Adaptive Model Pooling for Online Deep Anomaly Detection from a Complex Evolving Data Stream

Susik Yoon
UIUC
Illinois, USA
susik@illinois.edu

Youngjun Lee
KAIST
Daejeon, Korea
youngjun.lee@kaist.ac.kr

Jae-Gil Lee*
KAIST
Daejeon, Korea
jaegil@kaist.ac.kr

Byung Suk Lee
University of Vermont
Vermont, USA
bslee@uvm.edu

ABSTRACT

Online anomaly detection from a data stream is critical for the safety and security of many applications but is facing severe challenges due to *complex and evolving* data streams from IoT devices and cloud-based infrastructures. Unfortunately, existing approaches fall too short for these challenges; online anomaly detection methods bear the burden of handling the complexity while offline deep anomaly detection methods suffer from the evolving data distribution. This paper presents a framework for online deep anomaly detection, ARCUS, which can be instantiated with any autoencoder-based deep anomaly detection methods. It handles the complex and evolving data streams using an *adaptive model pooling* approach with two novel techniques—*concept-driven inference* and *drift-aware model pool update*; the former detects anomalies with a combination of models most appropriate for the complexity, and the latter adapts the model pool dynamically to fit the evolving data streams. In comprehensive experiments with ten data sets which are both high-dimensional and concept-drifted, ARCUS improved the anomaly detection accuracy of the streaming variants of state-of-the-art autoencoder-based methods and that of the state-of-the-art streaming anomaly detection methods by up to 22% and 37%, respectively.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; • **Information systems** → **Data stream mining**.

KEYWORDS

Anomaly detection; Data stream; Autoencoder; Concept drift; Model pooling

ACM Reference Format:
Susik Yoon, Youngjun Lee, Jae-Gil Lee, and Byung Suk Lee. 2022. Adaptive Model Pooling for Online Deep Anomaly Detection from a Complex Evolving Data Stream. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539348

*Corresponding author.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539348

1 INTRODUCTION

1.1 Background and Motivation

An anomaly can be identified as a data point that has different characteristics from a majority of other data points, and means a novel observation, a certain failure, unexpected noise, etc. in a system of interest. Anomaly detection has numerous real-world applications such as fraud detection in financial institutions and abnormality detection in healthcare devices [24]

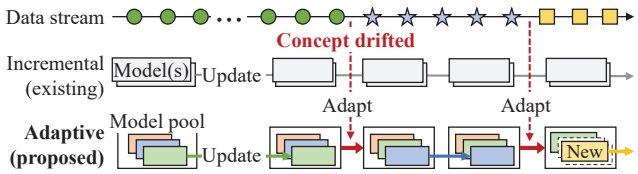
Today, a *complex* data stream is common from IoT devices and cloud-based infrastructures, where data items with hundreds of features of often unknown correlations and heterogeneous data types are continuously arriving. This complexity brings about an insurmountable challenge to anomaly detection, thereby necessitating significant preprocessing or heavy model training to cope with the complexity. The challenge is aggravated when the data stream is *evolving*, thereby making the preprocessing or the trained models outdated quickly. This phenomenon is often referred to as a *concept drift* [19], where properties of a target domain (i.e., concept) change arbitrarily. Such complex and evolving data streams are observed in various real-world situations [18, 29, 30]; e.g., gas sensor value streams to monitor gas leaks with varying gas concentrations.

This paper concerns the computational method to deal with complex evolving data streams. *Deep anomaly detection* [24] based on a deep neural network has proven to handle the complexity effectively, better than classical methods (e.g., k nearest neighbors [12]) [23, 25]. In particular, an autoencoder (AE) has been widely used, as it is appropriate for an *unsupervised setting* that is natural for anomaly detection with rare labels. Existing state-of-the-art AE-based methods [11, 14, 38], however, are designed for an offline setting and, thus, cannot effectively cope with evolving data streams. There are a few recurrent neural network (RNN)-based methods proposed for time series anomaly detection [9, 20, 28, 36]. However, they focus on learning temporal relationships inside local sequences and incrementally updating a single model, which are not suitable to handle arbitrarily evolving data streams.

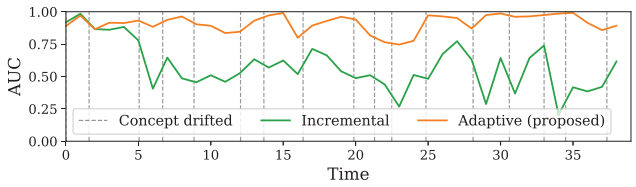
The goal of this paper, thus, is to provide a novel framework for *online deep anomaly detection* that adopts the existing deep anomaly detection methods adaptively in an online setting, thereby effectively dealing with the complexity and evolution challenges of a data stream.

1.2 Main Idea

Undoubtedly, using a pre-trained fixed model or creating a new model repeatedly would not work at all to handle a complex evolving data stream; they are either too ineffective or too inefficient. A common approach of the existing streaming anomaly detection methods is to build an initial model and incrementally update the



(a) Processing flow of online deep anomaly detection. The marks on the timeline represent the concepts of data points.



(b) Detection accuracy in AUC (using an AE-based model for an MNIST data set simulated with abrupt and recurrent concept drifts).

Figure 1: Comparison of incremental and adaptive approaches.

model over a data stream [1, 8, 34, 35] which can be easily applicable to deep anomaly detection methods. However, this *incremental approach* adapts a model to only the latest data points, regardless of how data streams evolve. With arbitrary concept drifts in data streams, the incremental approach could be ineffective as it needs some time to perfectly adapt to new concepts and inefficient as it quickly forgets the previous concept that may reoccur in the future.

The main idea in this paper is to use ***adaptive model pooling***, which manages multiple assorted models in both inference over and adaptation to a complex evolving data stream. Faced with concept drifts involving the indeterminate number of multiple patterns, a fixed model or models cannot handle all of them. A model pooling approach thus allows multiple models to work together adaptively to handle multiple and time-varying concept drifts, thereby achieving versatile anomaly detection performance for a varying number of unexpected concept drifts. Unlike the existing ensemble approaches [4, 8, 17] where the set of models is fixed in advance, the model pool membership is dynamically managed over time.

As illustrated in Figure 1a, the incremental approach (top) updates a fixed model or models without regard to concept drifts, whereas the adaptive approach (bottom) uses a model pool and adjusts it in response to concept drifts—by using the best combination of existing models or creating a new model. As shown in Figure 1b, this adaptive model pooling brings a clear advantage in anomaly detection accuracy when arbitrary concept drifts occur.

This paper realizes the adaptive model pooling with *autoencoder-based* deep anomaly detection models, which have shown state-of-the-art performances in unsupervised anomaly detection. Specifically, we propose a novel framework **ARCUS** (Adaptive framework foR online deep anomaly deteCtion Under a complex evolving data Stream), which employs two key techniques:

- *Concept-driven inference*: ARCUS calculates the anomaly scores of incoming data points using the best combination of models in the model pool. While optimizing individual models for different sets of data points, ARCUS estimates the reliability of each model against the given data points and decides how much each model contributes to the final anomaly score, to maximize the usability of the model pool in varying concepts.
- *Concept drift-aware update*: ARCUS continuously monitors the reliability of the model pool for the incoming data points. When the model pool is assessed inadequate for the latest data points, presumably due to a concept drift, ARCUS updates the model pool to incorporate a new model optimized for the new concept of data points while keeping the model pool as compact as possible. This update enables ARCUS to efficiently keep the best performance regardless of the patterns of concept drifts.

1.3 Highlights

- To the best of our knowledge, this is the first work that proposes an adaptive model pooling mechanism of deep anomaly detection models which addresses both the complexity and concept drift challenges of a data stream.
- With the model pooling in place, this paper proposes a novel framework **ARCUS** equipped with concept-driven inference and concept drift-aware update. ARCUS can be implemented with any existing AE-based anomaly detection model. For reproducibility, the source code of ARCUS is publicly available[1].
- Comprehensive experiments are conducted using ten data sets which are both high-dimensional and concept-drifted. ARCUS, when implemented using three state-of-the-art AE-based models, achieved up to 22% higher anomaly detection accuracy than their streaming variants and surpassed the best accuracy results of the existing online anomaly detection algorithms by up to 37%.

2 RELATED WORK

2.1 Deep Anomaly Detection

The recent rapid advancement of a deep neural network has led to many deep anomaly detection methods with various types of approaches (e.g., AE, RNN, or generative adversarial networks (GAN)) [24]. Among them, the autoencoder has been widely studied and achieved the state-of-the-art performances [11, 14, 38], thanks to its unsupervised but effective capability to remove noisy or anomalous information in the input. DAGMM [38] combines an AE with the Gaussian mixture model to detect anomalies by predicting the Gaussian mixture membership of a data instance in a low-dimensional representation obtained from an AE. RSRAE [14] added a linear transformation layer after a latent space of an AE to learn the hidden linear structure of the non-linearly embedded data points by an encoder. RAPP [11] investigates hidden activation values of layers in an AE in the same way as calculating reconstruction errors to verify the information loss during encoding and decoding processes. Although these methods have shown high anomaly detection accuracy for static data sets, none of them is geared for data streams. Besides, it is worth mentioning that the AE-based ensemble method [4] was proposed to use multiple AE models differentiated by random edge sampling and trained with adaptive data sampling for anomaly detection. In addition to that it is still designed for an offline setting, our AE-based adaptive model pooling is fundamentally different from the AE-based ensemble in that we manage the dynamically changing number of models which are selectively adapted to evolving data distributions.

[1] https://github.com/kaist-dmlab/ARCUS

2.2 Streaming Anomaly Detection

Popular anomaly detection methods based on different approaches, such as k nearest neighbors (kNN), kernel density estimation (KDE), isolation forest (IF) [17], and locality sensitive hashing (LSH), have been actively extended to work online on a data stream. The kNN-based streaming methods detect outliers based on queries (e.g., NETS [33] with set-based update and MDUAL [35] with data-query duality) or local outlier factor (e.g., MiLOF [27] with summarization and DILOF [22] with sampling). The KDE-based method STARE [34] employs stationary region skipping for updating densities and anomaly scores. The LSH-based method MStream [1] uses two hashing-based features accompanied with dimensionality reduction techniques including the AE. The IF-based method RRCF [8] manages an ensemble of decision trees with a sketching technique. While they are effective in handling evolving data streams to some extent by adopting window-based processing, their focus is on reducing the computational overhead of incremental updates of a model or a pre-fixed ensemble of models for the changing data distributions. Further, they fundamentally rely on manual feature engineering such as dimensionality reduction, random sub-sampling, and linear feature transformation to deal with complex data, which often leads to sub-optimal results and limited scalability [23, 25]. In this work, ARCUS proactively adapts to evolving data streams with a dynamic model pool while being free of ad-hoc feature engineering thanks to an AE-based deep anomaly detection model.

2.3 Online Deep Learning

Online deep learning from a data stream is to manage a deep neural network-based model over a data stream for continuous inference and update. Model adaptation [26, 32] and incremental update [9, 28] are approaches popularly used in existing studies. In the model adaptation approach, the hedge backpropagation is used to determine the weight of each hidden layer when combining the outputs from all layers [26], and an attention model and a Fisher matrix are employed to adjust layer weights and exploit the previously trained weights [32]. In the incremental update approach, adaptive gradient learning adjusts the weights of gradients in response to data distribution changes [9], and local normalization handles statistical shifts in model outputs [28]. While these studies pioneered online deep learning, they focus on adapting a *single* model to a data stream. Moreover, they are designed for supervised classification [26, 32] and prediction-based time series anomaly detection [9, 28], which are outside the scope of this work.

3 PRELIMINARIES

3.1 Problem Setting

Given an unbounded sequence of data points $\langle \ldots, x_{t-1}, x_t, x_{t+1}, \ldots \rangle$ arriving in a data stream, an anomaly detection model M with parameters θ_M calculates anomaly scores of the individual data points, $\langle \ldots, M(x_{t-1};\theta_M), M(x_t;\theta_M), M(x_{t+1};\theta_M), \ldots \rangle$, continuously while updating the parameters θ_M unsupervised and reports the data points whose scores exceed a threshold as anomalies. Figure 2 illustrates this anomaly detection performed in streaming batch processing. A batch B of data points newly coming from a data stream is used by M for inference first and then used to update

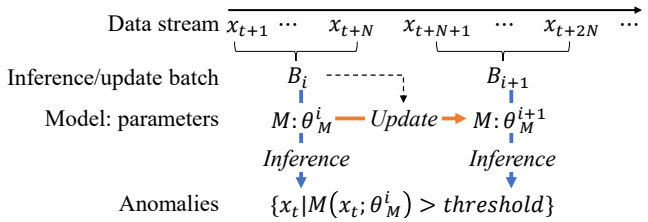


Figure 2: Batch-based continuous anomaly detection.

the parameters θ_M afterward, following the *prequential evaluation* scheme [6] designed to evaluate an online stream learning algorithm by interleaving training and testing within the same batch. The inference returns a set of anomaly scores of the data points in the batch B, i.e., $\{M(x_i;\theta_M) \mid x_i \in B\}$, abbreviated as $M(B;\theta_M)$.

3.2 Autoencoder-based Anomaly Detection

An autoencoder (AE) is a feed-forward neural network with a *encoder* E and a *decoder* D, aiming at reconstructing an input as exactly as possible, i.e., $\min_{E,D} \|X - D(Z)\|^2$ where $Z = E(X)$ is the latent representation of an input X. Typically, the reconstruction error for a given input X is used as the anomaly score of X. Motivated by the latest AE-based anomaly detection models [11, 14, 37, 38], which have shown the state-of-the-art performances, we use an AE and its variants as a base anomaly detection model M in this work.

3.3 Concept Drift

As a concept refers to a certain distributional or statistical property of data in a domain, a *concept drift* refers to an extrinsic phenomenon of the concept changing arbitrarily over time [19]. Formally, a concept drift occurs at time t if the joint probability of input data points X and their label y changes at time t, that is, $P_t(X, y) \neq P_{t+1}(X, y)$. Since $P_t(X, y) = P_t(X)P_t(y|X)$, the source of such a concept drift can be identified as one of the following three: (i) $P_t(X) \neq P_{t+1}(X)$, i.e., change in the data distribution; (ii) $P_t(y|X) \neq P_{t+1}(y|X)$, i.e., change in the anomaly decision boundary; and (iii) both (i) and (ii). Once a concept drift occurs, the current anomaly detection model becomes obsolete and should be updated to learn the new concept. Since the drift can occur in various forms, such as "sudden," "gradual," "incremental," and "reoccurring" [19, 29], it is important to have a versatile mechanism to handle all these forms equally well.

4 THE ARCUS FRAMEWORK

4.1 Overview

ARCUS is an online anomaly detection framework designed for any AE-based deep anomaly detection model. ARCUS manages a pool of models to perform inference over a batch of data stream, and then updates the model pool to adapt to new concepts detected in the batch. The overall procedure of ARCUS is illustrated in Figure 3 and outlined in Algorithm 1. Once a model pool is initialized with a model created for the first batch, ARCUS repeats, for every batch, anomaly detection using *concept-driven inference* and model pool adaptation using *concept drift-aware update*. The anomaly detection step calculates the anomaly scores of the data points in

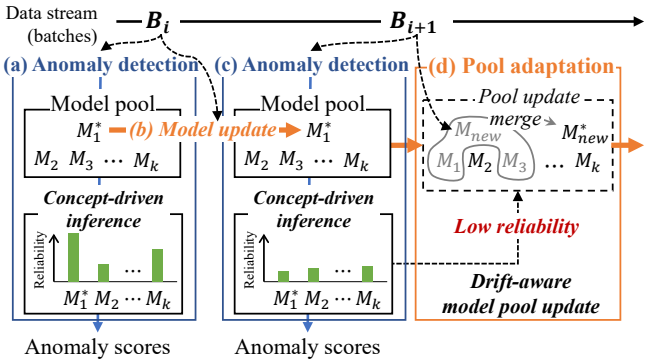


Figure 3: Overview of the procedure of ARCUS.

Algorithm 1 Overall Procedure of ARCUS

INPUT: a data stream DS, an AE-based model M, a reliability threshold α, a similarity threshold γ
OUTPUT: anomaly scores S of data points for each batch

1: Initialize a model pool P with a model built from the first batch of DS;
2: **for each** batch B of data points from DS **do**
3: /* 1. ANOMALY DETECTION */
4: Calculate the anomaly scores S of data points in B by P;
5: /* 2. MODEL POOL ADAPTATION */
6: Estimate the reliability R_P of the model pool P;
7: **if** $R_P \geq \alpha$ **then**
8: // Model update (i.e., minor update)
9: Select the most reliable model M^* in the model pool P;
10: Perform incremental update on M^* with B;
11: **else**
12: // Pool update (i.e., major update)
13: Initialize a new model M_{new} with B;
14: Compact P to P^* by recursively merging M_{new} with $M \in P$ whose similarity to M_{new} exceeds γ;
15: **end if**
16: **return** anomaly scores S;
17: **end for**

the current batch based on the reliability of individual models in the pool against the batch (Line 4 and Figures 3a and 3c). The model pool adaptation step evaluates the overall reliability of the model pool against the current batch (Line 6) and updates the model pool as needed (Lines 7–15). Specifically, if the model pool fits well, ARCUS keeps the current model pool and updates only the model contributing most to the pool's reliability (Lines 7–10 and Figure 3b); otherwise, ARCUS creates a new model and then merges it with similar existing models (to keep the model pool as compact as possible) (Lines 11–15 and Figure 3d). ARCUS then returns the anomaly scores of the current batch (Line 16). These two steps are discussed in detail in the following sections.

4.2 Model Pooling

A *model* and a *model pool* used in ARCUS are formalized as follows.

Definition 1. (MODEL) A *model* M is represented by an AE with an encoder E_M, a decoder D_M, and any additional components needed by the specific AE. □

Definition 2. (MODEL POOL) A *model pool* P is a set of models $\{M_1, M_2, \ldots, M_k\}$ that share the same architecture but have different parameters θ_{M_i} learned from different input batches. □

The membership of a model pool needs to find a balance between the efficacy and efficiency of anomaly detection. In one extreme, the pool may contain one dedicated model for every observed concept to achieve the highest accuracy while incurring significant overhead to keep all models. In the other extreme, the pool may contain only one model to minimize the update cost while sacrificing accuracy. ARCUS finds the necessary balance as adapting to data streams so that the accuracy considering all models in the pool is maximized while keeping the pool as compact as possible.

4.3 Anomaly Detection

4.3.1 ***Model Reliability.*** The concept reflected in a given model may be different from the concept of the current batch, especially when a concept drift has occurred. ARCUS estimates the *reliability* of a model by comparing the concept learned by the model with that of the current batch and uses it to calculate the concept-driven anomaly score. A straightforward way to estimate the model reliability is to directly investigate a sequence of error rates reported by the model, as adopted in the existing studies [2, 5]. However, they deal with a supervised setting where the error rate of a model is reported immediately and, thus, become impractical when the true labels of anomalies are not available in this unsupervised online anomaly detection. Thus, we exploit a set $M(B;\theta_M)$ of anomaly scores returned by a model M on a batch B, which can be obtained immediately during inference without additional work while reflecting the normal characteristics of the data points in the batch.

Let $M(B_{Curr};\theta_M)$ be the sets of anomaly scores on the current batch and $M(B_{Last};\theta_M)$ be those on the last batch used to update the model. Then, given the model M, the statistical significance of the difference between $M(B_{Curr};\theta_M)$ and $M(B_{Last};\theta_M)$ indicates how reliable the model could be to the current batch. To quantify such significance, we adopt the Hoeffding's Inequality [10]-based mean difference bound (see Theorem 1), which has been widely used for detecting statistical changes in streaming values [2, 5, 19].

Theorem 1. (HOEFFDING'S INEQUALITY-BASED MEAN DIFFERENCE BOUND) [5] Given independent random variables X and Y bounded by $[a_{min}, a_{max}]$, the probability of the sample mean difference between $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ and $\overline{Y} = \frac{1}{m}\sum_{j=1}^{m} Y_j$ is bounded by

$$Pr\{|\overline{X} - \overline{Y}| \geq \epsilon\} \leq e^{\frac{-2\epsilon^2}{(n^{-1}+m^{-1})(a_{max}-a_{min})^2}}. \quad \square \tag{1}$$

Applying Theorem 1 to $M(B_{Curr};\theta_M)$ and $M(B_{Last};\theta_M)$ gives the statistical significance of their difference in Corollary 1.

Corollary 1. (CONCEPT DIFFERENCE BOUND) Let X and Y be the independent anomaly scores returned by a model M, and let $M(B_{Curr};\theta_M)$ and $M(B_{Last};\theta_M)$ be the sets of anomaly scores sampled from X and Y, respectively. Then,

$$Pr\{|\overline{X} - \overline{Y}| \geq \epsilon\} \leq e^{\frac{-2\epsilon^2}{(b^{-1}+b^{-1})(s_{max}-s_{min})^2}} = e^{\frac{-b\epsilon^2}{(s_{max}-s_{min})^2}}, \tag{2}$$

where b is the batch size (i.e., $b = |B_{Curr}| = |B_{Last}|$),

$$\begin{aligned} \epsilon &= |avg(M(B_{Curr};\theta_M)) - avg(M(B_{Last};\theta_M))| \\ s_{max} &= max(max(M(B_{Curr};\theta_M)), max(M(B_{Last};\theta_M))), \text{ and} \\ s_{min} &= min(min(M(B_{Curr};\theta_M)), min(M(B_{Last};\theta_M))). \end{aligned} \tag{3}$$

PROOF. Straightforward from Theorem 1 [5]. □

By using the probability bound (Eq. (2)) in Corollary 1, the *reliability* of a single model is derived as in Definition 3. Thus, the probability that the sample mean difference of the anomaly scores by M is higher than or equal to ϵ is at most r_M.

Definition 3. (Model Reliability) The *model reliability* r_M of M for the current batch B_{Curr} is defined as

$$r_M = e^{\frac{-b\epsilon^2}{(s_{max}-s_{min})^2}}, \tag{4}$$

where ϵ, s_{max}, and s_{min} are the same as those in Eq. (3). □

Note that deriving the model reliability is efficient as it requires only the lightweight anomaly score statistics (i.e., the triplet of $\langle min(M(B;\theta_M)), max(M(B;\theta_M)), avg(M(B;\theta_M))\rangle$).

4.3.2 ***Concept-Driven Inference.*** The final anomaly score of a batch is determined by weighting the standardized anomaly scores of each model with its reliability (see Definition 4), making the contribution of each model proportional to its reliability.

Definition 4. (Concept-driven Anomaly Score) Given a set of models $\{M_1, \ldots, M_k\}$ in a model pool P and the corresponding reliabilities $\{r_{M_1}, \ldots, r_{M_k}\}$, the *concept-driven anomaly score* $C_P(x)$ of a data point x in the current batch B is calculated as

$$C_P(x) = \sum_{i=1}^{k} r_{M_i} \left(\frac{M_i(x;\theta_{M_i}) - avg(M_i(B;\theta_{M_i}))}{std(M_i(B;\theta_{M_i}))}\right). \quad \square \tag{5}$$

4.4 Model Pool Adaptation

4.4.1 ***Reliability of Model Pool.*** When a batch with a new concept that has never been seen arrives, none of the models in a model pool P would do inference on the batch correctly. Thus, ARCUS estimates the overall reliability of a model pool to decide whether the model pool needs to be updated.

By Definition 3, the probability that a model is not reliable for the current batch is $1 - r_M$. At the same time, the models in P are independent of one another since they have been separately updated with non-overlapping batches. Then, the reliability of P is defined by 1 minus the probability that none of the models in P are reliable, as formulated in Definition 5.

Definition 5. (Model Pool Reliability) Given a model pool $P = \{M_1, \ldots, M_k\}$, the *reliability* R_P of P is $1 - \prod_{i=1}^{k}(1 - r_{M_i})$. □

4.4.2 ***Model Merging.*** The latent representation Z learned by an AE-based model is expected to contain minimal but sufficient information of the input. If two models show similar latent representations of the same input, they must have been updated by the temporally separate disjoint batches of similar concepts, so merging them helps to remove redundancy in the model pool and also avoid overfitting. To this end, ARCUS uses the *centered kernel alignment (CKA)* for measuring the similarity of two models (see Definition 6), as it is known as the most appropriate similarity index for neural network representations since it is invariant to orthogonal transformation and isotropic scaling but not invariant to invertible linear transformation [13].

Definition 6. (Model Similarity) Given an input X and two models M_1 and M_2, let Z_1 and Z_2 be the latent representations of X by M_1 and M_2, respectively. For a kernel $\mathcal{K}$, let K_{ij}^{Z} be $\mathcal{K}(z_i, z_j)$ for $z_i, z_j \in Z$. Then, the *similarity* between Z_1 and Z_2, $Sim(Z_1, Z_2)$, is calculated as

$$CKA(K^{Z_1}, K^{Z_2}) = \frac{HSIC(K^{Z_1}, K^{Z_2})}{\sqrt{HSIC(K^{Z_1}, K^{Z_1})HSIC(K^{Z_2}, K^{Z_2})}}, \tag{6}$$

where $HSIC(K^{Z_1}, K^{Z_2})$ is Hilbert-Schmidt Independence Criterion [7]. We used a linear kernel $\mathcal{K}$, which is simple but comparable with other kernels [13]. Then, Eq. (6) becomes equivalent to

$$\|{Z_1}^T Z_2\|_F^2 / (\|{Z_1}^T Z_1\|_F \|{Z_2}^T Z_2\|_F),$$

where $\|\cdot\|_F$ denotes the Frobenius norm. □

The merging of two models M_1 and M_2 is conducted by weighted averaging of their parameters (see Definition 7). In federated learning, when local models with the same initialization of parameters are optimized via stochastic gradient descent, parameter averaging has been proven to be equivalent to gradient averaging and converging to the global model [16, 21]. Thus, an AE-based anomaly detection model merged from two models trained with two temporally separate disjoint batches of the same concept is guaranteed to *converge* to a model trained with the entire data set of the concept.

Definition 7. (Model Merging) Given two models M_1 and M_2, their number of batches N_{M_1} and N_{M_2} used to update the models, and their parameters θ_{M_1} and θ_{M_2}, the *merged model* is defined to have the parameters

$$\theta_{M_{merged}} = (N_{M_1}\theta_{M_1} + N_{M_2}\theta_{M_2})/(N_{M_1} + N_{M_2}). \quad \square \tag{7}$$

4.4.3 ***Drift-Aware Model Pool Update.*** ARCUS monitors the reliability of a model pool and triggers the update of the pool with a significance level $1 - \alpha$. A model pool will be kept unchanged when it has at least a single highly reliable model (i.e., $r_M > \alpha$), but it will be adjusted when the models in the pool have only neutral reliability values (i.e., $r_M \ll \alpha$). We set the default value of α to 0.95, which is commonly used in the statistical significance test, meaning that only 5% of the possibility that all models are not reliable will be allowed, and also confirmed to be valid in the sensitivity analysis in Section 5.6.

Once the update of the model pool is triggered, ARCUS first creates a new model with the current batch and derives a compact model pool (see Definition 8) in a greedy way, by recursively merging the new model with the most similar model exceeding a similarity threshold γ.

Definition 8. (Compact Model Pool) Given a model pool P, a new model M_{new} with a current batch B, and a similarity threshold γ, a *compact model pool* P^* satisfies

$$\begin{gathered} \min_{P^*} |P^*| \\ \text{s.t. } Sim(Z^B_{M_{new}}, Z^B_{M_i}) < \gamma \text{ for every } M_i \in P^* \text{ and } M_i \neq M_{new}, \\ \text{where } Z^B_{M_{new}} = E_{M_{new}}(B) \text{ and } Z^B_{M_i} = E_{M_i}(B). \quad \square \end{gathered} \tag{8}$$

We set the default value of γ in $[0, 1]$ to 0.8 to allow both the diversity (i.e., $\gamma > 0.5$) and the compactness (i.e., $\gamma < 1$) of a model pool. We have confirmed that the similarity of models trained under the same concept is usually higher than 0.8 (in Figure 10 in Appendix A.2), which was also valid in the sensitivity analysis in Section 5.6.

Table 1: Concept-drift data streams used for evaluation. A label after a data set name is the type of concept drift in the data set.

Data set	Description	# Obj	# Dim	Concept drift type	Anomaly target
MNIST-AbrRec	Handwritten digits	20,480	784	Abrupt and recurrent	Arbitrary digits
MNIST-GrdRec	Handwritten digits	20,480	784	Gradual and recurrent	Arbitrary digits
FMNIST-AbrRec	Fashion items	20,480	784	Abrupt and recurrent	Arbitrary fashion items
FMNIST-GrdRec	Fashion items	20,480	784	Gradual and recurrent	Arbitrary fashion items
INSECTS-Abr	Optical sensor values for insects	52,848	33	Abrupt	The southern house mosquito
INSECTS-Inc	Optical sensor values for insects	57,018	33	Incremental	The southern house mosquito
INSECTS-IncGrd	Optical sensor values for insects	24,150	33	Incremental and gradual	The southern house mosquito
INSECTS-IncRec	Optical sensor values for insects	79,986	33	Incremental and recurrent	The southern house mosquito
GAS	Gas sensor values	13,910	128	Unknown	Acetaldehyde
RIALTO	Building images around a bridge	82,250	27	Unknown	The building class 0

4.5 Complexity Analysis

Theorem 2. Given the batch size b, the number e of epochs, the number k of models in a model pool, and the model parameter size T, the time complexity of ARCUS is $O(b^2 + bT)$ and the space complexity of ARCUS is $O(kT)$.

Proof. The time complexity for the anomaly detection step is $O(kbT)$ and that for the concept drift adaptation step is $O(ebT + kb^2 + kT)$ where is $O(ebT)$ for updating a model, $O(kb^2)$ is for calculating CKA-based similarities, and $O(kT)$ is for merging models. Since typically $b, T \gg e, k$, the total time complexity is $O(b^2 + bT)$. The space complexity for managing models is $O(kT)$ and that for managing anomaly score statistics is $O(k)$, and thus the total space complexity is $O(kT)$. □

5 EXPERIMENTS

We conducted thorough experiments to evaluate the performance of ARCUS. The results are summarized as follows.

- In the *ten* high-dimensional and concept-drifted data sets, ARCUS outperforms the *ten* state-of-the-art streaming anomaly detection algorithms with different approaches in terms of online anomaly detection accuracy (Section 5.2) .
- ARCUS is able to detect exactly and adapt promptly to various types of real concept drifts (Section 5.3).
- The two main techniques employed in ARCUS are both highly effective to improve the accuracy (Section 5.4).
- ARCUS is efficient and scalable with respect to varying input data rates and concept drift types (Section 5.5).
- ARCUS is robust to the variation of its own hyperparameter values, and their default values are valid (Section 5.6).

5.1 Experiment Setup

5.1.1 ***Data Sets.*** Table 1 shows the summarized descriptions of ten *high-dimensional* (complex) and *concept-drifting* (evolving) benchmark data sets commonly used in other relevant literature as well. The synthetic data sets are generated from MNIST [15] and FMNIST [31] by simulating different concept drift types and durations, as widely used in high-dimensional anomaly detection and data stream classification [14, 26]. For the real data sets, the concept-drift types and durations are known in INSECTS [29] but not in GAS [30] and RIALTO [18]. For all data sets, the ratio of anomalies over all data points was set to 1%. Refer to Appendix A.1 for more details.

5.1.2 ***Algorithms.*** We evaluated three instances of ARCUS implemented with three state-of-the-art AE-based anomaly detection algorithms: RAPP [11], RSRAE [14], and DAGMM [38]. Each of the three ARCUS instances is denoted with their base models (e.g., ARCUS$_{RAPP}$); the only difference among them is their constituent AE model. For a more comprehensive evaluation, the streaming variants of the three AE-based algorithms and two popular RNN-based anomaly detection algorithms (LSTM-ED [20] and REBM [36]) are prepared and respectively denoted as sRAPP, sRSRAE, sDAGMM, sLSTM-ED, and sREMB. At the same time, five state-of-the-art streaming anomaly detection algorithms with different approaches (ensemble IF-based RRCF [8], LSH-based MStream [1], KDE-based STARE [34], and LOF-based MiLOF [27] and DILOF [22]) were also compared. Algorithm-specific hyperparameters were either set to the default values suggested by the authors or tuned by us to achieve the best accuracy. Details of the compared algorithms and hyperparameter settings are given in Appendix A.2.

5.1.3 ***Performance Metrics.*** Since the exact threshold of anomaly scores for verifying anomalies can vary across different applications and contexts, we used the Area Under Receiver Operating Characteristic (AUC) as the accuracy measure, which is widely used to evaluate anomaly detection [3]. For the deep learning-based algorithms, the ensemble-based RRCF, and the hashing-based MStream, the mean and the standard error of *ten* repetitions, each with the different random initialization, were measured. The wall clock time for processing a batch of varying sizes was measured for efficiency and scalability tests.

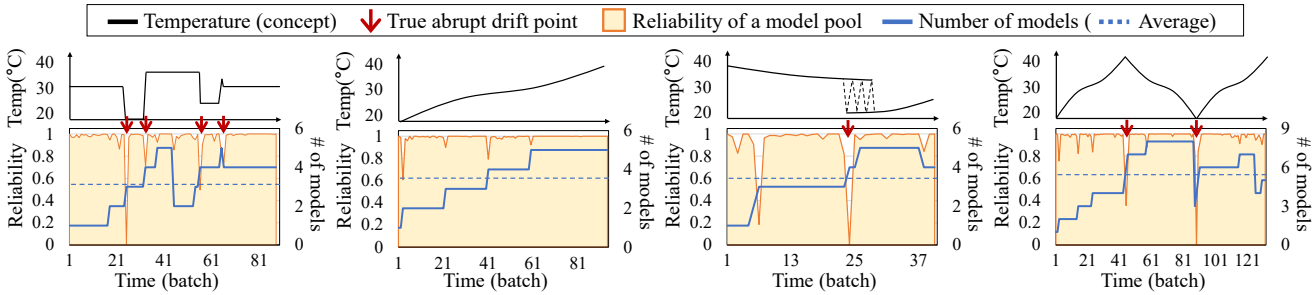
5.1.4 ***Computing Platform.*** All experiments were conducted on a Linux server with Intel Core i7-6700, 16GB RAM, and 1TB HDD. Ubuntu 16.04, Python 3.8, and TensorFlow 2.2 were installed. NVIDIA TITAN X was used for the deep learning algorithms.

5.2 Overall Accuracy Comparison

We compared the AUC achieved by the three instances of ARCUS (i.e., ARCUS$_{RAPP}$, ARCUS$_{RSRAE}$, and ARCUS$_{DAGMM}$) with other algorithms for all data sets. The results are shown in Table 2 (see Appendix A.3 for the results of sRAPP, sRSRAE, and sDAGMM). Evidently, the ARCUS instances achieved the highest accuracy in online anomaly detection for all data sets, regardless of the concept drift type. Specifically, the accuracy of ARCUS instances outperformed the best accuracy among the other algorithms by up to 36.9% (for ARCUS$_{RAPP}$ in MNIST-GrdRec), and by 12.0% on average over all data sets. The ARCUS instances improved the accuracy of the streaming variants of their base models by 10.7% for sRAPP,

Table 2: Overall performance comparison. The highest AUC results in each data set are marked in underlined bold.

	Data set	ARCUS{*base model*} *RAPP*	ARCUS{*base model*} *RSRAE*	ARCUS{*base model*} *DAGMM*	Streaming anomaly detection algorithms sLSTM-ED	sREBM	STARE	RRCF	MiLOF	DILOF	MStream
Synthetic	MNIST-AbrRec	**0.909** (±0.004)	0.831 (±0.003)	0.747 (±0.009)	0.662 (±0.006)	0.581 (±0.001)	0.574 (±0)	0.711 (±0.006)	0.460 (±0)	0.655 (±0)	0.491 (±0.002)
	FMNIST-AbrRec	**0.806** (±0.001)	0.743 (±0.006)	0.657 (±0.004)	0.772 (±0.004)	0.603 (±0.003)	0.576 (±0)	0.713 (±0.013)	0.434 (±0)	0.513 (±0)	0.717 (±0.002)
	MNIST-GrdRec	**0.904** (±0.011)	0.784 (±0.009)	0.707 (±0.002)	0.622 (±0.003)	0.502 (±0.002)	0.574 (±0)	0.660 (±0.007)	0.460 (±0)	0.649 (±0)	0.632 (±0.004)
	FMNIST-GrdRec	**0.783** (±0.012)	0.682 (±0.004)	0.652 (±0.004)	0.630 (±0.008)	0.485 (±0.003)	0.566 (±0)	0.730 (±0.010)	0.494 (±0)	0.510 (±0)	0.497 (±0.003)
Real (known drifts)	INSECTS-Abr	0.631 (±0.009)	**0.814** (±0.006)	0.652 (±0.018)	0.749 (±0.008)	0.471 (±0.001)	0.555 (±0)	0.695 (±0.018)	0.393 (±0)	0.730 (±0)	0.709 (±0.015)
	INSECTS-Inc	0.600 (±0.004)	**0.794** (±0.001)	0.572 (±0.034)	0.696 (±0.004)	0.383 (±0.001)	0.559 (±0)	0.669 (±0.011)	0.415 (±0)	0.757 (±0)	0.593 (±0.001)
	INSECTS-IncGrd	0.641 (±0.040)	**0.845** (±0.002)	0.658 (±0.028)	0.795 (±0.005)	0.575 (±0.015)	0.594 (±0)	0.719 (±0.032)	0.395 (±0)	0.746 (±0)	0.628 (±0.001)
	INSECTS-IncRec	0.634 (±0.012)	**0.811** (±0.001)	0.667 (±0.001)	0.709 (±0.005)	0.491 (±0.006)	0.551 (±0)	0.680 (±0.003)	0.381 (±0)	0.743 (±0)	0.637 (±0.001)
Real (unknown drift)	GAS	**0.878** (±0.008)	0.573 (±0.017)	0.545 (±0.039)	0.408 (±0.005)	0.506 (±0.002)	0.635 (±0)	0.804 (±0.010)	0.589 (±0)	0.470 (±0)	0.480 (±0.001)
	RIALTO	**0.784** (±0.015)	0.683 (±0.020)	0.562 (±0.020)	0.617 (±0.010)	0.492 (±0.001)	0.532 (±0)	0.731 (±0.003)	0.456 (±0)	0.742 (±0)	0.699 (±0.001)



(a) INSECTS-Abr. **(b) INSECTS-Inc.** **(c) INSECTS-IncGrd.** **(d) INSECTS-IncRec.**

Figure 4: The trends of the model pool reliability estimated by ARCUS in real concept-drifted data streams.

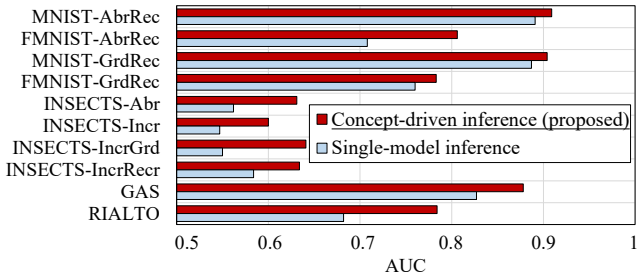


Figure 5: Ablation study results of inference strategies.

6.3% for sRSRAE, and 9.7% for sDAGMM when averaged over all data sets. This result clearly demonstrates the model-agnostic behavior and versatility of ARCUS.

5.3 Concept Drift Adaptation

We tracked how the reliability of a model pool changes in real data streams with known concept drifts (INSECTS) where temperature changes refer to concept drifts. Figure 4 shows the representative results of ARCUS$_{RSRAE}$ with the true trends of concepts and drift points. The model pool kept consistently high reliability throughout the entire data stream and the sudden drop points of the reliability were very close to the true abrupt drift points. This consistently high reliability was achieved using only five to eight models at most (3.9 models on average) in the model pool. This result means that ARCUS promptly detects and efficiently adapts to the real concept drifts so that it can achieve the highest anomaly detection accuracy with the minimum number of sufficient models. Interestingly, an unexpected drop was observed around the timestamp 7 in INSECTS-IncGrd in Figure 4c; it may have been caused by an additional unknown sudden drop of temperature during the incremental decrease or another type of concept drift (e.g., humidity change) that was overlooked.

5.4 Ablation Study

We conducted ablation studies on the two main techniques used in ARCUS—the concept-driven inference and the concept drift-aware update. We present the results of ARCUS$_{RAPP}$ while the results from the other ARCUS instances showed similar patterns. For evaluating the efficacy of concept-driven inference, we prepared the variant of ARCUS employing the *single-model inference* strategy, which uses only the most reliable model in the anomaly detection step. Figure 5 shows that the concept-driven inference consistently improved the accuracy by up to 17% over the single-model inference. This result confirms that it is worth considering various previous and ongoing concepts for calculating anomaly scores more exactly.

For evaluating the efficacy of concept drift-aware update with the proposed *similarity-based merge* strategy, we prepared the two variants of ARCUS respectively employing *always-merge* strategy, which manages a single model by always merging with a new

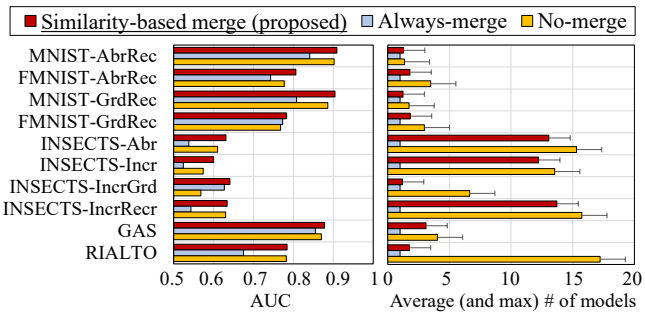
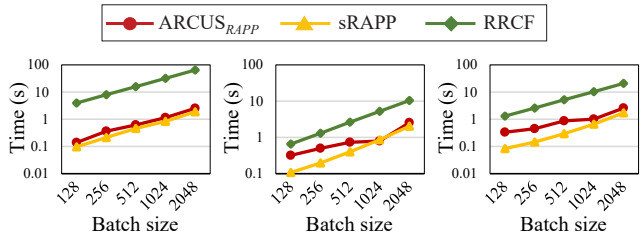


Figure 6: Ablation study results of merge strategies.



(a) MNIST-AbrRec. (b) INSECTS-IncGrd. (c) GAS

Figure 7: Scalability test results.

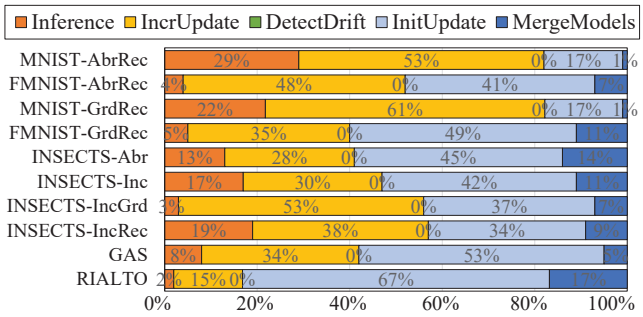
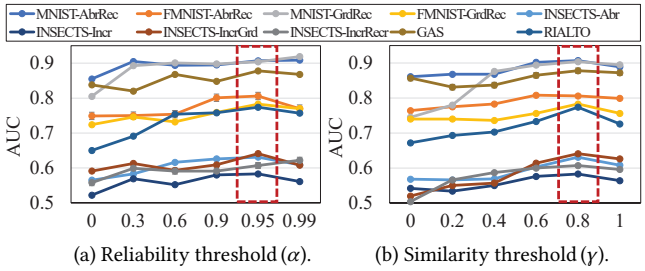


Figure 8: Processing time breakdown results.



(a) Reliability threshold (α). (b) Similarity threshold (γ).

Figure 9: Sensitivity analysis results.

model, and the *no-merge* strategy, which keeps all models without merging. As shown in Figure 6, the similarity-based merge achieved higher accuracy, by up to 17% and 13% over the always-merge and the no-merge, respectively. The similarity-based merge is more efficient than the no-merge by managing 37.6% fewer models in the model pool, averaged over all data sets. These results show that keeping an adequate number of models with significantly different properties facilitates adapting to various types of concept drifts more accurately and efficiently.

5.5 Efficiency and Scalability

We measured the processing time of ARCUS, simulating a real environment in which 128 to 2,048 devices emitting sensor-generated values. Figure 7 shows the representative results of ARCUS$_{RAPP}$, sRAPP, and RRCF (which is the ensemble-based algorithm and showed the strongest performance among the other algorithms). The other results are provided in Appendix A.4. Note that the results from the other ARCUS instances and data sets showed similar patterns. Notably, ARCUS took less than a second for processing a batch of 512 data points, which would be sufficiently fast for a high-dimensional data stream from hundreds of devices processed by a single commodity machine. At the same time, ARCUS was consistently faster than the ensemble-based RRCF in all cases, which shows the merits of dynamically managed model pool compared with pre-fixed model ensembles. The increase rate of the processing time of ARCUS was comparable to those of the baselines.

Figure 8 shows the breakdown of processing time of ARCUS$_{RAPP}$ into five steps for each data set. (The results of the other ARCUS instances were similar.) The five steps correspond to Line 4, Lines 8–9, Lines 6–7, Line 11, and Line 12 in Algorithm 1 in the paper. When averaged over all data sets, the processing time for inference and incremental model update was similar to that for the model pool adaptation consisting of drift detection, initial model update, and model merge. Except for the initial model update, which can be flexibly controlled by the user-provided number of initialization epochs, the remaining model pool adaptation (i.e., drift detection and model merging) took only 8.2% of the total processing time. Interestingly, the specific proportions of each step vary across the data sets since the behavior of ARCUS is affected by the characteristics of the data stream. For instance, the model pool adaptation of MNIST data sets took less than 20% of the total processing time, while those of GAS and RIALTO took more than 50% of the total processing time. Overall, ARCUS tended to adapt more frequently in a data stream with unknown and implicit concept drifts (e.g., GAS and RIATLO) than a data stream with known and explicit concept drifts (e.g., MNIST).

5.6 Parameter Sensitivity Analysis

We conducted sensitivity analysis on the two main parameters used in the concept drift-aware update step: the reliability threshold α (to trigger model pool update) and the similarity threshold γ (to merge models in a model pool). Due to space limitation, we present the results of ARCUS$_{RAPP}$ while the results from the other ARCUS instances showed similar patterns. For α, Figure 9a shows that AUC peaks at or nearly converges around the default value, 0.95. An extremely high reliability threshold may help improve the accuracy marginally, but it will damage the efficiency because of too frequent pool updates. For γ, Figure 9b shows that AUC peaks at or nearly converges around the default value, 0.8. As mentioned earlier, the similarity of models trained under the same concept was also usually higher than 0.8, which is also demonstrated in Figure 10 in Appendix A.2). Likewise, an extremely high similarity threshold may improve accuracy marginally, but it will incur too many models in a model pool.

6 CONCLUSION

This paper proposed ARCUS, a framework for online deep anomaly detection which can be instantiated with any AE-based deep anomaly detection method. ARCUS is specialized to handle complex evolving data streams by the *adaptive model pooling* approach with two main techniques—*concept-driven inference* and *concept drift-aware update*. In comprehensive experiments using ten data sets, ARCUS outperformed state-of-the-art streaming anomaly detection methods by up to 37% in accuracy. Much of this performance advantage is attributed to the versatile modeling power from model pooling. Overall, we believe that our work opens a new possibility in online anomaly detection research.

There are interesting directions ARCUS can be further developed into. First, the deep anomaly-detection model used to instantiate ARCUS can be extended beyond the AE. While we chose the AE as the default model to materialize the adaptive model pooling approach because of its structural simplicity and unsupervised learning mechanism, other models (e.g., GAN- or RNN-based) can be used under the same approach. The concept of model similarity, however, should be carefully tailored to the specific model since the latent representation similarity proposed for AE-based models may not always be applicable. Second, a model pool can adopt other adaptation strategies than the model initialization from scratch or the model merging based on federated learning. For instance, the initialization and update of a model can be facilitated by the incorporation of shared common knowledge of models through distillation or regularization strategies based on continual learning or transfer learning. Third, a semi-supervised approach can help understand the dynamics of a model pool and optimize it for different concept drifts observed in a data stream. A few human-provided labels or domain knowledge of anomalies or concept drifts are useful to derive the reliability threshold or the similarity threshold which are the main tunable hyperparameters in ARCUS.

ACKNOWLEDGMENTS

This work was partly supported by Samsung Electronics Co., Ltd. (IO201211-08051-01) through the Strategic Collaboration Academic Program and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00157, Robust, Fair, Extensible Data-Centric Continual Learning). The first author was also supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1A6A3A14043765).

REFERENCES

[1] S. Bhatia, A. Jain, P. Li, R. Kumar, and B. Hooi. Mstream: Fast anomaly detection in multi-aspect streams. In *Proc. WWW*, pages 3371–3382, 2021.
[2] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proc. SDM*, pages 443–448, 2007.
[3] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *DMKD*, 30(4):891–927, 2016.
[4] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. In *Proc. SDM*, pages 90–98. SIAM, 2017.
[5] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE TKDE*, 27(3):810–823, 2014.
[6] J. Gama, R. Sebastião, and P. P. Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.
[7] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Proc. ALT*, 2005.
[8] S. Guha, N. Mishra, G. Roy, and O. Schrijvers. Robust random cut forest based anomaly detection on streams. In *Proc. ICML*, pages 2712–2721, 2016.
[9] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya. Robust online time series prediction with recurrent neural networks. In *Proc. DSAA*, pages 816–825. IEEE, 2016.
[10] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.
[11] K. H. Kim, S. Shim, Y. Lim, J. Jeon, J. Choi, B. Kim, and A. S. Yoon. RaPP: Novelty detection with reconstruction along projection pathway. In *Proc. ICLR*, 2020.
[12] E. M. Knox and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. VLDB*, pages 392–403, 1998.
[13] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. In *Proc. ICML*, pages 3519–3529, 2019.
[14] C.-H. Lai, D. Zou, and G. Lerman. Robust subspace recovery layer for unsupervised anomaly detection. In *Proc. ICLR*, 2020.
[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. the IEEE*, 86(11):2278–2324, 1998.
[16] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. On the convergence of fedavg on non-iid data. In *Proc. ICLR*, 2019.
[17] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Proc. ICDM*, 2008.
[18] V. Losing, B. Hammer, and H. Wersing. KNN classifier with self adjusting memory for heterogeneous concept drift. In *Proc. ICDM*, pages 291–300, 2016.
[19] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE TKDE*, 31(12):2346–2363, 2018.
[20] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-based encoder-decoder for multi-sensor anomaly detection. In *Proc. ICML Anomaly Detection Workshop*, 2016.
[21] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, pages 1273–1282, 2017.
[22] G. S. Na, D. Kim, and H. Yu. DILOF: Effective and memory efficient local outlier detection in data streams. In *Proc. KDD*, pages 1993–2002, 2018.
[23] G. Pang, L. Cao, L. Chen, and H. Liu. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *Proc. KDD*, pages 2041–2050, 2018.
[24] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, 54(2):1–38, 2021.
[25] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft. Deep semi-supervised anomaly detection. In *Proc. ICLR*, 2020.
[26] D. Sahoo, Q. Pham, J. Lu, and S. C. H. Hoi. Online deep learning: Learning deep neural networks on the fly. In *Proc. IJCAI*, pages 2660–2666, 7 2018.
[27] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang. Fast memory efficient local outlier detection in data streams. *IEEE TKDE*, 28(12):3246–3260, 2016.
[28] S. Saurav, P. Malhotra, V. TV, N. Gugulothu, L. Vig, P. Agarwal, and G. Shroff. Online anomaly detection with concept drift adaptation using recurrent neural networks. In *Proc. CODS-COMAD*, pages 78–87, 2018.
[29] V. Souza, D. M. d. Reis, A. G. Maletzke, and G. E. Batista. Challenges in benchmarking stream learning algorithms with real-world data. *DMKD*, 34:1805–1858, 2020.
[30] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166:320–329, 2012.
[31] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.
[32] Y. Yang, D.-W. Zhou, D.-C. Zhan, H. Xiong, and Y. Jiang. Adaptive deep models for incremental learning: Considering capacity scalability and sustainability. In *Proc. KDD*, page 74–82, 2019.
[33] S. Yoon, J.-G. Lee, and B. S. Lee. NETS: Extremely fast outlier detection from a data stream via set-based processing. In *Proc. VLDB*, pages 1303–1315, 2019.
[34] S. Yoon, J.-G. Lee, and B. S. Lee. Ultrafast local outlier detection from a data stream with stationary region skipping. In *Proc. KDD*, pages 1181–1191, 2020.
[35] S. Yoon, Y. Shin, J.-G. Lee, and B. S. Lee. Multiple dynamic outlier-detection from a data stream by exploiting duality of data and queries. In *Proc. SIGMOD*, pages 2063–2075, 2021.
[36] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang. Deep structured energy based models for anomaly detection. In *Proc. ICML*, pages 1100–1109, 2016.
[37] C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proc. KDD*, pages 665–674, 2017.
[38] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *Proc. ICLR*, 2018.

A SUPPLEMENTARY MATERIAL

A.1 Data Sets

- Synthetic data sets: MNIST [15] and FMNIST [31] are handwritten digit images and fashion item images, respectively, and are widely used to *simulate* anomaly detection scenarios with a high complexity [11, 14, 37] or data streams [26, 32]. To represent a concept, we randomly set certain digits of item classes out of ten classes as anomaly target classes and the other ones as normal classes. The duration of each concept was varied randomly between one to four times the batch size. Two types of concept drifts were simulated: "abrupt and recurrent" (MNIST-AbrRec, FMNIST-AbrRec) and "gradual and recurrent" (MNIST-GrdRec, FMNIST-GrdRec).
- Real data sets (with known drifts): INSECTS [29] is a real concept-drifting benchmark data set containing optical sensor values collected while monitoring flying insects (e.g., mosquitos). Concepts are controlled by changing the temperature level, which affects the flying behaviors of insects. The southern house mosquito, which transmits zoonotic diseases, was chosen as an anomaly target class. We used four types of INSECTS data sets with different concept drift types—"abrupt" (INSECTS-Abr), "incremental" (INSECTS-Inc), "incremental and gradual" (INSECTS-IncGrd), and "incremental and recurrent" (INSECTS-IncRec).
- Real data sets (with unknown drifts): GAS [30] is a data set gathered in a gas delivery platform for 36 months and contains sensor values from monitoring six types of pure gaseous substances. A different gas concentration refers to a concept where the true drift types and timings are unknown. Acetaldehyde, a toxic chemical, was chosen as an anomaly target class. RIALTO [18] contains normalized RGB encodings from 20 days of video recordings of ten buildings around the Rialto bridge in Venice. The building class 0 was set as an anomaly target class. Weather and lighting conditions refer to concepts as they directly affect the video recording results, while the true types and timings of concept drifts are unknown.

A.2 Algorithms and Hyperparameter Settings

A.2.1 ***ARCUS and AE-based Algorithms.*** We used the three AE-based state-of-the-art anomaly detection algorithms: RAPP [11][2], RSRAE [14][3], and DAGMM [38][4]. They are all based on a basic AE and its variants (e.g., variational AE or denoising AE) but use different techniques for anomaly detection—hidden reconstruction errors along the projection pathway of encoding and decoding layers by RAPP, a linear transformation layer in the latent space by RSRAE, and the Gaussian mixture model added to the AE by DAGMM. We implemented the three algorithms based on the source code publicly available.

Each of the three algorithms was used for (i) a straightforward streaming variant and (ii) an AE-based model M of ARCUS. For the first purpose, we re-trained an AE model incrementally whenever a new batch is received. The streaming variants of them are referred to sRAPP, sRSRAE, and sDAGMM, respectively. For the second purpose (i.e., to incorporate each model into ARCUS), we implemented the interface for creating a new model, executing training epochs with incoming batches, obtaining anomaly scores, extracting latent representations, and merging with existing models. As mentioned in the main paper, the reliability threshold and the similarity threshold for ARCUS were fixed to 0.95 (the commonly-used statistical significance threshold) and 0.8 (the empirically-confirmed threshold in Figure 10), respectively.

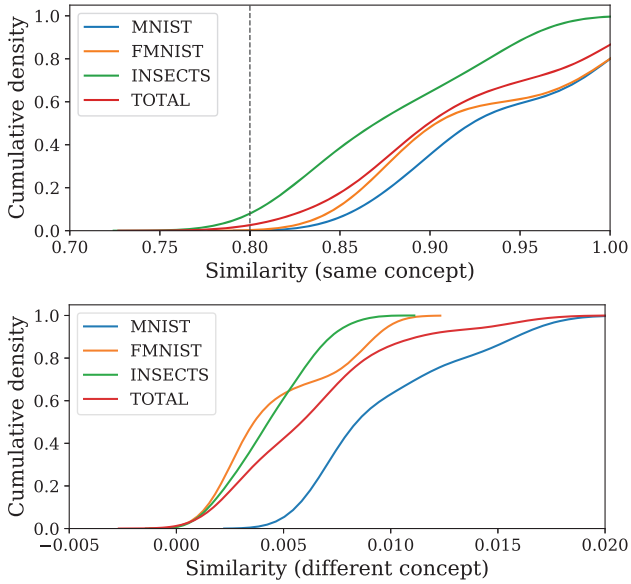
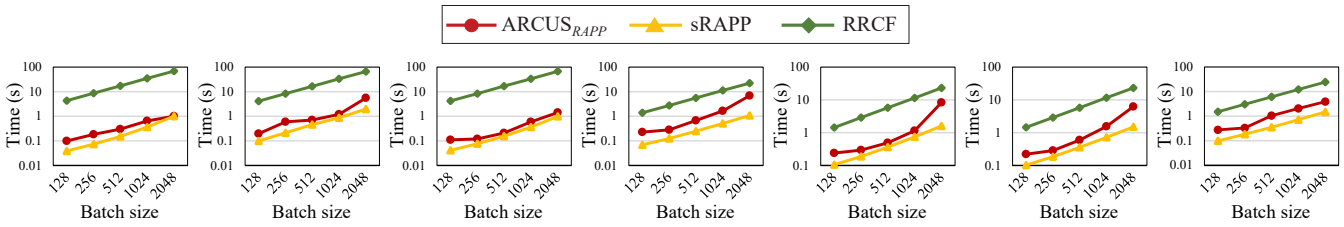


Figure 10: For three data streams with known concepts, we analyzed cumulative density as a function of the latent representation similarities between AE-based models trained under the same concept (top) or the different concepts (bottom). The similarity values between the models trained under the same concepts are mostly above 0.8, while those under the different concepts are at most 0.02.

We set the number of epochs for updating the AE-based algorithms to 5 for the initial update and 1 for the incremental update. The batch size was set to 512 (with the mini-batch size 32). The size of a latent space in AE-based models was set to the number of principal components, which guarantees at least 70% of the variance to be explained, following the relevant work [11]. The optimal number of layers (i.e., the depth of the encoder / decoder) and learning rate were tuned to achieve the best result from 2 to 5 and from 0.1 to 0.0001, respectively, for each AE-based model. Unless otherwise specified, the encoding layer sizes of the AE-based algorithms were proportionally decreased from the input dimensionality to the latent dimensionality given the number of layers, and the decoding layer sizes were increased in the opposite way. The layer size and learning rate determined for each data set and algorithm are provided with the source code for reproducibility.

[2]RAPP: https://github.com/Aiden-Jeon/RaPP
[3]RSRAE: https://github.com/dmzou/RSRAE
[4]DAGMM: https://github.com/tnakae/DAGMM



(a) FMNIST -AbrRec. (b) MNIST -GrdRec. (c) FMNIST -GrdRec. (d) INSECTS -Abr. (e) INSECTS -Inc. (f) INSECTS -IncRec. (g) RIALTO.

Figure 11: Scalability test results for all the data sets omitted in the paper.

A.2.2 ***RNN-based Algorithms.*** We prepared the streaming variants of the popular RNN-based anomaly detection algorithms LSTM-ED [20] and REBM [36][5] to be usable in our problem setting. LSTM-ED is based on an encoder-decoder network with long short-term memory, and REBM is based on a recurrent energy-based model. For the streaming variants of the models, referred to sLSTM-ED and sREBM, we trained the models incrementally by continuously feeding incoming batches over data streams. Epoch number, batch (and mini-batch) size, and latent space size were set in the same way as applied in the AE-based algorithm. The source codes of the two algorithms are available in the public repository.

A.2.3 ***Streaming Algorithms.*** We used the five existing state-of-the-art streaming anomaly detection algorithms: MStream [1], STARE [34][6], DILOF [22], MiLOF [27], and RRCF [8][7]. STARE employs kernel density estimation (KDE)-based local outlier detection, MiLOF and DILOF employ kNN for local outlier factor (LOF), RRCF employs ensemble decision trees based on the isolation forest [17], and MStream [1][8] employs locality-sensitive hashing with dimensionality reduction. We implemented the five algorithms based on the source codes provided by the authors or publicly available. Similarly to the AE- or RNN-based algorithms, incoming batches of 512 data points over data streams were fed to each algorithm for continuous inference and update.

While most hyperparameters in these algorithms are set to the default values suggested by the authors, sensitive hyperparameters are tuned to achieve the best accuracy. For STARE, MiLOF, and DILOF, the number of neighbors (for KDE-based local outlier scores or LOF scores) was tuned between 2 and the batch size. For RRCF, the number of trees was tuned from 2 to 16, and the size of the trees was tuned between the batch size and ten times the batch size. For MStream, the temporal decaying factor was tuned between 0 and 1 while using the AE-based dimensionality reduction which showed the best results in the original paper.

A.3 Performance Comparison of the Streaming Variants of AE-based Algorithms

Table 3 shows the AUC results of the streaming variants of AE-based algorithms (i.e., sRAPP, sRSRAE, and sDAGMM) which are omitted in Table 2 in the main paper. The comparison between these streaming variants and their corresponding ARCUS instances—sRAPP and ARCUS$_{RAPP}$, sRSRAE and ARCUS$_{RSRAE}$, and sDAGMM and ARCUS$_{DAGMM}$—directly shows the benefit of our adaptive model pooling technique. The streaming variants showed much lower accuracy than the corresponding ARCUS instances, because the variants have limitations in handling diverse concept drifts.

Table 3: The AUC results of the streaming variants of AE-based algorithms omitted in Table 2.

	Data set	sRAPP	sRSRAE	sDAGMM
Synthetic	MNIST-AbrRec	0.860 (±0.006)	0.756 (±0.006)	0.637 (±0.010)
	FMNIST-AbrRec	0.697 (±0.018)	0.706 (±0.006)	0.640 (±0.012)
	MNIST-GrdRec	0.813 (±0.019)	0.752 (±0.004)	0.633 (±0.017)
	FMNIST-GrdRec	0.718 (±0.020)	0.665 (±0.004)	0.610 (±0.006)
Real (known drifts)	INSECTS-Abr	0.601 (±0.017)	0.797 (±0.002)	0.596 (±0.009)
	INSECTS-Inc	0.528 (±0.026)	0.765 (±0.002)	0.545 (±0.024)
	INSECTS-IncGrd	0.553 (±0.010)	0.827 (±0.001)	0.628 (±0.025)
	INSECTS-IncRec	0.562 (±0.024)	0.786 (±0.001)	0.606 (±0.011)
Real (unknown drift)	GAS	0.813 (±0.011)	0.514 (±0.007)	0.435 (±0.031)
	RIALTO	0.712 (±0.033)	0.579 (±0.006)	0.539 (±0.023)

A.4 Scalability of Compared Algorithms

Figure 11 shows the scalability evaluation results of ARCUS$_{RAPP}$ for varying batch size in the seven data sets omitted in Figure 7 in the main paper. Note that both x-axis and y-axis are in a log scale. Similarly, the time ARCUS took was less than a second for processing hundreds of high-dimensional data points, and its rate of increase was comparable to that of the baseline for all data sets with different concept drifts. This result, again, demonstrates that ARCUS is scalable with respect to varying input data rates regardless of concept drift types.

[5]LSTM-ED and REBM: https://github.com/KDD-OpenSource/DeepADoTS
[6]STARE: https://github.com/kaist-dmlab/STARE
[7]RRCF: https://github.com/kLabUM/rrcf
[8]MStream: https://github.com/Stream-AD/MStream