoT endpoints, networks have become highly dynamic, heterogeneous, and complex. Frequent updates to active IoT applications, the emergence of new IoT applications, and the continuous delisting of dormant applications can easily lead to Data Distribution Drift (DDD). Clearly, IoT traffic classification models face a practical challenge: when deployed in open environments, the models lose their classification ability in the face of constantly changing applications. Therefore, it is necessary to restart the model training process each time new data becomes available to ensure model performance.

2836-3701/24/$31.00 ©2024 IEEE

DOI 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics62450.2024.00054

However, frequently retraining models from scratch requires a significant amount of computational time and resources. In our dynamic world, this practice quickly becomes intractable to data streams or may only be available temporarily due to storage constraints. Effectively mitigating the issues brought by DDD and providing reliable traffic classification methods has thus become a critical issue.

In response to the aforementioned scenario, continual learning (CL) [1] has emerged as a new paradigm within the academic community. This approach addresses the inability of traditional machine learning/deep learning models to adapt to the gradual accumulation of data streams and the constant emergence of new categories in practical applications. However, in practical applications, manual judgment or periodic model updates are impractical, lacking a theoretical basis and prone to issues such as untimely updates or resource waste. Therefore, drift detection is necessary to precisely determine when to initiate continual learning, allowing for a rapid response when drift occurs, and timely model updates to maintain performance.

Although numerous scholars have explored various drift detection methods, the following issues still persist:

1) **Real-time network traffic data labels are unavailable.** Most existing drift detection methods employ explicit detectors to identify drifts, as exemplified by references [2]–[5]. However, these methods presuppose the availability of true labels for all data instances. Given the constraints of time and resources, it is impractical to label every instance of network traffic promptly.
2) **Low detection efficiency and high latency in real-time stream data.** Some existing data-based methods do not require labels, such as those mentioned in references [6], [7], and instead calculate the distance or similarity between data stream distributions. Yet, this often involves complex statistical methods or distance metrics. Particularly in the context of network traffic classification, where there may be many application categories and data typically has high-dimensional features, comparing the distribution distances of real-time stream data necessitates substantial computational costs and memory overhead. This increases the complexity of detection, leading to issues such as low efficiency and high latency.

According to PAC model theory, under a stable distribution, the model error rate decreases as the number of instances increases. A significant increase in error signifies a change in category distribution, indicating the inadequacy of the decision model [2]. Therefore, drift detection methods based on model performance feedback are reliable. Unlike methods that require labels, such as those based on model error rates, this work presents an unsupervised real-time drift detection method that does not require data labels and ensures the detector can accurately identify drift with low latency, allowing for rapid model adaptation. This method continuously monitors network traffic fluctuations and triggers continual learning upon detecting DDD. The main contributions to this paper are as follows:

1) **An unsupervised real-time drift detection method based on Model Logits is introduced for IoT Traffic Classification.** Based on the Logit scores from the deep learning model's traffic classification/inference process, the model's Maximum Logit Score (MLS) will be calculated as the key metric for detection DDD. This method does not need to rely on the data labels, thus it can explore in an unsupervised manner, which is superior to many other supervised methods in terms of expert experience and labor.
2) **An adaptive weighted CUSUM-type rapid detection method based on a sliding window is employed to detect DDD during the IoT traffic flow proceeding.** This provides a flexible and robust means to monitor and identify changes within data streams. In addition, this method can further locate the 'Drift Flow' where DDD happened, namely Flow Drift Location. This offers a real-time and efficient method to monitor and identify DDD with minimal resource consumption.
3) **The experimental evaluations on public and private network traffic classification datasets are conducted** to validate the performance of the proposed unsupervised real-time drift detection method based on model Logits.

The subsequent organization of this article is as follows: Section II introduces related work, Section III provides a detailed description of our proposed unsupervised real-time drift detection method, Section IV validates the method's performance through experiments and results analysis, and Section V concludes the paper.

II. Related Work

The literature [8] categorizes the research outcomes of DDD into three types: detection, understanding, and adaptation. Among these, understanding DDD serves as an intermediary phase and is not studied independently; it is generally integrated into the process of DDD detection.

A. Data Distribution Drift Detection

Drift detection methods possess proactive mechanisms that issue timely alerts when drift occurs, offering targeted responses. These methods can be categorized into error rate-based drift detection, data distribution-based drift detection, and multiple hypothesis testing drift detection. Error rate-based algorithms constitute the largest category, focusing on tracking changes in model error rates. A statistically significant increase or decrease triggers a drift alert. The Drift Detection Method (DDM) [2] is one of the earliest and most commonly used algorithms, utilizing online error rates and their standard deviation as detection metrics. ADaptive WINdowing (ADWIN) [4] adjusts the size of two windows based on the rate of change between them, with the difference in average error trending towards a normal distribution; a difference exceeding the threshold indicates DDD. Recent

studies include the Ultimately Simple Drift Detector (USDD) by Maciel et al. [9], which maintains a window with new data instances, considering drift to have occurred if the window's error rate exceeds 50%. Yu et al. [5] introduced Active Drift Detection based on Meta learning (Meta-ADD), extracting the average error rate difference between two windows as a meta-feature and training a machine learning model to detect drift categories. Guo et al.'s CDT-MSW [10] method accurately identifies drift types and subcategories through detection, growth, and tracking phases. The detection phase employs single static and dynamic windows to pinpoint drift locations, characterizing the degree of distribution change by the difference in real-time accuracy between fixed and sliding windows. However, all the aforementioned algorithms rely on changes in model error rates or accuracy metrics to detect DDD, requiring labels for all incoming data instances, which is clearly impractical in real-world applications.

Data distribution-based drift detection constitutes the second major category of methods, utilizing distance functions or metrics to quantify the differences between historical and new data distributions, thereby detecting DDD. Castellani et al. [6] proposed a task-sensitive drift detection framework that detects drift by monitoring the distance of new samples to centroids. The literature [7] introduced the Centroid Distance Drift Detector (CDDD), which retains historical block information and analyzes the distances between centroids of entire data blocks in an unsupervised version without class division. Li et al. [11] modeled drift trends and proposed Data Distribution Generation for Predictable Concept Drift Adaptation (DDG-DA), an algorithm that uses weighted sampling and a differentiable distribution distance equivalent to KL divergence to train future data generators, thus predicting drift trends. These algorithms address the root cause of DDD, i.e., distribution drift, without the need for new sample labels. However, comparing the distribution distances of new and old data instances requires significant computational and memory overhead, increasing the complexity of detection and leading to issues such as low detection efficiency and high latency.

Multiple Hypothesis Testing Drift Detection combines techniques from the previous two methods in several ways and can be divided into two categories: parallel multiple hypothesis testing and hierarchical multiple hypothesis testing. For parallel multiple hypothesis testing methods, Just-In-Time adaptive classifiers (JIT) [12] were the first to set up multiple drift detection hypotheses in this manner, with the core idea of extending CUSUM charts to detect average changes in features of interest to the learning system. Hierarchical multiple hypothesis testing methods typically use existing detection layer methods to detect drift, then apply additional hypothesis testing, known as the validation layer, to obtain a second verification of the detected drift. Examples include Hierarchical Change Detection Tests (HCDTs) [13] and Hierarchical Linear Four Rate (HLFR) [14]. Adding a validation layer after detection indeed helps improve accuracy but at the cost of significant time performance, which may pose substantial issues in real-time monitoring. Moreover, using multiple hypothesis testing in drift detection may increase the risk of false positives.

B. Data Distribution Drift Adaptation

Drift adaptation methods do not perform drift detection; instead, they passively adapt to environmental changes by ensuring model performance. These methods are divided into single model optimization and ensemble retraining. Single model optimization strategies mainly include simple retraining and model adjustment. Simple retraining involves training a new model with the latest data to replace the outdated one. Baier et al. [15] followed this strategy, demonstrating the importance of continuous monitoring and showing that frequent retraining and model adaptation are necessary to ensure model performance. Dong et al. [16] proposed a sample filtering method based on drift regions. Model adjustment strategies develop models that adaptively learn from constantly changing data. Yu et al. [17] based their approach on meta-learning and introduced the Learn-to-Adapt framework to address the problem of unlabeled DDD. The Weighted Incremental-Decremental SVM (WIDSVM) [18] combines windowing and weighting methods, adding new samples through incremental learning and imposing constraints on new input vectors while relaxing old constraints to reduce the relevance of old information.

Ensemble retraining is a common approach within drift adaptation strategies, saving considerable effort in retraining new models for recurrent DDD by preserving and reusing old models. Examples include Probability Threshold Bagging (PT-Bagging) [19], Kappa Updated Ensemble (KUE) [20], and Mix-Drift Data Learning (MDDL) [21].

III. Methodology

This chapter focuses on the proposed methodology. Section III-A presents the overall framework and Section III-B describes the drift detection methodology in detail.

A. Framework

The overall implementation process of the unsupervised real-time drift detection method based on Model Logits in network traffic classification is illustrated in Fig. 1 We have established two types of storage: Long-term Memory (L) and Short-term Memory (S). S is used to store data instances obtained by the device during the current interval, along with the corresponding Model Max Logit Scores (MLS) for these instances. This recent data is preserved temporarily for processing to check for the occurrence of drift. L is used to store a portion of the data samples for each category accumulated so far, which will be retained long-term for training and updating the model, enabling it to learn all categories. The framework is primarily divided into the following four stages.

1) Initial Model Training: Based on the requirements of traffic classification services, real network traffic data is acquired to complete traffic data engineering and feature engineering, resulting in initial training data. This data is used to train a classification model capable of accurately

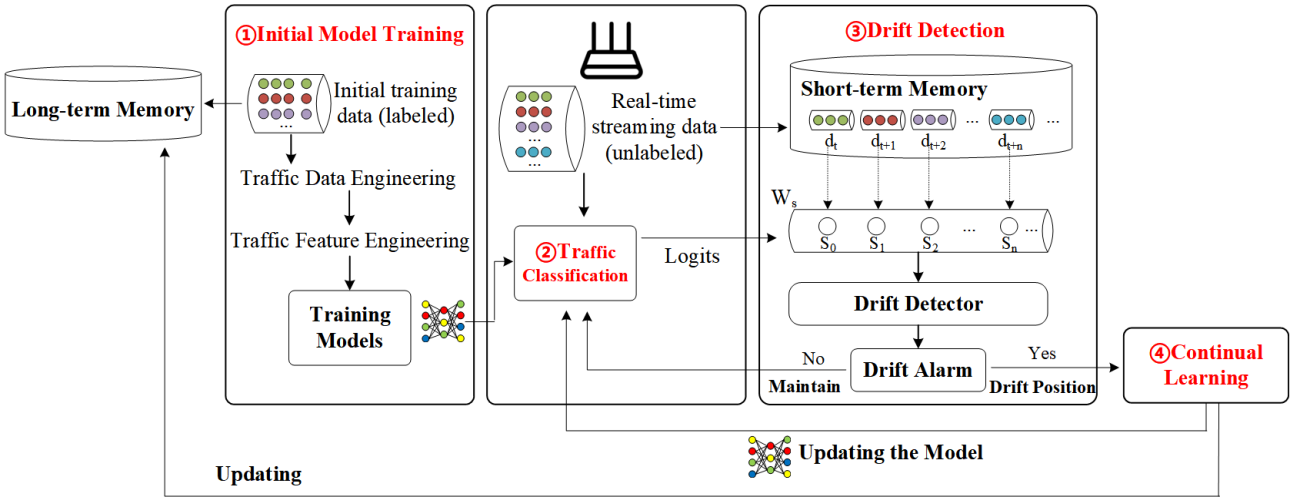


Fig. 1. The overall framework of ML-UDD.

identifying the application categories of traffic. Traffic data engineering includes traffic collection, preprocessing, and annotation. Network traffic data is collected and preprocessed within the serviced network, involving traffic distillation and reassembly to remove noise such as background traffic and redundant packets. Traffic annotation involves labeling the traffic with the correct application type, serving as the true label for model training. Traffic feature engineering encompasses feature extraction, selection, representation, and reduction. Representative traffic features are extracted and selected from raw packets, including packet-level, flow-level, and statistical features. Dimensionality reduction techniques such as autoencoders and Principal Component Analysis are applied to compress features, further reducing model parameters and training complexity. After forming the traffic features suitable for model training, an appropriate method is chosen to represent the traffic feature information, including two-dimensional vectors, images, byte sequences, etc., to create a feature set for model training.

2) Traffic Classification: The original classification model is deployed in network devices such as routers to classify dynamic real-time traffic data within the network. At this stage, the traffic data is obviously unlabeled.

3) Drift Detection: The latest data instances are stored in Short-term Memory (S), and the Max Logit Score obtained by the model during classification of these instances is output to a sliding window. The drift detector determines whether DDD has occurred based on the detection algorithm. If no drift is identified, no action is taken; if drift is detected, an alert is triggered, and the drift location is output, initiating continual learning. The specific drift detection algorithm will be described in detail in the following subsections.

4) Continual Learning: Based on the drift detector's output of the drift location, new data post-drift is collected. Then, using this data, the classification model is updated through a continual learning process. CL is a method where the model is continuously updated as new data becomes available, allowing it to adapt to changes in the data distribution without the need for complete retraining. One of the main challenges of learning from dynamic data distributions is catastrophic forgetting [22], where adapting to new distributions often leads to a significant decrease in the ability to capture old distributions. As a simple baseline, retraining all old training samples can easily solve this challenge, but it will incur significant computational and storage costs (as well as potential privacy issues). The main goal of CL is to ensure that model updates are resource-efficient, ideally approaching only learning new training samples. In this case, employing an appropriate CL method (such as Regularization-based and Replay-based continual learning) [23] to integrate new information while preserving knowledge gained from previous data helps mitigate catastrophic forgetting. This method ensures that the classifier remains up-to-date and can accurately identify both new and old applications in an open environment. By continuously updating the model, even as the underlying data changes, we can maintain real-time traffic classification performance.

The framework is capable of online monitoring of network traffic fluctuations and swiftly detecting impacts on existing models. Upon detecting DDD, it triggers continual learning. This ensures that the classifier remains stable during periods of data stream inactivity and can rapidly adapt to new data distributions when DDD occurs, offering strong real-time performance.

B. Unsupervised Real-Time Drift Detection Based on Model Logits

To facilitate rapid drift detection in real-time traffic data without the need for labels, we propose an adaptive weighted CUSUM-type quick detection method based on a sliding window, inspired by the original work of Haque et al. [24]. This method is executed on the device as soon as a new data instance $X = (X_i)$ becomes available. Initially, a metric

is estimated for each new instance X_i to help quantify the difference in model performance. Unlike the most commonly used methods based on model error rates, we select the Maximum Logit Score (MLS) of the running traffic classifier as the metric, as it can indirectly represent the model's classification accuracy without known data labels. Moreover, the classifier's Logit scores, which are calculated during the online classification process, do not incur additional computational overhead. Logits refer to the raw, unprocessed scores from the model's output layer, indicating the model's preference for each category; the higher the score, the more confident the model is in its prediction for that category. This means that while using the traffic classifier to identify the class of an instance X_i, we obtain Logits for each category: $L(y_c \mid X_i)$, where y_c is one of the C possible classes. The maximum value of $L(y_c \mid X_i)$ is the model's MLS.

The current model's MLS s_i for instance X_i is added to a sliding window W_s. $n = |W_s|$, that is n represents the current length of W_s, with a maximum size set to N. Once the maximum size is of W_s is reached, adding a new element to W_s implies removing the old one. Subsequent drift detection calculations are performed by dividing W_s into two sub-windows: W_a and W_b, where W_b contains the new data. Setting the parameter Δ, for each data k within W_s from Δ to $n-\Delta$, such that $(s_1 : s_k)$ forms W_a and $(s_{k+1} : s_n)$ forms W_b, as shown in Fig. 2. To avoid unnecessary calculations with each new insertion causing system bottlenecks, a trigger mechanism is set up, limiting the number of executions. The steps are as follows:

1) Check whether each sub-window W_a and W_b contains at least Δ elements to maintain statistical properties of a distribution, i.e., ensuring that W_s has at least 2Δ elements. If this condition is met, proceed to the next step.
2) Set a random number r between (0,1) and evaluate whether $r \leq e^{-2s_i}$ holds true; if it does, proceed to the next step. This implies that the higher the MLS, indicating greater confidence in the model's prediction for that category, the lower the likelihood of performing drift detection.
3) When DDD occurs, the MLS will decrease; hence, only negative direction changes of the MLS need to be detected. Calculate the average MLS in W_a and W_b, denoted as m_a and m_b respectively. A change point is searched only when $m_a - m_b \geq \lambda$. According to literature [4], λ can be calculated using the following formula:

$$m = \frac{1}{1/n_0 + 1/n_1} \tag{1}$$

$$\delta' = \frac{\delta}{n} \tag{2}$$

$$\lambda = \sqrt{\frac{1}{2m} \cdot \ln \frac{4}{\delta'}} \tag{3}$$

Here, n_0 and n_1 represent the lengths of sub-windows W_a and W_b, respectively, while n denotes the length of W_s. δ represents sensitivity to change, and we aim to keep the global error below δ. The value of δ can be adjusted as needed; smaller values make detection more sensitive to negative changes in the MLS. The purpose of δ' is to mitigate issues related to multiple hypothesis testing.

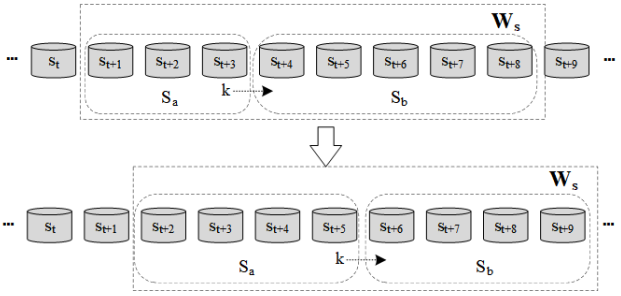


Fig. 2. The sliding window.

Upon meeting the aforementioned criteria, an adaptive weighted CUSUM-type quick detection is executed. The advantage of the CUSUM algorithm lies in its real-time nature and sensitivity, allowing for early detection of process changes to facilitate appropriate control measures. With the target value set as m_a, we calculate the weighted cumulative sum of differences at k, denoted as P_k. Since the window may contain a small amount of noise in the middle, and the actual distribution drift data is typically the newly incoming data at the end of the window, corresponding weights are added, with the formula as follows:

$$P_k = \sum_{i=k+1}^{n} \max\left(\frac{i}{n}(m_a - s_i), 0\right) \tag{4}$$

where m_a is the average of the MLS in the sub-window W_a, n is the length of the current W_s, and s_i is the MLS of the instance.

Compute all P_k within the range $\Delta \leq k \leq n-\Delta$, and select the maximum value P_f. Let $k_{\max}$ correspond to P_f, and set a threshold T_h. If $P_f > T_h$, a drift warning point M is detected at position $k_{\max}$. Continue evaluating k further, repeating the process. If two consecutive drift warning points are detected, report DDD, with the drift position at M.

IV. Experiments

A. Experimental Setting

1) Introduction to Datasets: **Public dataset:** The MIRAGE2019 [25] dataset is designed for mobile traffic analysis and contains real data related to mobile applications. This dataset encompasses multiple domains including social, video, music, gaming, and news. It provides detailed information for each traffic packet, including timestamps, source and destination addresses, protocols, and lengths. Additionally, the dataset includes a label for each traffic packet, ensuring comprehensiveness and coherence. **Private dataset:** Our proprietary NJUPT2023 dataset comprises 91 features. During the data collection phase, for gaming, the traffic generated was

TABLE I
EXPERIMENTAL ENVIRONMENT.

CPU	13th Gen Intel Core i7-13700KF
Memory Capacity	32GB RAM
Graphics Card	NVIDIA GeForce RTX 4080
Python Version	3.10

concentrated during the game loading process, for which we employed automated scripts to crawl. For other applications, we opted for manual collection. On personal terminals, we utilized PCAPDroid to tag network traffic, while on routers, we captured it using Wireshark, followed by a comparison and filtration of both. After filtering, we computed the features for each application's PCAP file on the router, calculating network traffic characteristics. Moreover, the experiment also provides backend statistical information that includes details on network location services, security components, and system log services.

2) Experimental Environment: The experimental environment is AMD Ryzen 3600, 16GB RAM, NVIDIA GTX 3060, CUDA 7.5 and CDNN 10.5. In this paper, Python3 is the main programming language. As shown in Table I.

B. Drift Dataset Construction and Validation for Network Traffic

In real-world network traffic data, even when we are aware of occurrences such as application upgrades or the emergence of new applications, it is still challenging to confirm whether DDD has occurred and where exactly it has taken place. Therefore, we construct accurately identifiable drift scenarios from the network traffic dataset to validate the performance of the proposed drift detection methods. Initially, we select data from 15 application categories within the dataset and divide it into training and test sets. The training set serves as initial data for training a CNN model, while the test set is considered drift-free. The remaining categories are treated as new applications and are randomly divided into 10 change stages, thus constructing 10 drifts, which are incorporated into the drift-free test set. The newly constructed test set containing 10 drifts is referred to as the drift test set. Both drift-free and drift test sets are used to evaluate the CNN model, with results depicted in Fig. 3 and Table II. It is observed that the initial classification model achieves classification accuracy of 0.96 and 0.78 on the two datasets for data without distribution drift. However, when classifying distribution drifted data, the accuracy significantly decreases to 0.78 (a drop of 18.75%↓) and 0.69 (a drop of 11.54%↓), respectively, failing to recognize the newly added applications.

Fig. 4 illustrates the variation in model accuracy as the number of test samples increases on both the drift-free and drift test sets. The red dashed line represents the location of drift. From the graph, we observe that the model accuracy in the drift-free test set gradually stabilizes as the number of test samples increases. However, with the inclusion of distribution drift data, the model accuracy significantly decreases. These experiments collectively demonstrate that the constructed drift

TABLE II
MODEL CLASSIFICATION RESULTS FOR THE CASE OF YES/NO DRIFT EVENTS IN DIFFERENT DATASETS.

DATASET	Testing data	Precision	Recall	F1	Accuracy
MIRAGE2019	No Drift	0.9593	0.9505	0.95379	0.9643
	Drift	0.7064	0.8391	0.7412	0.7760
NJUPT2023	No Drift	0.7848	0.7759	0.7776	0.7759
	Drift	0.6198	0.6787	0.6426	0.6784

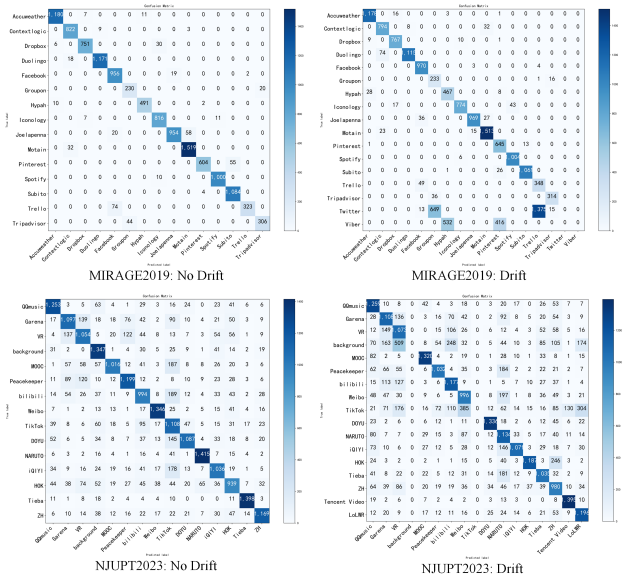


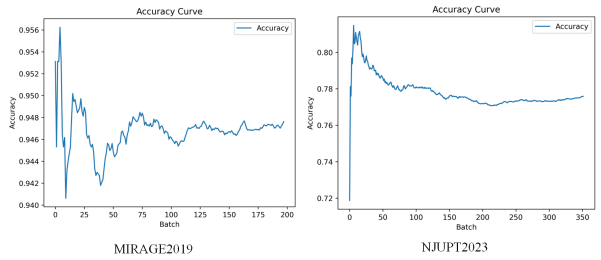
Fig. 3. Model classification results for the case of yes/no drift events in different datasets.

test set indeed experiences DDD at artificially inserted positions, resulting in a notable performance drop. Based on these results, we employ the same approach to insert 50 random drifts into the originally drift-free test dataset, creating a drift test dataset with 50 drift points, where all drift locations are known. This dataset will be used to validate the performance of the drift detection method in the subsequent sections.

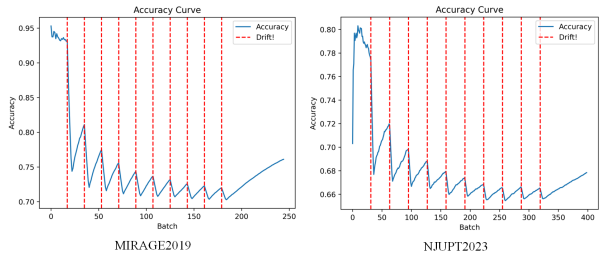
C. Experimental Results and Analysis

This section presents the results of the experimental runs, including an analysis of drift detection accuracy on two network traffic datasets. A CNN is employed as the base learning model for traffic classification, and the distribution drift datasets obtained using the previously described methods are tested. The comparative methods for drift detection are DDM [2], ADWIN [4], and CDDD [7]. The first two methods require the use of all test traffic data labels, while the third is unsupervised and does not require label acquisition.

To assess the performance of drift detectors, the quantity and location of DDD identified by each method are analyzed. Given the sparse number of drift points in the drift test set, 40 repeated experiments were conducted, with randomly constructed distribution drift data for each iteration. The average distance to the true drift points for true positive drift detections (μD), as well as the total count of false negatives (FN) and



(a) No Drift



(b) Drift

Fig. 4. Variation of model accuracy with instances in case of yes/no drift events in different datasets.

false positives (FP) across the 40 experiments, were recorded. These were used to calculate the corresponding Precision, Recall, and F1 scores as metrics for evaluating drift detection performance. Precision is defined as the proportion of true drifts among the detected drifts, given by $TP/(TP + FP)$. Recall, provided by $TP/(TP + FN)$, is the proportion of existing drifts correctly detected by each method. F1 is the harmonic mean of Precision and Recall, defined as

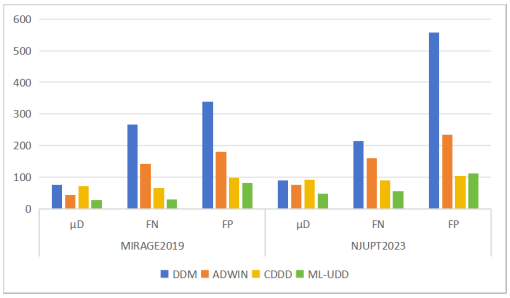
$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

Higher values of these three metrics indicate better performance of the respective drift detection methods.

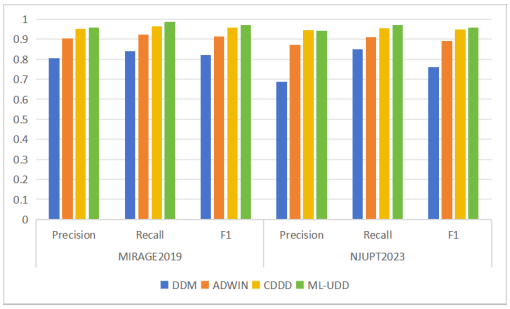
For the calculation of true positive detections, we considered DDD detected within a 2% range before and after the actual drift points. For instance, in the NJUPT2023 dataset, when a DDD spanned 1K instances, detections up to 200 instances after the precise point were deemed true positives. Pinpointing the exact location of DDD occurrence is challenging, necessitating a set margin of positional deviation. The final results are presented in Table III and Fig. 5. In most tests, our proposed method's average distance for correct detection was closest to the exact point compared to the three benchmark methods. False negatives, related to undetected existing drifts or omissions, showed that our method outperformed the comparative methods in this metric. False positives, also known as false alarms, indicated that our method, overall, performed comparably with the CDDD method and surpassed the DDM and ADWIN methods. Furthermore, based on the metrics of Precision, Recall, and F1, our method proved to be the best in terms of Recall and F1, and was on par with CDDD in Precision.

TABLE III
COMPARISON OF DRIFT DETECTION PERFORMANCE OF DIFFERENT METHODS.

DATASET	Method	Labeled/ Unlabeled	μD	FN	FP	Precision	Recall	F1
MIRAGE2019	DDM [2]	Labeled	75.67	267	338	0.8050	0.8394	0.8218
	ADWIN [4]	Labeled	43.36	143	181	0.9025	0.9214	0.9119
	CDDD [7]	Unlabeled	72.44	66	97	0.9498	0.9653	0.9575
	ML-UDD	Unlabeled	**28.38**	**29**	**81**	**0.9589**	**0.9849**	**0.9717**
NJUPT2023	DDM [2]	Labeled	90.06	215	557	0.6880	0.8510	0.7608
	ADWIN [4]	Labeled	76.24	160	235	0.8723	0.9093	0.8904
	CDDD [7]	Unlabeled	92.61	89	**104**	**0.9456**	0.9531	0.9493
	ML-UDD	Unlabeled	**47.56**	**56**	112	0.9424	**0.9703**	**0.9562**



(a) Comparison of μD, FN and FP for different methods, smaller values indicate better method performance.



(b) Comparison of Precision, Recall, and F1 for different methods, with larger values indicating better method performance.

Fig. 5. Comparison of drift detection performance of different methods.

To conduct a more comprehensive performance evaluation, we compared various evaluation metrics from the experiments, including CPU average usage rate (%), GPU average usage rate (%), and memory usage rate (%). As shown in Table IV, it is evident that under identical conditions, supervised drift detection methods outperformed unsupervised methods in terms of CPU average usage rate, GPU average usage rate, and memory usage rate, at the cost of requiring all data labels. In contrast, the ML-UDD method in the unsupervised approach excelled in these three indicators. Therefore, for large-scale IoT networks with high traffic and diverse application categories, the method proposed in this paper is less resource-intensive, which holds significant practical value.

Based on the experimental results reported above regarding drift identification, we conclude that our proposed method is the best-performing one. Although its advantages over the CDDD method in terms of Precision, Recall, and F1 scores, as well as the rate of false positives detected, are not extremely

TABLE IV
RESOURCE CONSUMPTION PERFORMANCE OF DIFFERENT METHODS.

DATASET	Method	Labeled/ Unlabeled	Average CPU usage(%)	Average GPU usage(%)	Memory usage(%)
MIRAGE2019	DDM [2]	Labeled	3.92	57.08	12.2
	ADWIN [4]	Labeled	4.10	61.57	14.5
	CDDD [7]	Unlabeled	6.53	76.24	19.4
	ML-UDD	Unlabeled	4.25	67.89	18.9
NJUPT2023	DDM [2]	Labeled	4.02	69.93	15.3
	ADWIN [4]	Labeled	4.12	75.59	18.0
	CDDD [7]	Unlabeled	6.75	89.94	24.6
	ML-UDD	Unlabeled	4.22	80.60	22.8

pronounced, it significantly outperforms CDDD in the average distance from detected drift locations to the actual drift points in true positive drift detection. This means that our proposed method detects drift locations more accurately and responds to drift occurrences more swiftly.

V. CONCLUSION

To tackle the common challenge of DDD in deep learning traffic classification, we introduce an unsupervised, real-time drift detection method called ML-UDD. It utilizes classifier Logit scores from deep learning models to calculate the Model Max Logit Score, a metric for drift detection. This approach eliminates the need for real-time data labels and model error rates, effectively handling unlabeled network traffic data. It also incorporates an adaptive weighted CUSUM-type method for quick DDD detection, using a sliding window to efficiently monitor and identify data stream changes with minimal resources. Our method facilitates online network traffic monitoring and rapid model impact detection, prompting continuous learning when drift is detected. It ensures classifier stability during static data streams and swift adaptation to new data distributions upon DDD, showcasing robust real-time performance. Testing on MIRAGE2019 and NJUPT2023 datasets confirms the feasibility of our unsupervised ML-UDD method, which uses the Model Logit Score (MLS) for drift detection without needing data labels for model error calculations. Compared to other methods, ML-UDD shows lower false positive and negative rates. While its false positive rate advantage is modest against CDDD, ML-UDD's average distance from detected drift locations to actual drift points is significantly smaller, indicating a more accurate and faster response to drifts. Additionally, our method is less resource-intensive, which holds significant practical value.

REFERENCES

[1] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[2] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, 2004, pp. 286–295.

[3] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, vol. 6. Citeseer, 2006, pp. 77–86.

[4] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[5] H. Yu, Q. Zhang, T. Liu, J. Lu, Y. Wen, and G. Zhang, "Meta-add: A meta-learning based pre-trained model for concept drift active detection," *Information Sciences*, vol. 608, pp. 996–1009, 2022.

[6] A. Castellani, S. Schmitt, and B. Hammer, "Task-sensitive concept drift detector with constraint embedding," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 01–08.

[7] J. Klikowski, "Concept drift detector based on centroid distance analysis," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[8] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[9] B. I. F. Maciel, J. I. G. Hidalgo, and R. S. M. de Barros, "An ultimately simple concept drift detector for data streams," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 625–630.

[10] H. Guo, H. Li, Q. Ren, and W. Wang, "Concept drift type identification based on multi-sliding windows," *Information Sciences*, vol. 585, pp. 1–23, 2022.

[11] W. Li, X. Yang, W. Liu, Y. Xia, and J. Bian, "Ddg-da: Data distribution generation for predictable concept drift adaptation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, 2022, pp. 4092–4100.

[12] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers—part i: Detecting nonstationary changes," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1145–1153, 2008.

[13] C. Alippi, G. Boracchi, and M. Roveri, "Hierarchical change-detection tests," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 2, pp. 246–258, 2016.

[14] S. Yu and Z. Abraham, "Concept drift detection with hierarchical hypothesis testing," in *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 2017, pp. 768–776.

[15] L. Baier, J. Reimold, and N. Kühl, "Handling concept drift for predictions in business process mining," in *2020 IEEE 22nd Conference on Business Informatics (CBI)*, vol. 1. IEEE, 2020, pp. 76–83.

[16] F. Dong, J. Lu, Y. Song, F. Liu, and G. Zhang, "A drift region-based data sample filtering method," *IEEE Transactions on Cybernetics*, vol. 52, no. 9, pp. 9377–9390, 2021.

[17] E. Yu, Y. Song, G. Zhang, and J. Lu, "Learn-to-adapt: Concept drift adaptation for hybrid multiple streams," *Neurocomputing*, vol. 496, pp. 121–130, 2022.

[18] H. Gâlmeanu and R. Andonie, "Weighted incremental–decremental support vector machines for concept drift with shifting window," *Neural Networks*, vol. 152, pp. 528–541, 2022.

[19] G. Collell, D. Prelec, and K. R. Patil, "A simple plug-in bagging ensemble based on threshold-moving for classifying binary and multiclass imbalanced data," *Neurocomputing*, vol. 275, pp. 330–340, 2018.

[20] A. Cano and B. Krawczyk, "Kappa updated ensemble for drifting data stream mining," *Machine Learning*, vol. 109, no. 1, pp. 175–218, 2020.

[21] L. Zhao, Y. Zhang, Y. Ji, A. Zeng, F. Gu, and X. Luo, "Heterogeneous drift learning: classification of mix-attribute data with concept drifts," in *2022 IEEE 9th international conference on data science and advanced analytics (DSAA)*. IEEE, 2022, pp. 1–10.

[22] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." *Psychological review*, vol. 102, no. 3, p. 419, 1995.

[23] A. Maracani, U. Michieli, M. Toldo, and P. Zanuttigh, "Recall: Replay-based continual learning in semantic segmentation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 7026–7035.

[24] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[25] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Mirage: Mobile-app traffic capture and ground-truth creation," in *2019 4th International conference on computing, communications and security (ICCCS)*. IEEE, 2019, pp. 1–8.