PAPER

A Sequential Approach to Detect Drifts and Retrain Neural Networks on Resource-Limited Edge Devices*

Kazuki SUNAGA†a), **Takeya YAMADA**†b), ***Nonmembers*, *and* Hiroki MATSUTANI**†c), ***Member***

SUMMARY A practical issue of edge AI systems is that data distributions of trained dataset and deployed environment may differ due to noise and environmental changes over time. Such a phenomenon is known as a concept drift, and this gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, retraining of neural network models triggered by concept drift detection is a practical approach. However, since available compute resources are strictly limited in edge devices, in this paper we propose a fully sequential concept drift detection method in cooperation with an on-device sequential learning technique of neural networks. In this case, both the neural network retraining and the proposed concept drift detection are done only by sequential computation to reduce computation cost and memory utilization. We use three datasets for experiments and compare the proposed approach with existing batch-based detection methods. It is also compared with a DNN-based approach without concept drift detection. The evaluation results of the proposed approach show that the proposed method is capable of detecting each of four concept drift types. The results also show that, while the accuracy is decreased by up to 0.9% compared to the existing batch-based detection methods, it decreases the memory size by 88.9%–96.4% and the execution time by 45.0%–87.6%. As a result, the combination of the neural network retraining and the proposed concept drift detection method is demonstrated on Raspberry Pi Pico that has 264 kB memory.

key words: *edge AI, concept drift, on-device learning, OS-ELM*

1. Introduction

With the rapid spread of AI (Artificial Intelligence) and IoT (Internet-of-Things) technologies, the number of IoT devices connected to the Internet continues to grow significantly. In cloud-based AI systems, IoT devices typically collect data at deployed edge environments and send the data to datacenters via the Internet. In this case, IoT devices focus on the data collection, and cloud servers are in charge of big data analysis and sophisticated machine learning tasks using plenty of compute resources. In addition, edge intelligence [1] in which some machine learning tasks such as prediction are performed in the edge side is also becoming popular since performance and efficiency of edge devices have been improved significantly. Although conventional edge AI systems focus on prediction tasks, recently an on-device learning approach of neural networks is proposed for resource-limited IoT devices [2].

However, there are some limitations on such on-device learning approaches. First, there is a limitation on computation power and cost. Since they are often battery-powered, low-power consumption is required. In addition, deployed environments around the edge devices may change over time. That is, data distribution observed by the edge devices may shift as time goes by. For example, data distributions of trained dataset and deployed environment may differ due to noise and environmental changes. This gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, a concept drift detection is a well-known approach [3].

Since edge devices are resource-limited, in this paper we propose a lightweight concept drift detection method for resource-limited edge devices. The contributions of this paper are as follows.

- We propose a fully sequential concept drift detection method to be combined with the on-device sequential learning approach of neural networks.
- Since both the neural network retraining and the proposed concept drift detection are done only by sequential computation, we demonstrate that the combined approach is implemented on Raspberry Pi Pico that has 264 kB memory.

The proposed approach is compared to existing concept drift detection methods in terms of accuracy using practical datasets. It is compared with a DNN-based approach without concept drift detection. It is also evaluated in terms of execution time and memory utilization on edge devices.

The rest of this paper is organized as follows. Section 2 overviews concept drift detection methods. Section 3 explains the proposed detection method. Section 4 describes the experimental setup including the datasets, counterparts, and platforms. Section 5 shows the evaluation results in terms of the accuracy, execution time, and memory utilization. Section 6 concludes this paper.

2. Background and Related Work

2.1 Concept Drift Types

A concept drift [5] is known as a phenomenon where statistical properties of target data change over time. It is sometimes

Manuscript received June 22, 2023.
Manuscript revised November 13, 2023.
Manuscript publicized February 9, 2024.

†The authors are with Graduate School of Science and Technology, Keio University, Yokohama-shi, 223–8522 Japan.

*This paper is an extended version of our previous work presented at an international workshop [4] by updating the evaluation results and adding comparisons to a DNN-based approach.

a) E-mail: sunaga@arc.ics.keio.ac.jp
b) E-mail: takeya@arc.ics.keio.ac.jp
c) E-mail: matutani@arc.ics.keio.ac.jp

DOI: 10.1587/transinf.2023EDP7123

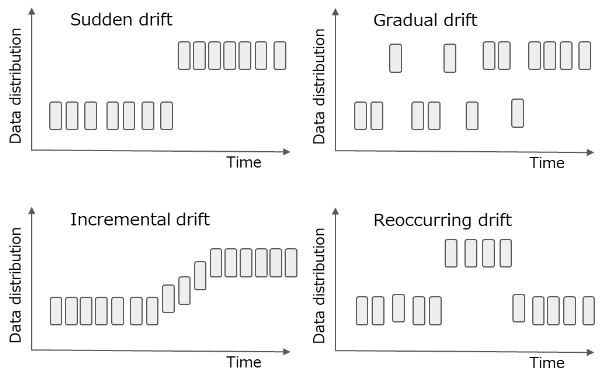

Fig. 1 Four concept drift types [5]

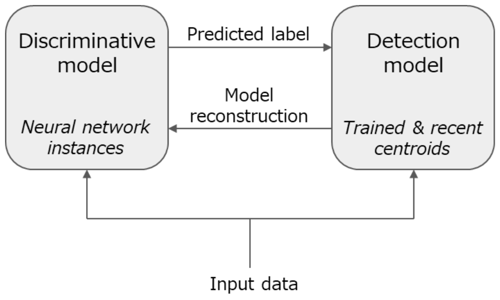

Fig. 2 Overview of proposed detection method

caused by changes on hidden variables which cannot be observed directly. There are various types of concept drifts, and representative ones [5] are illustrated in Fig. 1. In the figure, the vertical and horizontal axes represent data distribution and elapsed time, respectively. The sudden drift is a concept drift in which a data distribution changes suddenly. In the sudden drift, an old data distribution before the concept drift does not appear after the concept drift. The gradual drift is a concept drift in which an old data distribution is gradually replaced with a new data distribution. Both the old and new distributions appear during the concept drift. In the incremental drift, the data distribution is incrementally shifted from an old distribution to a new distribution during the concept drift. In the reoccurring drift, after the data distribution has been changed to a new one, the old data distribution reoccurs.

2.2 Concept Drift Countermeasures

There are various approaches to address the concept drifts, and they can be classified into active approaches and passive approaches [6]. In this paper, a machine learning model that solves classification or regression tasks is called a "discriminative model", and a model that detects concept drifts is called a "detection model". An example of their relationship is illustrated in Fig. 2.

2.2.1 Passive Approach

In the passive approach, a discriminative model is retrained whenever a new data arrives. Since the discriminative model can be always trained with the latest data, its accuracy tends to be high. It does not use any detection model. However, it requires computation resource and memory to retrain a discriminative model. This may limit its application to resource-limited edge devices. To enable retraining of neural networks on resource-limited edge devices, OS-ELM [7] is used as an online sequential learning algorithm of neural networks that have a single hidden layer in ONLAD [8]. Since it sequentially updates weight parameters of neural networks when new training samples come, the memory utilization is quite small compared to batch training algorithms. ONLAD is classified as a passive approach. Specifically, OS-ELM is combined with a forgetting mechanism to forget old collected data and follow a concept changes quickly. The training batch size is fixed to one so that pseudo inverse operation of matrixes can be eliminated.

2.2.2 Active Approach

In the active approach, a machine learning model is retrained only when a concept drift is detected. It thus requires a detection model in addition to a discriminative model. Since it may be difficult to address all the concept drift types introduced in Sect. 2.1 at the same time for any applications, existing active approaches often focus on some specific concept drift types [9]. There are several detection models and they can be broadly classified into two methods below and their ensemble [5].

The first detection method is an error-rate based drift detection method. This method monitors prediction errors of a discriminative model using labeled teacher data, and it detects a concept drift when the error-rate exceeds a threshold value. DDM [10] and ADWIN [11] are the error-rate based drift detection methods. DDM (Drift Detection Method) uses two threshold levels: warning level and drift level. When an error-rate reaches the warning level, it starts a retraining of a discriminative model. When the error-rate reaches the drift level, the retrained discriminative model replaces the old model. The number of samples required to judge concept drifts, which is called window size, is fixed at DDM. In ADWIN (Adaptive Windowing), the window size is adaptively adjusted based on test statistics. Since these approaches need a labeled teacher dataset to detect a concept drift, they are not suited to resource-limited edge devices with a limited memory capacity.

The second detection method is a distribution-based drift detection. Quant Tree [12] and SPLL [13] are the distribution-based drift detection methods. Quant Tree detects concept drifts by using a histogram. Although the size of histogram increases as the number of features (the number of dimensions) increases in typical histogram-based detection methods, it can be fixed in Quant Tree. Also, the test statistics to detect concept drifts does not depend on training and test datasets. SPLL detects concept drifts by using semi-parametric log-likelihood. Input data samples are clustered by using k-means method, and then the resultant clusters are modeled assuming GMM (Gaussian Mixture Model) to detect concept drifts. The distribution-based drift detection methods, such as Quant Tree and SPLL, often process a batch of data samples to detect concept drifts. Thus, they are also not suited to resource-limited edge devices with a limited memory capacity. In this paper, we will propose a concept drift detection method of neural networks both of which require only sequential computation to reduce computation cost and memory utilization.

3. Proposed Detection Method

Figure 2 illustrates an overview of the proposed concept drift detection method. Specifically, the proposed lightweight detection method is combined with the on-device sequential learning approach of neural networks [2] as a discriminative model. In this section, the discriminative model assumed in this paper is briefly illustrated first. Then the proposed detection method is explained.

3.1 Discriminative Model

Assume data can be classified into one of multiple labels. In the discriminative model, the same number of OS-ELM based neural networks (called "instances") as the number of labels in the training dataset are used. For each label in the training dataset, a discriminative model instance is trained with the data belonging to the label. Each discriminative model instance forms an autoencoder [14] for unsupervised anomaly detection. That is, the numbers of input and output layer nodes of the discriminative model instances are the same, and each instance is trained so that its output can reconstruct a given input data with a smaller number of hidden nodes.

In the test phase, a reconstruction error (anomaly score) is calculated by comparing the input and output data in each model instance, and the smallest anomaly score among all the instances is used as the final prediction result (see lines 6 and 7 in Algorithm 1). For the sequential training, a single model instance that outputs the smallest anomaly score (i.e., the "closest" instance) trains the input data sequentially. We employ this architecture for anomaly detection on multiple normal patterns as in [2].

3.2 Concept Drift Detection

At first, the proposed concept drift detection method calculates a centroid of trained data for each label. It records the same number of trained centroids as the number of labels. Then it sequentially updates the centroid with recent test data for each label whenever it predicts. It maintains test centroids in addition to the trained centroids. A drift rate is calculated based on a sum of the distance between the trained centroid and corresponding test centroid for each label (see line 14 in Algorithm 1). Then a concept drift is detected when the drift rate exceeds a pre-determined threshold value.

The proposed method is illustrated below. In the initial

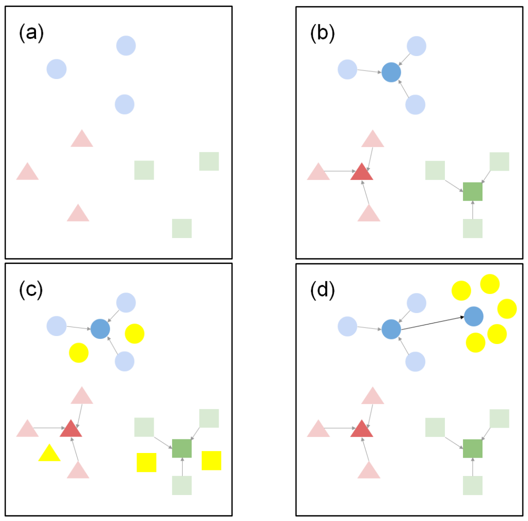

Fig. 3 Overview of proposed detection algorithm: (a) initial samples, (b) trained centroids, (c) test centroids before concept drift, and (d) those after concept drift

training phase, a discriminative model is trained with initial samples. Assume there are initial samples which are labeled as one of three different colors as shown in Fig. 3 (a). In the case of unsupervised learning, it is assumed that these initial samples can be labeled with a clustering algorithm such as k-means. A centroid of the initial samples is calculated for each label, as shown in Fig. 3 (b). In this figure, the centroids are represented as deeper-colored points. They are referred as "trained centroids". The proposed method thus records the same number of trained centroids as the number of labels in the initial training phase.

In the prediction phase, the discriminative model predicts a label for each test sample. The centroids are sequentially updated based on each test sample and its predicted label. They are referred as "test centroids". When the centroids are calculated, it is possible to assign a higher weight to a newer sample (a lower weight to an older sample) so that they can represent "recent" test centroids. Assume a new test sample comes and it is labeled as "blue" by the discriminative model. The recent test centroid of "blue" label is then sequentially updated. The data distribution is relatively stable before a concept drift occurs. We can thus expect that distances between the trained centroids and the recent test centroids are small as shown in Fig. 3 (c).

Next, let us illustrate a case when a concept drift happens. Assume the data distribution is changed and new test samples appear as shown in yellow circles in Fig. 3 (d). If these new test samples are labeled as "blue" by the discriminative model, the recent test centroid of "blue" label is moved to near the yellow circles in Fig. 3 (d). The distance between the test centroid and the trained centroid increases due to the new data distribution. A drift rate is calculated based on a sum of these distances, and a concept drift is detected when the drift rate exceeds a pre-determined threshold value. Please note that in the proposed method, the distances can be sequentially updated when it predicts. It thus requires

Algorithm 1 Concept drift detection

```
Require: discriminative model $model$, test dataset $testdata$, number of classes $C$, number of dimensions $D$, recent coordinates $cor$, trained coordinates $train\_cor$, numbers of samples in classes $num$, window size $W$, error threshold $\theta_{error}$, drift threshold $\theta_{drift}$
 1: function Main
 2:   $drift \leftarrow$ False
 3:   $check \leftarrow$ False
 4:   for $data \in testdata$ do:
 5:     if $drift =$ False & $check =$ False then
 6:       $c \leftarrow \mathrm{argmin}_{i \in C} model[i].\mathrm{predict}(data)$
 7:       $error \leftarrow model[c].\mathrm{predict}(data)$
 8:       if $error \geq \theta_{error}$ then
 9:         $check \leftarrow$ True
10:         $win \leftarrow 0$
11:     if $check =$ True & $win < W$ then
12:       $cor[c] \leftarrow \dfrac{cor[c] \times num[c] + data}{num[c] + 1}$
13:       $num[c] \leftarrow num[c] + 1$
14:       $dist \leftarrow \Sigma_{i \in C} \Sigma_{j \in D} |cor[i][j] - train\_cor[i][j]|$
15:       $win \leftarrow win + 1$
16:       if $win = W$ then
17:         if $dist \geq \theta_{drift}$ then
18:           $drift \leftarrow$ True
19:         $check \leftarrow$ False
20:     if $drift =$ True then
21:       $drift \leftarrow$ Reconstruct_Model$(data)$
22:   end for
```

much less memory than batch-based concept drift detection methods introduced in Sect. 2.2.2.

Algorithm 1 shows the proposed detection method mentioned above. The inputs to the proposed algorithm are a discriminative model, a test dataset, the number of class labels, the number of dimensions of the data, trained centroids, the number of samples in each label, a window size, and error/drift threshold values. Variable $drift$ indicates whether a concept drift is occurring now or not. In line 2, it is initialized to False. Variable $check$ indicates whether a concept drift needs to be checked or not. In line 3, it is initialized to False. In line 6, assuming the discriminative model with multiple instances introduced in Sect. 3.1, a class label predicted by the model is set to variable c. In line 7, an anomaly score predicted by the discriminative model is set to variable $error$. In lines 8 and 9, if the anomaly score exceeds a given threshold value θ_{error}, $check$ is set to True so that a concept drift will be checked. θ_{error} is a tuning parameter. In line 10, a window counter win is initialized to 0.

If $check$ is True and the window counter is less than a pre-determined window size W, recent test centroid of the predicted label c is sequentially updated based on an incoming $data$. In line 12, $cor[c]$ is the test centroid of label c. Then a sum of the distance between the trained centroid and the recent test centroid for each label is updated. In line 14, $dist$ is the sum of the distances. If the window counter win reaches a given window size W, a concept drift is checked by comparing the distance $dist$ and a threshold value θ_{drift} in line 17. The threshold θ_{drift} can be determined as explained in Sect. 3.4. If the distance exceeds the threshold, $drift$ is set to True in line 18. In this case, the discriminative model is retrained in line 21. Reconstruct_Model() function is described in Sect. 3.3.

Algorithm 2 Discriminative model reconstruction

```
Require: discriminative model $model$, number of classes $C$, number of dimensions $D$, coordinates $cor$, numbers of samples in classes $num$, number to search initial coordinates $N_{search}$, number to update coordinates $N_{update}$, number to finish reconstruction $N$
 1: function Reconstruct_Model$(data)$
 2:   $count \leftarrow count + 1$
 3:   if $count < N_{search}$ then
 4:     Init_Coord$(data)$
 5:   else if $count < N_{update}$ then
 6:     Update_Coord$(data)$
 7:   else if $count < N/2$ then
 8:     $label \leftarrow \mathrm{argmin}_{c \in C} |cor[c] - data|$
 9:     $model[label].\mathrm{training}(data)$
10:   else if $count < N$ then
11:     $label \leftarrow \mathrm{argmin}_{c \in C} model[c].\mathrm{predict}(data)$
12:     $model[label].\mathrm{training}(data)$
13:   if $count = N$ then
14:     $count \leftarrow 0$
15:     return False
16:   else
17:     return True
```

Algorithm 3 Label coordinates initialization

```
Require: number of classes $C$, coordinates $cor$, number of dimensions $D$
 1: function Init_Coord$(data)$
 2:   $label \leftarrow -1$
 3:   $min \leftarrow \sum_{j=0}^{C-1} \sum_{k=j+1}^{C} \sum_{dim=0}^{D} |cor[j][dim] - cor[k][dim]|$
 4:   for $c \in C$ do:
 5:     $tmp \leftarrow cor[c]$
 6:     $cor[c] \leftarrow data$
 7:     $dist \leftarrow \sum_{j=0}^{C-1} \sum_{k=j+1}^{C} \sum_{dim=0}^{D} |cor[j][dim] - cor[k][dim]|$
 8:     $cor[c] \leftarrow tmp$
 9:     if $min < dist$ then
10:       $label \leftarrow c$
11:       $min \leftarrow dist$
12:   end for
13:   if $label \neq -1$ then
14:     $cor[label] \leftarrow data$
```

3.3 Model Reconstruction

The discriminative model reconstruction is divided into four parts. The model reconstruction flow is shown in Algorithm 2. In the first part, an initial coordinate is selected for each label by Init_Coord() function in line 4. In Init_Coord() function, given the number of labels C, C initial samples are selected as initial coordinates of C labels. Specifically, these initial coordinates are selected so that a sum of all the distances between these coordinates is maximized. This part is inspired from an idea of k-means++ algorithm [15] that spreads out initial cluster centroids for a better clustering. The detail implementation of Init_Coord() function is shown in Algorithm 3.

In the second part, centroids of C labels are updated based on an incoming data by Update_Coord() function in line 6. Specifically, a label that minimizes the distance between the incoming data and centroid of the label is selected.

Algorithm 4 Label coordinates update

```
Require: number of classes $C$, coordinates $cor$, numbers of samples in
    classes $num$
1: function Update_Coord($data$)
2:     $label \leftarrow \mathrm{argmin}_{c \in C} |cor[c] - data|$
3:     $cor[label] \leftarrow \frac{cor[label] \times num[label] + data}{num[label] + 1}$
4:     $num[label] \leftarrow num[label] + 1$
```

Based on the selected label, its centroid is then sequentially updated. This part is very similar to a sequential k-means algorithm. The detail implementation of Update_Coord() function is shown in Algorithm 4. Please note that since there is a possibility that initial coordinates selected by Init_Coord() are outliers, the centroids are further refined by Update_Coord() function using more samples.

In the third part, the discriminative model is retrained. In lines 8 and 9, a label that minimizes the distance between the incoming data and centroid of the label is selected, and then the discriminative model is sequentially updated by the incoming data and the selected label.

The fourth part is similar to the third part, but as shown in lines 11 and 12, a label is predicted by the discriminative model being retrained, and then the discriminative model is sequentially updated by the incoming data and the predicted label.

3.4 Threshold

The threshold value θ_{drift} is used to detect a concept drift in line 17 of Algorithm 1. For each trained sample, a distance between the sample and the centroid of its predicted label is calculated and stored in *dist* array. In this paper, θ_{drift} is calculated based on the mean and standard deviation of *dist* array, as shown below.

$$\begin{aligned} \mu &= \frac{1}{N}\Sigma_{i \in N} dist[i] \\ \theta_{drift} &= \mu + z\sqrt{\frac{1}{N}\Sigma_{i \in N}(dist[i] - \mu)^2}, \end{aligned} \tag{1}$$

where N is the number of trained samples and $dist[i]$ is a distance between the i-th sample and the centroid of a cluster the i-th sample belongs to. z is a tuning parameter and we simply assume $z = 1$ in this paper.

4. Evaluation Setup

This section describes datasets, counterparts of the proposed method, and platforms for the evaluations.

4.1 Datasets

Three datasets used in the evaluations are described below.

4.1.1 Synthetic Dataset

First, we evaluate the proposed method using sine and cosine waves to demonstrate its capability to detect each type of concept drifts shown in Fig. 1. Each data has 100 features, and a random noise of $[-0.1, 0.1]$ is added to each feature. We use $\sin\theta$, $\cos\theta$, and $-\sin\theta$ as a training dataset. The following four patterns are used as test datasets.

1. The first dataset focuses on a sudden drift. We use $\sin\theta$, $\cos\theta$, and $-\sin\theta$ as test data before a concept drift, and those with a phase shift of 135 degrees for each as test data after the concept drift. The concept drift occurs at the 1998th data point.
2. The second dataset focuses on a gradual drift. The dataset used is almost the same as the first dataset, but in the second dataset, both the patterns are mixed between the 1998th data point and the 2808th data point so that a gradual drift can be reproduced.
3. The third dataset focuses on an incremental drift. The dataset used is almost the same as the first dataset, but in the third dataset, the phase gradually shifts in a continuous manner from the 1998th data point to the 2808th data point so that an incremental drift can be reproduced.
4. The fourth dataset focuses on a reoccurring drift. The dataset used is almost the same as the first dataset, but in the fourth dataset, data after the concept drift appear only between the 1998th data point and the 2808th data point. Those before the concept drift reoccur after the 2808th data point so that reoccurring drift can be reproduced.

4.1.2 NSL-KDD Dataset

NSL-KDD [16] is a famous dataset which can be used to evaluate network intrusion detection methods. As data distribution of the dataset shifts from the training data to the test data, it can also be used for evaluating concept drift detection methods [3]. This change in data distribution is attributed to changes in attack patterns. The original dataset contains a lot of samples with 23 labels. In this paper, we use selected samples labeled with "normal" and "satan" for training, and "normal" and "ipsweep" for testing. We further select 1523 and 13709 samples for the initial training and test, respectively. A concept drift occurs at the 6110th data point.

4.1.3 Cooling Fan Dataset

The cooling fan dataset [17] contains vibration patterns of various cooling fans measured by an industrial accelerometer PCB M607A11. These vibration patterns were measured at a silent environment and a noisy environment near a ventilation fan. The vibration pattern is represented as a frequency spectrum ranging from 1 Hz to 511 Hz. Thus, the number of features is 511 in the cooling fan dataset. The data of four vibration patterns (0 rpm, 1500 rpm, 2000 rpm, and 2500 rpm) are used. Specifically, those observed in a silent environment are used as a training dataset, and those

Table 1 Detector and discriminative model parameter settings in synthetic dataset

	Proposed
Train methods	OS-ELM (sequential)
Layers & nodes	$\{100, 22, 100\} \times$ instances
Hyperparameters of discriminative model	instances : 3
Hyperparameters of detector	W : 100

Table 2 Detector and discriminative model parameter settings in NSL-KDD dataset

	Proposed	Quant Tree	SPLL	Baseline	ONLAD	DNN
Train methods	OS-ELM (sequential)				OS-ELM (w/ forgetting)	Backprop & SGD (mini-batch)
Layers & nodes	$\{37, 22, 37\} \times$ instances				$\{37, 22, 37\} \times$ instances	$\{37, 30, 22, 10, 2\}$
Hyperparameters of discriminative model	instances : 2, $N = 400$ $N_{search} = N/8, N_{update} = N/5$			instances : 2	instances : 2 forgetting late : 0.99	batch : 16, epoch : 10 learning rate : 0.005
Drift detection	Yes	Yes	Yes	No	No	No
Hyperparameters of detector	W : 100	batch : 400	batch : 400 histograms : 32	-	-	-

Table 3 Detector and discriminative model parameter settings in fan dataset

	Proposed	Quant Tree	SPLL	Baseline	ONLAD	DNN
Train methods	OS-ELM (sequential)				OS-ELM (w/ forgetting)	Backprop & SGD (mini-batch)
Layers & nodes	$\{511, 22, 511\} \times$ instances				$\{511, 22, 511\} \times$ instances	$\{511, 256, 96, 16, 4\}$
Hyperparameters of discriminative model	instances : 4, $N = 180$ $N_{search} = N/8, N_{update} = N/5$			instances : 4	instances : 4 forgetting late : 0.99	batch : 4, epoch : 10 learning rate : 0.005
Drift detection	Yes	Yes	Yes	No	No	No
Hyperparameters of detector	W : 20	batch : 235	batch : 235 histograms : 16	-	-	-

observed in a noisy environment are used as a testing dataset.

4.2 Evaluated Methods

In this paper, the following six combinations are evaluated and compared as concept drift countermeasures.

1. Detector: the proposed method, Discriminative model: OS-ELM
2. Detector: none (no concept drift detection), Discriminative model: OS-ELM
3. Detector: Quant Tree, Discriminative model: OS-ELM
4. Detector: SPLL, Discriminative model: OS-ELM
5. Detector: none, Discriminative model: ONLAD (OS-ELM with a forgetting mechanism)
6. Detector: none, Discriminative model: DNN (3 hidden layers)

The first method is our proposal, and the second method is a baseline without concept drift detection. The first, third, and fourth methods are classified as the active detection approach, while the fifth method is the passive approach. The sixth method is a DNN-based approach without concept drift detection.

Except for the DNN-based approach, OS-ELM is used in the discriminative model. More specifically, the same number of OS-ELM based autoencoder instances as the number of labels are used, as mentioned in Sect. 3.1. The reason for using OS-ELM in the discriminative model is that it can be retrained on resource-limited edge devices [2]. Another reason is that we want to compare the proposed method with ONLAD that uses OS-ELM and a lightweight forgetting mechanism. By adding the DNN-based approach, we also compare OS-ELM with a DNN.

The synthetic dataset is used to verify the operation of the proposed method. Table 1 shows the hyperparameters used for the synthetic dataset. The numbers of input and output layer nodes of the model are 100 and that of the hidden layer nodes is 22. Table 2 shows the hyperparameters used for the NSL-KDD dataset. In the OS-ELM based discriminative model, the numbers of input and output layer nodes are 37 and that of the hidden layer nodes is 22. In Quant Tree, the batch size is 400 and the number of histograms is 32. In SPLL, the batch size is 400. In ONLAD, the numbers of input, hidden, and output layer nodes are the same as those of the OS-ELM model. The forgetting rate is 0.99. The DNN model has three hidden layers and are trained by a back-propagation algorithm with a SGD optimizer. Also, Table 3 shows the hyperparameters used for the cooling fan dataset. The six methods are set up accordingly.

4.3 Evaluation Platforms

The above-mentioned six methods are running on Raspberry Pi 4 Model B [18]. In addition, due the memory size constraints, only the proposed method is demonstrated on Raspberry Pi Pico [19]. These methods are evaluated in terms of memory utilization in Raspberry Pi 4 Model B. The execution time breakdown of the proposed method is further analyzed in Raspberry Pi Pico. Table 4 shows the specifications of Raspberry Pi 4 Model B and Raspberry Pi Pico.

Table 4 Specifications of Raspberry Pi 4 Model B and Raspberry Pi Pico

	Raspberry Pi 4 Model B	Raspberry Pi Pico
OS	Raspberry Pi OS	-
CPU	ARM Cortex-A72, 1.5GHz	ARM Cortex-M0+, 133MHz
RAM	4GB	264kB

5. Evaluation Results

This section shows evaluation results of the six methods listed in Sect. 4.2 in terms of the accuracy, delay to detect concept drifts, memory utilization, and execution time.

5.1 Synthetic Dataset

First, the proposed concept drift detection method is evaluated for the four concept drift types. Figure 4 (a), (b), (c), and (d) are the evaluation results for sudden drift, gradual drift, incremental drift, and reoccurring drift, respectively. The X-axis indicates the number of data, and Y-axis is the error between input and output in each graph. The first vertical bar at the 1998th data point in Fig. 4 represents an occurrence of concept drift, and the second vertical bar at the 2808th data point represents the end of concept drift in the cases of (b), (c), and (d). Please note that the second bar in (d) indicates the data distribution before the concept drift occurred has reoccurred. It can be seen from each figure that loss is sharply increased when a concept drift occurs. In the sudden drift, it is observed that loss is decreased by learning after the drift detection. In the gradual drift and incremental drift, there may appear "intermediate concept" [20] during the transformation, so if the intermediate concept is learned, the loss may be high even after the end of drift point. However, in the proposed method, a retraining flag is set based on the loss. Therefore, it can be observed that the proposed method is able to learn the data after the drift even if an intermediate concept is learned. In the reoccurring drift, the emphasis is on how to find the best matched historical concept in a short time. Since, the proposed method uses centroids of the data, the best matched one can be selected simply by storing these centroids and comparing them with centroids of the most recent data.

5.2 NSL-KDD Dataset

The six methods are evaluated with NSL-KDD dataset. Figure 5 shows evaluation results in terms of the accuracy of the discriminative model. Table 5 summarizes the accuracy and the delay to detect a concept drift. The delay means the number of samples needed to detect a concept drift after the concept drift actually happens. A vertical bar at the 6110th data point in Fig. 5 shows a concept drift.

First, the results show that the parameter tuning of a forgetting rate of ONLAD is difficult. Since a concept drift happens at the 6110th data point, it is expected that accuracy of the ONLAD model should be constantly high before the concept drift. However, the results show that the accuracy

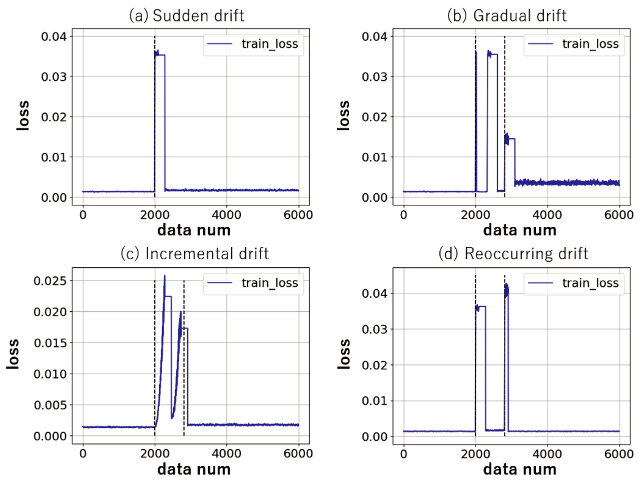

Fig. 4 Loss changes on synthetic dataset

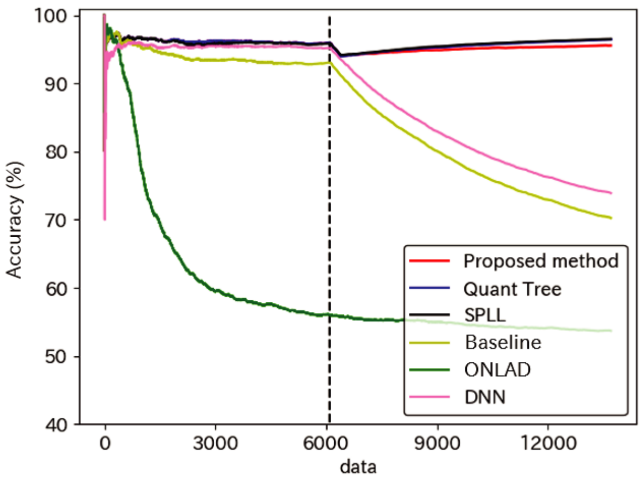

Fig. 5 Accuracy changes on NSL-KDD dataset

Table 5 Accuracy (%) and delay (number of samples) for detecting concept drift on NSL-KDD dataset

	Accuracy	Delay
Quant Tree	96.4	290
SPLL	96.5	290
Baseline (no concept drift detection)	70.2	-
ONLAD	53.7	-
DNN	73.9	-
Proposed method	95.6	193

of the ONLAD model gradually decreases even before the concept drift happens.

The results also show that the proposed method can detect the concept drift as well as the batch-based Quant Tree and SPLL methods in the NSL-KDD dataset. After the concept drift is detected, the accuracy of the proposed method becomes high compared to the baseline method that does not detect concept drifts. As a result, the proposed method outperforms the baseline method without concept drift detection, ONLAD with a forgetting mechanism, and DNN by 25.4%, 41.9%, and 21.7%, respectively, while the

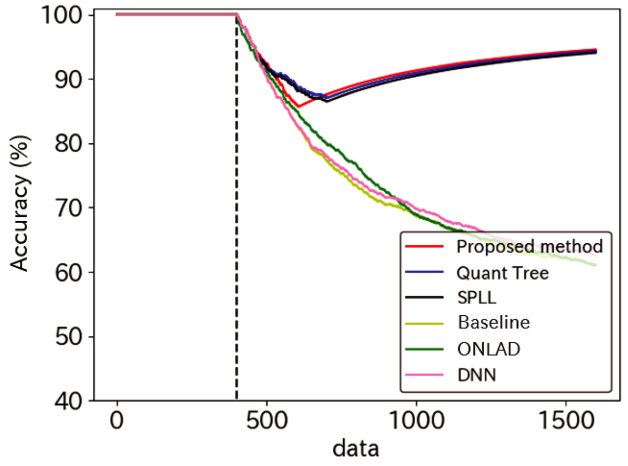

Fig. 6 Accuracy changes on cooling fan dataset

Table 6 Accuracy (%) and delay (number of samples) for detecting concept drift on cooling fan dataset

	Accuracy	Delay
Quant Tree	94.3	70
SPLL	94.1	70
Baseline (no concept drift detection)	61.0	-
ONLAD	61.1	-
DNN	62.5	-
Proposed method	94.6	25

accuracy is decreased by up to 0.9% and 0.8% compared to the batch-based SPLL and Quant Tree methods, respectively. Please note that the proposed method needed less samples to detect the concept drift compared to the batch-based Quant Tree and SPLL methods.

5.3 Cooling Fan Dataset

Next, we show the results of the evaluation on the cooling fan dataset in Fig. 6. Table 6 summarizes the accuracy and the delay to detect a concept drift. A vertical bar at the 400th data point in Fig. 6 shows a concept drift as well as Fig. 5.

In the cooling fan dataset, the accuracy becomes high in the methods that detect the concept drifts. As a result, the proposed method outperforms the baseline method without concept drift detection, ONLAD with a forgetting mechanism, and DNN by 33.6%, 33.5%, and 32.1%, respectively, and the accuracy is increased by up to 0.5% and 0.3% compared to the batch-based SPLL and Quant Tree methods, respectively. It also needed less samples to detect the concept drift compared to the batch-based Quant Tree and SPLL methods.

5.4 Memory Utilization

Table 7 shows the evaluation results in terms of the memory utilization on Raspberry Pi 4 Model B. The cooling fan dataset is used for the memory size evaluation; in this case, the batch size of the Quant Tree and SPLL methods is 235 while it is one in the proposed method.

Table 7 Memory utilization (kB)

	Memory size
Quant Tree	619
SPLL	1933
Proposed method	69

The results show that the proposed method uses much less memory size compared to the batch-based Quant Tree and SPLL methods. Specifically, the proposed method decreases the memory utilization by up to 96.4% and 88.9% compared to SPLL and Quant Tree, respectively. This is because in the batch-based concept drift detection methods, data samples are stored in the device memory to detect concept drifts, while the proposed method processes data samples one by one and detects concept drifts sequentially; thus, the proposed method does not store past samples in the device memory.

Please note that since RAM size of Raspberry Pi Pico is only 264 kB as shown in Table 4, the batch-based Quant Tree and SPLL methods cannot operate on Raspberry Pi Pico. It is known that Raspberry Pi Pico is available from only $4 [19]. The memory size reduction by the proposed approach is beneficial in terms of the hardware cost, because our proposed system can be implemented on this low-cost device. In addition, the memory size reduction by the proposed approach can extend the applicable range of microcontrollers. For example, ultra-low power products of STMicroelectronics STM32 microcontrollers include several series, such as STM32L0, STM32L4, STM32L4+, STM32L5, and STM32U5 [21]. Especially, in the cases of STM32L4, STM32L4+, and STM32L5 series, their SRAM sizes typically range from 40 kB to 640 kB. Thus, our approach can be implemented on a wider range of these microcontrollers compared to the counterparts. In Sect. 5.5, only the proposed method shows the execution time on Raspberry Pi Pico in addition to that on Raspberry Pi 4 Model B.

5.5 Execution Time

Tables 8 and 9 show evaluation results in terms of the execution times on Raspberry Pi 4 Model B and Raspberry Pi Pico, respectively. The same cooling fan dataset is used for this evaluation.

Table 8 shows the execution time to process the cooling fan dataset that contains 1600 samples in total. As shown, the execution time of the proposed method is much less than that of Quant Tree and SPLL. Specifically, the proposed method decreases the execution time by up to 87.6% and 45.0% compared to SPLL and Quant Tree, respectively. Since SPLL executes k-means clustering, the execution time of SPLL is increased compared to the others. In the case of the proposed approach, there is a possibility to meet a timing constraint with a lower operating frequency or less compute resources. Since a higher operating frequency and more compute resources tend to increase the power consumption and hardware cost, the proposed approach is beneficial in terms of the power consumption and miniaturization of tar-

Table 8 Execution time (sec) for 1600 samples on Raspberry Pi 4 Model B

	Execution time
Quant Tree	7.50
SPLL	33.35
Baseline (no concept drift detection)	2.61
Proposed method	4.13

Table 9 Execution time breakdown (msec) for 1 sample by proposed method on Raspberry Pi Pico

	Execution time
Label prediction	148.87
Distance computation	10.58
Model retraining without label prediction	25.42
Model retraining with label prediction	166.65
Label coordinates initialization	25.59
Label coordinates update	6.05

get devices. Although the proposed method increases the execution time by 58.2% compared to the baseline method without concept drift detection, it significantly improves the accuracy compared to the baseline as shown in Figs. 5 and 6. Thus, the proposed method is an attractive option when concept drifts are expected in target applications.

Table 9 further analyzes the execution time breakdown for a single sample by the proposed method on Raspberry Pi Pico. In the table, "Label prediction" and "Distance computation" are corresponding to lines 6 and 14 in Algorithm 1, respectively. "Model retraining without label prediction" is done in lines 8 and 9, while "Model retraining with label prediction" is done in lines 11 and 12 in Algorithm 2. "Label coordinates initialization" is Init_Coord() function in Algorithm 3, and "Label coordinates update" is Update_Coord() function in Algorithm 4. The results show that the additional computation time for the concept drift detection is less than the label prediction time of the discriminative model. Please note that the latency is within a few hundred milliseconds even in such a low-end edge device.

6. Conclusions

In edge AI systems, data distributions of trained dataset and deployed environment may differ due to noise and environmental changes over time. This gap degrades the performance of edge AI systems and may introduce system failures. To address this gap, retraining of neural network models triggered by concept drift detection is a practical approach. As practical concept drift detection, error-rate based detection methods and distribution-based detection methods have been used. However, since such a batch-based processing and use of labeled teacher dataset are not suited for resource-limited edge devices, in this paper we proposed a fully sequential concept drift detection method in cooperation with the on-device sequential learning technique of neural networks. Both the neural network retraining and the proposed concept drift detection are done only by sequential computation to reduce computation cost and memory utilization. The proposed approach was compared with existing concept drift detection methods and a DNN model without concept drift detection.

Evaluation results of the proposed approach showed that while the accuracy is decreased by up to 0.9% compared to existing batch-based detection methods, it decreases the memory size by 88.9%–96.4% and the execution time by 45.0%–87.6%. Thanks to these significant decreases on the memory size and computation cost, the combination of the neural network retraining and the proposed concept drift detection method was demonstrated on Raspberry Pi Pico that has 264 kB memory. A possible extension of this work is a combination of multiple detection models with different window sizes to address more complicated concept drift behaviors since it is expected that the sizes of detection models are less dominant compared to the discriminative model. We are also planning to apply the proposed system to more practical applications, such as anomaly detection on rotating machines as in [22].

Acknowledgments

This work was partially supported by JST AIP Acceleration Research JPMJCR23U3, Japan.

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," Proc. IEEE, vol.107, no.8, pp.1738–1762, 2019.
[2] H. Matsutani, M. Tsukada, and M. Kondo, "On-Device Learning: A Neural Network Based Field-Trainable Edge AI," arXiv Preprint 2203.01077, 2022.
[3] L. Yang and A. Shami, "A Lightweight Concept Drift Detection and Adaptation Framework for IoT Data Streams," IEEE Internet of Things Magazine, vol.4, no.2, pp.96–101, 2021.
[4] T. Yamada and H. Matsutani, "A Lightweight Concept Drift Detection Method for On-Device Learning on Resource-Limited Edge Devices," Proc. International Parallel and Distributed Processing Symposium Workshops, pp.761–768, 2023.
[5] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," IEEE Trans. Knowl. Data Eng., vol.31, no.12, pp.2346–2363, 2019.
[6] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, "A novel concept drift detection method for incremental learning in nonstationary environments," IEEE Trans. Neural Netw. Learn. Syst., vol.31, no.1, pp.309–320, 2020.
[7] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, "A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks," IEEE Trans. Neural Netw., vol.17, no.6, pp.1411–1423, 2006.
[8] M. Tsukada, M. Kondo, and H. Matsutani, "A Neural Network-Based On-device Learning Anomaly Detector for Edge Devices," IEEE Trans. Comput., vol.69, no.7, pp.1027–1044, 2020.
[9] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," ACM Computing Surveys, vol.46, no.4, pp.1–37, 2014.
[10] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with Drift Detection," Proc. Brazilian Symposium on Artificial Intelligence, pp.286–295, 2004.
[11] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," Proc. SIAM International Conference on Data Mining, pp.443–448, 2007.
[12] G. Boracchi, D. Carrera, C. Cervellera, and D. Macciò, "Quant Tree:

Histograms for Change Detection in Multivariate Data Streams," Proc. International Conference on Machine Learning, pp.639–648, 2018.
[13] L.I. Kuncheva, "Change Detection in Streaming Multivariate Data Using Likelihood Detectors," IEEE Trans. Knowl. Data Eng., vol.25, no.5, pp.1175–1180, 2013.
[14] G.E. Hinton and R.R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," Science, vol.313, no.5786, pp.504–507, 2006.
[15] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," Proc. ACM-SIAM Symposium on Discrete Algorithms, pp.1027–1035, 2007.
[16] "NSL-KDD Dataset Research Candian Institute for Cybersecurity," https://www.unb.ca/cic/datasets/nsl.html.
[17] "A cooling fan dataset containing vibration patterns of various fans at silent and noisy environments," https://github.com/matutani/cooling-fan.
[18] "Raspberry Pi 4," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.
[19] "Raspberry Pi Pico," https://www.raspberrypi.com/products/raspberry-pi-pico/.
[20] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM computing surveys (CSUR), vol.46, no.4, pp.1–37, 2014.
[21] "STM32 32-bit Arm Cortex MCUs." https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.
[22] K. Sunaga, M. Kondo, and H. Matsutani, "Addressing Gap between Training Data and Deployed Environment by On-Device Learning," IEEE Micro, vol.43, no.6, pp.66–73, 2023.

Kazuki Sunaga received the B.E. degree from Keio University in 2023. He is currently a master course student in Keio University.

Takeya Yamada received the B.E. and M.E. degrees from Keio University in 2021 and 2023, respectively.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.