MOStream: A Modular and Self-Optimizing Data Stream Clustering Algorithm

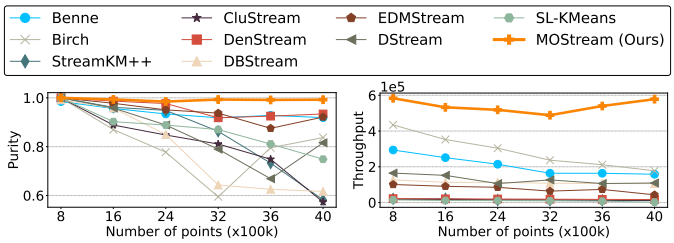
Zhengru Wang*, Xin Wang*, Shuhao Zhang

***Abstract*—Data stream clustering is a critical operation in various real-world applications, ranging from the Internet of Things (IoT) to social media and financial systems. Existing data stream clustering algorithms often lack the flexibility and self-optimization capabilities needed to adapt to diverse workload characteristics such as outlier, cluster evolution and changing dimensions in data points. These limitations manifest in suboptimal clustering accuracy and computational inefficiency. In this paper, we introduce *MOStream*, a modular and self-optimizing data stream clustering algorithm to dynamically balance clustering accuracy and computational efficiency at runtime. *MOStream* distinguishes itself by its adaptivity, clearly demarcating four pivotal design dimensions: the summarizing data structure, the window model for handling data temporality, the outlier detection mechanism, and the refinement strategy for improving cluster quality. This clear separation facilitates flexible adaptation to varying design choices and enhances its adaptability to a wide array of application contexts. We conduct a rigorous performance evaluation of *MOStream*, employing diverse configurations and benchmarking it against 9 representative data stream clustering algorithms on 4 real-world datasets and 3 synthetic datasets. Our empirical results demonstrate that *MOStream* consistently surpasses competing algorithms in terms of clustering accuracy, processing throughput, and adaptability to varying data stream characteristics.**

I. Introduction

Data stream clustering (*DSC*) is a crucial operation in data stream mining, with applications in network intrusion detection [1], social network analysis [2], weather forecasting [3], financial market analysis [4], and online recommendation [5]. Unlike traditional batch clustering algorithms such as KMeans [6], [7] and DBSCAN [8], *DSC* algorithms dynamically group incoming data based on attribute similarities and adjust clustering results as new data streams in. *DSC* algorithms must be efficient [9], [10], as they are used in high-velocity environments requiring real-time decisions. Additionally, they face challenges such as *cluster evolution* and *outlier evolution* [2], [11]–[14], which involve changes in data distribution and the emergence of new outliers.

For example, in network intrusion detection systems (NIDS) [15], *DSC* is used to continuously monitor network traffic for abnormal patterns that could indicate security breaches. As network conditions evolve with new attack types or traffic changes, *DSC* algorithms must adapt in real-time to detect emerging threats without degrading performance. The development of numerous *DSC* algorithms [2], [16]–[18], [18]–[25] reflects the diverse application needs and performance requirements in real-time environments.



(a) Accuracy (b) Efficiency

Fig. 1: Performance of nine representative *DSC* algorithms and *MOStream* on *KDD99* with high frequency of outlier evolution and increasing frequency of cluster evolutions. Measurements are taken at intervals of 800K data points.

However, existing *DSC* algorithms often lack the flexibility and self-optimization required for diverse application contexts. Typically, they consist of four main components [26]: summarizing data structure, window model, outlier detection mechanism, and refinement strategy. Design choices in these components vary based on application needs, making it complex to select an appropriate algorithm for diverse workloads, given the interdependent trade-offs involved [26].

Figure 1 shows the performance comparison of nine representative *DSC* algorithms, including Benne [26], BIRCH [16], StreamKM++ [22], CluStream [17], DenStream [18], DBStream [19], EDMStream [2], DStream [20], and SL-KMeans [21], compared to our proposed algorithm *MOStream* on a dataset with high outlier evolution and cluster evolution. The accuracy and efficiency of existing algorithms degrade significantly as data streams evolve, primarily due to their static configurations. While Benne offers flexibility through manual design choice selection, it requires user intervention, which is impractical in rapidly changing environments. In contrast, *MOStream*[1] achieves superior accuracy and efficiency by dynamically adapting to workload changes with minimal overhead.

MOStream adopts a **modular** architecture, enabling flexible choices at each design aspect and real-time adaptation to evolving workloads. It integrates dynamic adaptability, comprehensive component selection, and novel adaptation techniques, addressing the challenges of *DSC* through **self-optimization**:

Zhengru Wang is with Nvidia, China.
Xin Wang is with the Ohio State University, USA.
Shuhao Zhang is with the Nanyang Technological University, Singapore
*: The first two authors make equal contributions.

[1]Code, public and synthetic datasets are public available at https://github.com/intellistream/Sesame

1. Dynamic Adaptability: MOStream detects changes in workload characteristics and reconfigures itself in real-time, maintaining optimal performance via selecting the most suitable design options under varying stream conditions, such as shifts in data dimensionality and outlier frequency.

2. Comprehensive Component Selection: MOStream 's automated selection mechanism optimizes component interplay, ensuring holistic adaptability across diverse data stream scenarios.

3. Novel Adaptation Techniques: MOStream introduces techniques for flexible migration and stream characteristic detection, significantly enhancing its adaptability and efficiency compared to existing algorithms.

Extensive empirical evaluations across nine representative *DSC* algorithms on four real-world and three synthetic datasets demonstrate the superior clustering accuracy of *MOStream*, processing throughput, and adaptability to evolving stream characteristics, including cluster evolution, outlier evolution, and workload dimensionality. This robust performance highlights the versatility of *MOStream* and its ability to outperform existing methods across various complex, dynamic environments. In summary, the main contributions of this paper are as follows:

- We propose *MOStream*, a novel *DSC* algorithm that autonomously selects optimal configurations by dynamically adapting to evolving data streams.
- *MOStream* introduces a comprehensive component selection mechanism, optimizing key design aspects to ensure superior adaptability and performance.
- We integrate advanced adaptation techniques, such as flexible migration and stream characteristic detection, enabling *MOStream* to outperform state-of-the-art methods in both accuracy and efficiency.
- Extensive empirical evaluations on real-world and synthetic datasets demonstrate *MOStream* 's robustness, consistently surpassing existing algorithms in dynamic data stream scenarios.

The rest of the paper is organized as follows: Section II introduces the background of *DSC* algorithms. Section III discusses the algorithmic design of *MOStream*. Section IV presents our empirical evaluation of *MOStream*. Finally, Section V reviews additional related work, and Section VI concludes the paper.

II. Background

A. Data Stream Clustering Algorithms

***Definition** 1:* A **data stream** is represented as a sequence of tuples, denoted as $S = (x_1, x_2, \ldots, x_t, \ldots)$, where x_t denotes the t-th data point arriving at time t. Let C_t denote the set of clusters formed from the data stream up to time t. The distance from x_t to its closest cluster in C_t is denoted as $D(x_t, C_t)$. If $D(x_t, C_t) > \delta$, where δ is a predefined threshold, then x_t can be considered an **outlier**.

Data stream clustering (*DSC*) algorithms efficiently update C_t to C_{t+1} in response to changes in data distribution. Unlike traditional clustering algorithms such as KMeans or DBSCAN, which need to recalculate the entire clustering results from scratch upon receiving new data, *DSC* algorithms incrementally update their results to ensure more efficient clustering under the data stream scenario. Additionally, *DSC* algorithms need to dynamically and promptly handle the evolution of workload characteristics in data stream scenarios. Critical evolution aspects include *cluster evolution*, which refers to the changes of current clusters, such as splitting, merging, disappearing, and emerging upon receiving new streaming data, *outlier evolution*, which refers to the changes in the number of outliers in the data stream, and changes in data dimensionality.

B. Design Aspects of DSC Algorithms

DSC algorithms typically consist of several key components to address the challenges posed by data streams. Below we provide a brief illustration of each component and the detailed one has been provided in the previous work [26]. The pros and cons of each design choice are summarized in Table I.

a) Summarizing Data Structure provides a compact representation of data points, capturing essential information while minimizing memory usage. We consider six summarizing structures, each addressing specific challenges in stream clustering: The *Clustering Feature Tree (`CFT`)* [27] is well-suited for hierarchical clustering, allowing efficient merging and splitting of clusters as the data stream evolves. Its balanced tree structure ensures quick updates, making it ideal for large, dynamically changing datasets. *Coreset Tree (`CoreT`)* [22] leverages coreset-based techniques to provide highly accurate summaries for k-means clustering, particularly effective in high-dimensional streams where minimizing memory usage is critical. *Dependency Tree (`DPT`)* [2] captures the dependency structure among data points, which is especially useful in high-dimensional data streams where complex relationships exist between attributes. This improves the clustering quality in scenarios that require detecting these dependencies. *Micro Clusters (`MCs`)* [17] allow for incremental clustering by continuously integrating new data points without restarting the entire process, offering robustness to noise and adaptability in environments with frequent data arrivals. *Grids (`Grids`)* [20] divide the data space into fixed-size cells, providing an efficient, grid-based approach to clustering, particularly effective in lower-dimensional spaces with uniform data distribution. Lastly, the *Augmented Meyerson Sketch (`AMSketch`)* [21] approximates data streams using sketching techniques, offering a balance between clustering accuracy and computational efficiency, making it highly suitable for resource-constrained environments like edge computing.

b) Window Model manages the temporal aspect of data streams by focusing on recent data while discarding outdated points. We implemented three window models to meet different temporal needs: The *Landmark Window Model (`LandmarkWM`)* [22] segments data based on significant events or fixed time points, preserving historical data in chunks for analysis. This model is particularly useful in applications

TABLE I: Design Options Considered in *MOStream*

Aspect	Option	Pros	Cons
Data Structure	CFT	Supports a range of operations	-
	CoreT	Can handle dense data streams	Involves tree rebuild
	DPT	Can handle evolving data streams	-
	MCs	Reduces computation load	-
	Grids	Computationally efficient	Limited accuracy with changing data streams
	AMSketch	Ideal for known number of clusters	Involves sketch reconstruction
Window Model	LandmarkWM	Capable of detecting drifts	Sensitive to landmark spacing
	SlidingWM	Responsive to trends	Accuracy loss with small windows
	DampedWM	Smooth adaptation to data evolution	Sensitive to outlier effects
Outlier Detection	NoOutlierD	-	May reduce clustering quality
	OutlierD	Helps improve accuracy	-
	OutlierD-B	Prevents immediate incorporation of outliers	-
	OutlierD-T	Robust to outliers	Algorithm complexity
	OutlierD-BT	High accuracy	Requires buffer time
Refinement Strategy	NoRefine	Saves computational resources	Less accurate with evolving data
	One-shotRefine	Balances efficiency and accuracy	-
	IncrementalRefine	Adapts to new data	Computation overhead

like financial markets, where clustering is tied to specific time intervals or market events. The *Sliding Window Model (SlidingWM)* [21] maintains a fixed-size window of the most recent data points, continuously updating the clustering model as new data arrives. This model is effective in scenarios like network monitoring, where only the most current information is relevant, ensuring the clustering reflects the latest trends. The *Damped Window Model (DampedWM)* [18]–[20] applies exponentially decreasing weights to older data, gradually reducing their influence over time. This approach is ideal for applications where data evolves smoothly and older data may still be relevant, but should have less impact as time progresses, such as in sensor networks.

c) Outlier Detection Mechanism handles noise and outliers in the data stream, preventing them from distorting the clustering results. We examined five outlier detection mechanisms to address different scenarios: *Basic Outlier Detection (OutlierD)* [16] uses a fixed distance threshold from cluster centers to filter initial noise, making it efficient for simple environments with infrequent outliers. *Outlier Detection with Buffer (OutlierD-B)* [17] temporarily stores potential outliers for later re-evaluation, useful in dynamic environments where an outlier may later become part of a cluster as the stream evolves. *Outlier Detection with Timer (OutlierD-T)* [19], [20] introduces a time-based evaluation, monitoring outliers over a set window. This model is ideal for environments where outliers are transient and may re-enter clusters, such as sensor data with temporary deviations. *Outlier Detection with Buffer and Timer (OutlierD-BT)* [2], [18] combines both buffering and timing approaches, offering a robust solution for managing evolving outliers in complex data streams. This mechanism is particularly effective in high-stakes applications like fraud detection, where distinguishing between temporary anomalies and persistent outliers is critical.

d) Refinement Strategy ensures the clustering model adapts to shifts in data distribution as new data points arrive. We evaluated three strategies to meet different needs: *No Refinement (NoRefine)* [2], [16], [20], [21] is appropriate in scenarios where the initial clustering is considered sufficient and frequent updates are unnecessary, such as in static environments with minimal changes. *One-shot Refinement (One-shotRefine)* [17]–[19], [22] performs a single, comprehensive update to the clustering model after the initial clustering, ideal for periodic environments where data distribution changes infrequently but still requires occasional updates to maintain accuracy. *Incremental Refinement (IncrementalRefine)* periodically applies updates during the online clustering process, ensuring continuous adaptation to gradual data changes. This strategy works well in highly dynamic environments like traffic monitoring, where data distribution is constantly evolving and requires ongoing refinement to prevent performance degradation.

III. Design of *MOStream*

In this section, we first discuss our motivation towards the design of *MOStream* in Section III-A. We then provide a brief summary of the general workflow of *MOStream* in Section III-B. After that, we discuss the three key designs of *MOStream* to self-adjust its design components in response to dynamically detected workload characteristics of the data stream, including Regular Stream Characteristics Detection in Section III-D, Automatic Design Choice Selection in Section III-C, Flexible Algorithm Migration in Section III-E.

A. Motivation

Despite numerous *DSC* algorithms, all commonly incorporate the four key design components mentioned earlier. As summarized in Table I, Wang et al. [26] revealed that no single design choice consistently guarantees good performance across varying workload characteristics and optimization targets. Based on these findings, Wang et al. proposed Benne, the first *DSC* algorithm with a modular architecture that selects different design options from four design aspects. While Benne achieved state-of-the-art performance, it struggles with changes in stream characteristics where a fixed modular composition may not be effective. For instance, when the frequency of cluster evolution of the data stream suddenly increases, using the old configuration designed for low cluster evolution frequency

can result in poor performance. To address this issue, we designed *MOStream*, a modular and self-optimizing data stream clustering algorithm. Unlike Benne, *MOStream* dynamically detects changes in workload characteristics and reconfigures itself with the best modular composition.

B. General Workflow

The *MOStream*'s adaptability is rooted in the flexible configuration of four core components: the summarizing data structure (*struc.*), window model (*win.*), outlier detection mechanism (*out.*), and refinement strategy (*ref.*). Based on the detected evolving workload characteristics, *MOStream* automatically selects the best design option according to their pros and cons summarized in Table I and adjusts its modular configuration to cluster the new data. *MOStream* consists of four main functions to integrate these core components together. Specifically, *Insert Fun.(...)* adds the new streaming data to the current summarizing data structure, *Window Fun.(...)* manages the data points in the summarizing data structure, *Outlier Fun.(...)* assesses if the current input point qualifies as an outlier. *Refine Fun.(...)* runs the offline clustering before obtaining the final results.

Guided by real-time stream characteristics detected with sample queue (*queue*), predefined threshold values (*thresholds*), and a performance objective (o), *MOStream* dynamically tailors its components to meet the specialized requirements of diverse applications and data streams. In particular, the Regular-detection Fun (Section III-C) detects the changes of characteristics in the current data stream, Auto-selection Fun (Section III-D) selects the best modular configurations based on the detected changes of characteristics, Flexible-migration Fun (Section III-E) migrates the clustering results stored in the old modular configuration to the new one.

Algorithm 1 outlines the high-level execution flow of *MOStream*. To leverage advantageous design options under different workload characteristics and optimization targets, *MOStream* operates using a bifurcated execution strategy, comprising an online phase and an offline phase. In the online phase, the algorithm sequentially processes incoming data points. Initially, parameters and design choices for the summarizing data structure (*struc.*), window model (*win.*), outlier detection mechanism (*out.*), and refinement strategy (*ref.*) are initialized. For each incoming data point p, it is pushed into the sample queue (*queue*). When the queue is full, the algorithm performs regular detection to assess the stream characteristics. The current design choices are stored as *struc_old.*, and the characteristics are detected using the *Regular-detection Function*. Based on the detected characteristics, the *Auto-selection Function* dynamically selects the optimal design choices for *struc.*, *win.*, *out.*, and *ref.*. The *Flexible-migration Function* then facilitates the transition to the new design choices, and the queue is emptied after the selection process. The *Window Function* processes the data points within the current window model. If an outlier detection mechanism is specified (*out.* $\neq$ `NoOutlierD`), the algorithm uses the *Outlier Function* to check if the point p is an outlier. If p is not an outlier, it is inserted into the summarizing data structure and the structure is updated using the *Insert Function*. If no outlier detection is specified, the point p is directly inserted into the summarizing data structure. If the refinement strategy is set to *Incremental Refinement*, the algorithm updates the clustering model using the *Refine Function*. In the offline phase, after processing the entire data stream, if the refinement strategy is set to *One-shot Refinement*, the clustering model is updated using the *Refine Function* to finalize the clustering results.

Algorithm 1: Execution flow of *MOStream*

```
/* p: Input point, c: Temporal clusters, struc.: Type
   of summarizing data structure, win.: Type of window
   model, out.: Type of outlier detection mechanism,
   ref.: Type of refinement strategy, queue: Batch of
   stream for detection, thresholds: Identifying
   characteristic changes, o: User's primary
   performance objective                                  */
1  Function Exe(p, c, struc., win., out., ref., queue, thresholds, o):
       /* Online Phase                                    */
2      Initialize parameters and design choices;
3      while processing input streams do
4          if queue is full then
5              struc_old = struc.;
6              characteristics = Regular-detection
                 Fun.(queue, thresholds);
7              struc., win., out., ref. = Auto-selection
                 Fun.(characteristics);
8              Flexible-migration Fun.(struc_old, struc.);
9              Empty queue;
10         else
11             queue.push(p);
12         Window Fun.(win.);
13         if out. != NoOutlierD then
14             b = Outlier Fun.(p, out.);
15             if b = false then
16                 Insert Fun.(p, c) // Insert p into cluster c
                     and update c
17         else
18             Insert Fun.(p, c) // Insert p into cluster c and
                 update c
19         if ref. = IncrementalRefine then
20             Refine Fun.(ref.);
       /* Offline Phase                                   */
21     if ref. = One-shotRefine then
22         Refine Fun.(ref.);
```

C. Regular Stream Characteristics Detection

Various stream characteristics such as data dimensionality, cluster evolution, and the number of outliers are subject to change in evolving data streams. To make informed decisions about the most appropriate design choices for real-time clustering, *MOStream* must first ascertain the current characteristics of the data stream. Algorithm 2 delineates the procedure *MOStream* employs to automatically detect key attributes like dimensionality, cluster evolution, and the number of outliers in the current data stream.

Specifically, the algorithm initializes various counters and variables to store stream characteristics at Lines 3-5 of Algorithm 2. For each new data point in the $queue$ (Line 6), *MOStream* evaluates its dimensionality. If the dimension exceeds the threshold T_d, the counter $high\_dim\_data$ is incremented (Line 7). The variance of

Algorithm 2: Regular-detection Function of *MOStream*

```
/* queue: a queue of new input stream data stored for
   detecting stream characteristic changes, centers:
   previous clustering centers of the stream data,
   thresholds: threshold values for identifying the
   characteristics, num_samples: the number of
   samples in the current data stream window        */
1  Function Regular-detection
     Function(queue, centers, thresholds, num_samples):
       /* Initialize stream characteristics          */
2      characteristics = null;
3      high_dim_data = var = num_of_outliers = 0;
4      new_center = mean(queue);
5      for s ∈ queue do
6          if s.dim > T_d then
7              high_dim_data += 1 ;
8          var = UpdateVariance(var, s, new_center);
9          if min(dist(s, centers)) > thresholds.dist then
10             num_of_outliers += 1 ;
11     if high_dim_data > num_samples/2 then
12         characteristics.high_dimension = true ;
13     else
14         characteristics.high_dimension = false ;
15     if var > thresholds.variance then
16         characteristics.frequent_evolution = true ;
17     else
18         characteristics.frequent_evolution = false ;
19     if num_of_outliers > num_samples/2 then
20         characteristics.many_outliers = true ;
21     else
22         characteristics.many_outliers = false ;
23     Return characteristics;
```

Algorithm 3: Auto-selection Fun. of *MOStream*

```
/* objective: User's primary performance objective
   (Accuracy, Efficiency, or Balance); data: Input
   data stream; characteristics: Stream characteristics
   detected by regular-detect Fun.                  */
1  Function Auto-Selection Fun.(objective, data,
     characteristics):
       /* Initialize four modular design components  */
2      struc.= win. = out.= ref. = null;
3      if objective = Accuracy then
           /* Select modules with high accuracy based on
              input data stream characteristics      */
4          if characteristics.frequent_evolution = true then
5              struc. = MCs;
6          else
7              struc. = CFT;
8          if characteristics.many_outliers = true then
9              win. = LandmarkWM;
10             out. = OutlierD-BT;
11         else
12             win. = DampedWM;
13             if characteristics.high_dimension = true then
14                 out. = OutlierD-B;
15             else
16                 out. = OutlierD-BT;
17         ref. = IncrementalRefine;
18     else
           /* Choose a structure with high efficiency
              based on input data stream characteristics
              */
19         if characteristics.frequent_evolution = true then
20             struc. = DPT;
21             win. = LandmarkWM;
22         else
23             struc. = Grids;
24             win. = SlidingWM;
25         out. = NoOutlierD;
26         ref. = NoRefine;
27     Return struc., win., out., ref.;
```

the data stream is updated at Line 8. The algorithm also checks whether each data point is an outlier based on its distance to the current clustering centers (Line 9-10). After processing all the data points in the $queue$, *MOStream* sets the $characteristics.high\_dimension$ attribute to "true" or "false" based on the value of $high\_dim\_data$ (Lines 11-14). Similarly, the $characteristics.frequent\_evolution$ attribute is determined based on the calculated variance (Lines 15-19), and the $characteristics.many\_outliers$ attribute is set based on the number of outliers detected (Lines 20-23).

D. Automatic Design Choice Selection

Upon receiving the stream characteristics from the *Regular-detection Function*, *MOStream* proceeds to select the most appropriate design choices based on these characteristics and the user-defined optimization objective. The detailed selection process is delineated in Algorithm 3.

If the optimization objective is set to Accuracy (Line 3), *MOStream* selects the summarizing data structure based on the frequency of cluster evolution. Specifically, `MCs` is chosen if frequent cluster evolution is detected (Line 6), otherwise `CFT` is selected (Line 8). For the window model and outlier detection mechanisms, *MOStream* selects `LandmarkWM` and `OutlierD-BT` if many outliers are present (Lines 10-11). If outliers are scarce, `DampedWM` is chosen as the window model, and the choice between `OutlierD-B` and `OutlierD-BT` for outlier detection is influenced by the data's dimensionality (Lines 13-16). The refinement strategy is set to `IncrementalRefine` (Line 18). If the optimization objective is set to Efficiency (Line 20), *MOStream* selects either `DPT` or `Grids` as the summarizing data structure based on the frequency of cluster evolution (Lines 22-25). The window model is also selected accordingly, with `LandmarkWM` chosen for frequent cluster evolution and `SlidingWM` otherwise (Lines 24-25). The outlier detection mechanism is set to `NoOutlierD` (Line 26), and the refinement strategy is set to `NoRefine` (Line 27).

E. Flexible Algorithm Migration

Due to the significant structural differences among various summarizing data structures, it is not feasible to directly transfer clustering information from the old summarizing data structure to the new one. To address this, *MOStream* employs a migration function, delineated in Algorithm 4. Specifically, if the newly selected summarizing data structure (*struc.*) differs from the previously selected one (*struc_old.*) (Line 2), the algorithm adapts as follows: For the **Accuracy Objective** (Lines 4-7), *MOStream* extracts the clustering centers (*c*) from the old summarizing data structure (Line 3). Additionally, if the old outlier detection mechanism (*out.*) is either `OutlierD-B` or `OutlierD-BT`, outliers (*o*) are also extracted (Line 6). These centers and outliers are then used to initialize the new summarizing data structure (*struc.*) (Line 7). For the **Efficiency Objective** (Lines 8-10), *MOStream* extracts

Algorithm 4: Flexible-migration Function of *MOStream*

```
/* objective: User's primary performance objective
   (Accuracy or Efficiency); struc.: New selected
   summarizing data structure; struc_old.: Previously
   selected summarizing data structure; out.:
   Previously selected outlier detection mechanism  */
1  Function Flexible-migration Function(objective, struc.,
     struc_old.):
2      if struc. != struc_old. then
3          Extract the clustering centers c from struc_old.;
4          if objective = Accuracy then
               /* Transfer the old clustering results to
                  the new summarizing data structure to
                  preserve accuracy                    */
5              if out. = OutlierD-B or OutlierD-BT then
6                  Extract the outliers o from out.;
7              Initialize the new summarizing data structure struc. with c
                 and o;
8          else
               /* Discard the old clustering results to
                  prioritize efficiency                */
9              Output c as the final result and discard the old clustering state;
10             Initialize the new summarizing data structure struc. as
                 empty;
```

the clustering centers (c) from the old summarizing data structure and sinks them into the output to avoid computational overhead (Line 9). A new, empty object is created to initialize the new summarizing data structure (*struc.*) (Line 10).

F. Further Implementation Details

MOStream is designed as a three-threaded pipeline to process data streams, approximating a realistic computational environment. Inter-thread communication is facilitated through a shared-memory queue, reducing latency associated with network transmissions. The *Data Producer Thread* loads benchmark workloads into memory and enqueues each data point into a shared queue. To simulate high-throughput, the input arrival rate is immediate, eliminating idle time for the algorithm. The *Data Consumer Thread* executes the *DSC* algorithm, processing the data stream by dequeuing input tuples from the shared queue and generating temporal clustering results. Efficiency metrics are captured in this thread for consistent comparison of various *DSC* algorithms. The *Result Collector Thread* stores the temporal clustering results from the Data Consumer Thread. It computes accuracy metrics to minimize interference with efficiency measurements. Clustering quality is evaluated using purity [28], and the ability to handle cluster evolution is assessed using CMM [29].

IV. Experimental Analysis

In this section, we present the evaluation results. All experiments are carried out on an Intel Xeon processor. Table II summarizes the detailed specification of the hardware and software used in our experiments.

A. Experiment Setups

Table III provides a summary of the datasets for evaluation. Our dataset selection is governed by two primary criteria. First, we aim for a fair and comprehensive evaluation by including the three most frequently used datasets across the

TABLE II: Specification of our evaluation platform

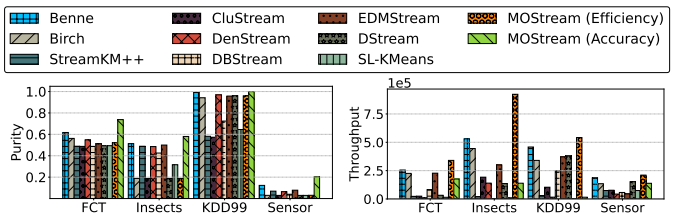
Component	Description
Processor	Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz
L3 Cache Size	48MiB
Memory	1024GB DDR4 RAM
OS	Ubuntu 22.04.4 LTS
Kernel	Linux 5.15.0-100-generic
Compiler	GCC 11.4.0 with -O3

TABLE III: Characteristics differences of selected workloads. Note that the outliers column refers to whether there are outliers in the final clustering results.

Workload	Num. Pts	Dim.	Cluster Num.	Outliers	Evolving Freq.
FCT [30]	581,012	54	7	False	Low
KDD99 [1]	4,898,431	41	23	True	Low
Insects [31]	905,145	33	24	False	Low
Sensor [32]	2,219,803	5	55	False	High
EDS [18]	245,270	2	363	False	Varying
ODS [18]	100,000	2	90	Varying	High
Dim [?]	500,000	20∼100	50	Low	Low

algorithms summarized in Table III. Specifically, *FCT* (Forest CoverType) is employed by algorithms such as SL-KMeans, StreamKM++, EDMStream, and DBStream. The *KDD99* dataset is utilized by StreamKM++, DStream, EDMStream, DenStream, CluStream, and DBStream, while *Sensor* is specifically used by DBStream. In addition to these three classical datasets, we incorporate a more recent dataset, *Insects*, which was proposed in 2020 [31]. Second, although some previous studies have proposed synthetic datasets, these datasets are not publicly available. To address this limitation and to evaluate the algorithm under varying workload characteristics, as delineated in Table III, we design three synthetic datasets: *EDS*, *ODS*, and *Dim*. The *EDS* dataset contains varying frequencies of the occurrence of cluster evolution, *ODS* features a time-varying number of outliers at different stages, and *Dim* comprises data with extremely high dimensions. A detailed account of each dataset is as follows:

- ***FCT*** (Forest CoverType) [30] consists of tree observations from four areas of the Roosevelt National Forest in Colorado. It is a high-dimensional dataset with 54 attributes, and each data point has a cluster label indicating its tree type. The dataset contains no outliers.
- ***KDD99*** [1] is a large dataset of network intrusion detection stream data collected by the MIT Lincoln Laboratory. It is also high-dimensional and contains a significant number of outliers, making it suitable for testing outlier detection capabilities.
- ***Insects*** [31] is the most recent dataset, generated by an optical sensor that measures insect flight characteristics. It is specifically designed for testing the clustering of evolving data streams.
- ***Sensor*** [32] contains environmental data such as temperature, humidity, light, and voltage, collected from sensors deployed in the Intel Berkeley Research Lab. It is a low-dimensional dataset with only five attributes but has a high frequency of cluster evolution.
- ***EDS*** is a synthetic dataset used in previous works [18] to study cluster evolution. It is divided into five



(a) Accuracy (b) Efficiency

Fig. 2: Performance Comparison on four Real-world Workloads. The throughput of DBStream on *Insects* and the throughput of SL-KMeans on *KDD99* dataset are much lower than others thus neglected to be shown in the figure.

stages according to evolving frequency, allowing for a comparative analysis of algorithmic performance across these stages.

- ***ODS*** is another synthetic dataset, distinct from *EDS* in that its second half is composed entirely of outliers, enabling an analysis of algorithmic performance under varying numbers of outliers.
- ***Dim*** is generated using the RandomTreeGenerator from the MOA framework [33]. It features data points with dimensions ranging from 20 to 100 and 50 classes, with other specific configurations set to default.

Evaluation Metrics. To measure the clustering quality of *DSC* algorithms, we use the widely adopted metric purity [28], which assesses how well the data points within each cluster belong to the same class. Additionally, we employ the Cluster Mapping Measure (CMM) [29], specifically designed to evaluate the ability of *DSC* algorithms to handle cluster evolution in data streams. CMM works by mapping clusters from one time window to the next and measuring how well the clusters preserve their structure as the data stream evolves, accounting for changes such as merging, splitting, and the emergence of new clusters. Finally, to assess performance, we introduce the throughput metric, which represents the amount of data the algorithm can process within a certain period.

B. General Evaluation of Clustering Behavior

The versatility of *MOStream* allows it to be configured into two primary variants: MOStream *(Accuracy)* and MOStream *(Efficiency)*. These variants cater to different optimization objectives based on distinct algorithmic design decisions. We initiate our evaluation by comparing the clustering behavior of these *MOStream* variants with nine existing *DSC* algorithms. The evaluation is conducted on four real-world datasets—*FCT*, *KDD99*, *Insects*, and *Sensor*. The outcomes are illustrated in Figure 2. We can see that MOStream (Accuracy) attains state-of-the-art purity across all four real-world datasets. Conversely, MOStream *(Efficiency)* exhibits purity levels comparable to existing algorithms but surpasses them in throughput. These observations confirm that by judiciously selecting and integrating different design elements, *MOStream* can either optimize for accuracy or efficiency. However, achieving both optimal accuracy and efficiency simultaneously remains elusive, corroborating the analysis regarding the trade-off between these two metrics in [26].

C. Clustering over Varying Workload Characteristics

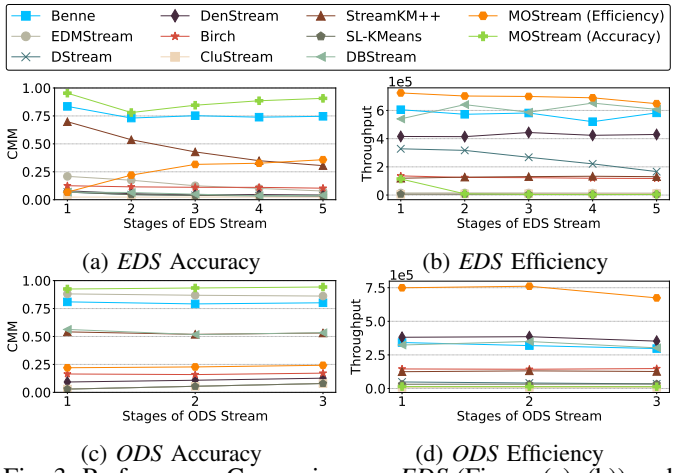
We now use three datasets *EDS*, *ODS*, and *Dim* to evaluate the impact of varying workload characteristics. Results are shown in Figure 3 and Figure 4.

First, By evaluating *EDS* and *ODS* workloads, we show that MOStream *(Accuracy)* and MOStream *(Efficiency)* maintain their respective optimization targets even under frequent cluster or outlier evolution. This stability contrasts with the deteriorating performance observed in several existing *DSC* algorithms, such as CluStream and SL-KMeans, which struggle to adapt to evolving conditions. We attribute this resilience to *MOStream*'s dynamic composition capability that will be shown in Section IV-D. Unlike most existing algorithms, *MOStream* continuously monitors stream characteristics to detect any changes, thereby enabling timely and accurate adaptations to the evolving data stream. This dynamic adaptability ensures superior clustering performance under varying conditions.

Second, we evaluate the impact of varying dimensions using the *Dim* workload, with datasets ranging from 20 to 100 dimensions (see Table III). Since existing baselines fix dimensionality, we compare only the two variants of *MOStream*. Figure 4 shows that *MOStream* maintains a consistent purity level of around 0.4 across varying dimensions. While moderate, this stability is noteworthy, especially considering the *Curse of Dimensionality*. Clustering operations, such as updating the summarizing data structure and selecting clusters, rely on distance calculations, which lose effectiveness as dimensionality increases, making it harder to distinguish data points. Additionally, we observe a decline in *MOStream*'s efficiency with higher dimensionality, mainly due to the increasing computational cost of these distance-based operations, which slow down the clustering process.

D. Empirical Evaluation of Dynamic Composition Ability

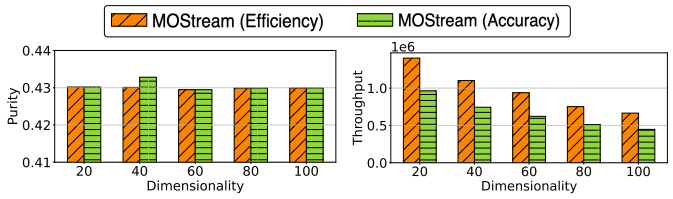
We conducted an empirical evaluation to demonstrate the effectiveness of *MOStream*'s dynamic composition abilities, including regular stream characteristics detection (Section III-C), automatic choice selection (Section III-D) and flexible algorithm migration (Section III-E) respectively under evolving workload characteristics. To assess the role of the regular stream characteristics detection module, we estimated the general location where the stream characteristics evolve in the workload and checked whether the evolution was detected timely and accurately based on the output of *MOStream*'s regular detection function, as discussed in Algorithm 2. To evaluate the effectiveness of the automatic choice selection and flexible algorithm migration modules, we introduced two additional variants: *MOStream (Accuracy) without migration* and *MOStream (Efficiency) without selection*. By comparing the changes in both purity and throughput across these four variants of *MOStream*, we identified the specific contributions of the automatic choice selection and flexible migration modules to the overall dynamic composition capability.



(a) *EDS* Accuracy (b) *EDS* Efficiency

(c) *ODS* Accuracy (d) *ODS* Efficiency

Fig. 3: Performance Comparison on *EDS* (Figure (a), (b)) and *ODS* (Figure (c), (d)) workloads with varying cluster or outlier evolution frequency. *EDS* is divided into five main stages according to the cluster evolution frequency that increases with the increase of the stages. *ODS* is divided into three main stages according to the outlier evolution frequency that increases with the increase of the stages.

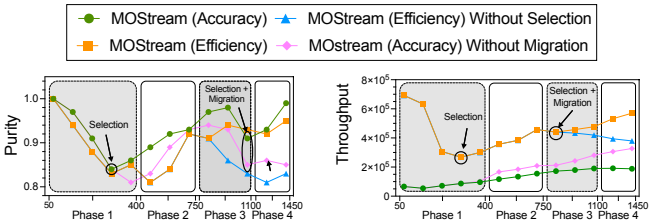


(a) Accuracy (b) Efficiency

Fig. 4: Performance Comparison on *Dim* workload with varying dimensionality.

1) Composition Effectiveness Study: We commence by evaluating the performance of the four *MOStream* variants on the real-world *KDD99* workload, characterized by a high frequency of outlier evolution and low but increasing frequency of cluster evolution. Measurements are taken at intervals of 350,000 data points with 4 phases in total, capturing both clustering outcomes and workload characteristic evolutions, as illustrated in Figure 5. Notably, *MOStream* (Accuracy) excels in purity, while *MOStream* (Efficiency) demonstrates superior throughput, each aligning with their respective optimization objectives. Three key observations emerge from this analysis.

First, a synchronized examination of *MOStream*'s clustering behavior and workload evolution (indicated by grey lines in the figure) reveals that while workload changes adversely affect clustering performance (evident in Phases 1 and 3), both *MOStream* (Accuracy) and *MOStream* (Efficiency) swiftly recover in Phases 2 and 4. For *MOStream* (Accuracy), the initial use of the `DampedWM` window model leads to a rapid decline in both purity and throughput during Phase 1 due to its inability to effectively manage increasing outlier frequencies. However, the algorithm's regular stream characteristics detection module, as outlined in Algorithm 2, identifies this issue at the close of Phase 1, prompting a



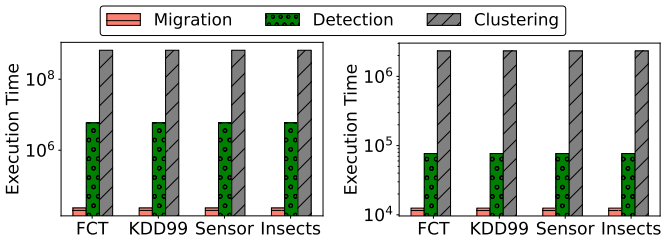
(a) Accuracy (b) Efficiency

Fig. 5: Detailed performance analysis of *MOStream*'s variants on *KDD99*. Measurements are taken at intervals of 350K data points with 4 phases in total.

transition to the `LandmarkWM` window model to better cope with outlier evolution. The subsequent improvement in purity during Phase 2 attests to the efficacy of this module. A similar recovery is observed in Phase 4, where the algorithm switches from `CFT` to `MCs` to adapt to increasing cluster evolution in Phase 3. For *MOStream* (Efficiency), the algorithm opts to forgo outlier detection to minimize overhead, in line with its optimization target. Consequently, its performance deteriorates in Phases 1 and 2. However, upon entering Phase 3, characterized by high cluster evolution, the algorithm promptly detects this change and switches from `Grids` to `DPT`, resulting in improved purity and throughput in Phase 4, as depicted in Figure 5.

Second, apparently, both the purity and throughput drops significantly when canceling the usage of automatic design choice selection and switching, as shown by *MOStream (Efficiency) without selection*. This indicates the limitation of the individual composition On the contrary, applying the automatic design choice selection into the algorithm can make full use of the strengths of every design choices, leading to both better clustering accuracy and efficiency, as shown by *MOStream* (Efficiency). Third, the inclusion of migration in the algorithm improves accuracy at the expense of clustering speed. A comparative analysis of *MOStream* (Accuracy) and *MOStream (Accuracy) without migration* reveals that while the former consistently outperforms the latter in purity, it lags in throughput. A detailed assessment of the overhead incurred by the three dynamic composition modules will be presented in the subsequent section.

2) Analysis of Composition-Related Overhead: We proceed to examine the computational overhead associated with *MOStream*'s two pivotal composition procedures: detection and migration. This is juxtaposed against the time expended on clustering, contingent on the selected composition. As illustrated in Figure 6, the time allocation for both detection and migration is relatively minimal for *MOStream* (Accuracy) in comparison to the primary clustering task. This underscores the efficiency of these composition procedures. In the case of *MOStream* (Efficiency), the scenario is somewhat different. Given that this variant is optimized for speed, the clustering operation itself is more time-efficient. Consequently, the proportion of time spent on the detection procedure appears to be larger relative to *MOStream* (Accuracy). However, it's crucial to note that *MOStream* (Efficiency) omits the migration



(a) *MOStream* (Accuracy) (b) *MOStream* (Efficiency)

Fig. 6: Execution Time Break Down Analysis on Real-world Workloads of two *MOStream* variants.

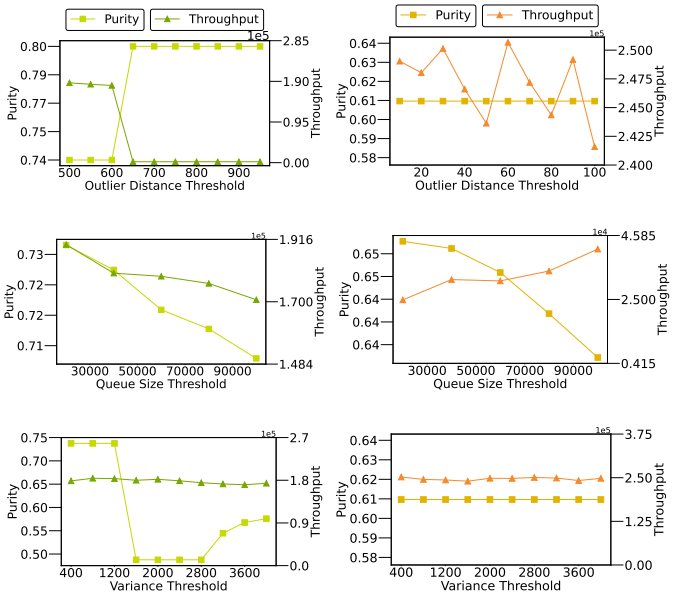
procedure altogether, as elaborated in Section III-E. This strategic omission further enhances its efficiency, aligning it closely with its optimization objectives. This analysis confirms that the overheads associated with dynamic composition in *MOStream* are well-contained, thereby not compromising the algorithm's primary objectives of either accuracy or efficiency.

E. Sensitivity Analysis of Parameters

We conducted a comprehensive sensitivity analysis of *MOStream* (Accuracy) and *MOStream* (Efficiency) across multiple datasets. Due to space constraints, we only present the results on the *FCT* workload, as shown in Figure 7. However, similar trends were observed across other datasets, with *MOStream* (Accuracy) consistently achieving higher purity levels, and *MOStream* (Efficiency) maximizing throughput. Both variants reliably meet their respective optimization goals across all tested workloads, reinforcing the generalizability of our findings.

1) **Outlier Distance Threshold** (δ): For each data point x_t and its nearest cluster C_t, we compute the distance $D(x_t, C_t)$. If this distance surpasses the threshold δ, x_t is deemed an outlier. We experimented with δ values from 50 to 950 for *MOStream* (Accuracy) and from 10 to 100 for *MOStream* (Efficiency). Both variants maintain stable purity and throughput levels across this range. Notably, *MOStream* (Accuracy) undergoes a window model transition from 'landmark' to 'damped' when the outlier distance threshold increases within a specific range, causing a sudden alteration in performance metrics, and leads to an increase in cluster size and purity. However, the throughput does not change, and this is due to the fact that both `LandmarkWM` and `DampedWM` are time consuming for clustering. Conversely, *MOStream* (Efficiency) remains unaffected as it does not consider the number of outliers as a performance metric.

2) **Queue Size Threshold**: We varied the queue size from 20,000 to 100,000. Both variants exhibit a decline in purity as the queue size increases, attributed to less frequent algorithmic adjustments. While the throughput for *MOStream* (Accuracy) diminishes with an increasing queue size due to the growing complexity of its summarizing data structure, *MOStream* (Efficiency) experiences a throughput increase. This is attributed to fewer algorithmic migrations and a transition to a more efficient data structure (`Grids`) as the queue size enlarges.



(a) *MOStream* (Accuracy) (b) *MOStream* (Efficiency)

Fig. 7: Parameter Analysis of *MOStream* variants on *FCT*.

3) **Variance Threshold**: *MOStream* activates the *characteristics.frequent evolution* flag when the calculated variance of sampled data exceeds a predefined threshold. We tested variance thresholds from 400 to 4,000. As the threshold rises, both purity and throughput for *MOStream* (Accuracy) decline. This is due to the algorithm's assumption of infrequent evaluations at higher variance thresholds, leading to less frequent algorithmic migrations and increased computational overhead. *MOStream* (Efficiency) remains relatively stable across varying variance thresholds. At a high variance threshold of 4,000, both variants achieve similar purity levels, but *MOStream* (Efficiency) outperforms *MOStream* (Accuracy) in throughput due to the absence of algorithmic migrations.

V. Related Work

Data stream clustering algorithms have evolved significantly, yet many still lack the flexibility and self-optimization needed for diverse applications, resulting in suboptimal performance. Early work by Zhang et al. [27] introduced the Clustering Feature Tree (CFT) with a static structure, limiting adaptability. BIRCH [16] identified outliers based on density thresholds but did not adapt well to evolving data. Metwally et al. [34] proposed the landmark window model, while Zhou et al. [35] and Borassi et al. [21] introduced variations of the sliding window model. These models improve adaptability but lack dynamic adjustment mechanisms. Aggarwal et al. [17] extended CFT into microclusters (MCs) and introduced the online-offline clustering paradigm and the outlier timer, yet their fixed parameters limit adaptability. Cao et al. [36] and Chen et al. [20] proposed the damped window model for temporal relevance, retaining all data but prioritizing recent information, though these models can be computationally expensive. Chen

et al. [20] also introduced a grid-based structure for efficiency. Wan et al. [18] developed the outlier buffer, improving outlier management but adding computational overhead. Gong et al. [2] presented the Dependency Tree (DPT), balancing efficiency and accuracy but struggling with complex features. These limitations highlight the need for a more flexible and adaptive approach. Our proposed algorithm, *MOStream*, addresses these gaps by offering a modular and self-optimizing framework that dynamically balances clustering accuracy and efficiency, outperforming existing methods in varied data stream scenarios.

VI. Conclusion

This paper introduced *MOStream*, a data stream clustering algorithm that autonomously selects optimal configurations based on user objectives and data stream characteristics. Our results show that *MOStream* outperforms state-of-the-art methods in accuracy and efficiency by optimizing key design aspects of *DSC* algorithms. Future work will focus on several areas to further strengthen *MOStream*. First, a formal theoretical analysis, including performance bounds and complexity analysis, is needed to provide deeper insights into the algorithm's behavior and computational requirements. Additionally, while the current implementation allows users to specify accuracy and efficiency trade-offs, integrating these parameters into *MOStream*'s self-optimizing framework could further enhance its adaptability. Lastly, expanding the evaluation to real-world production data would also provide a more comprehensive validation of the algorithm, revealing potential challenges in dynamic and noisy environments.

Acknowledgement

This work is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research & Development Programme FCP-SUTD-RG-2022-005, a MoE AcRF Tier 2 grant (MOE-T2EP20122-0010), and a Nanyang Technological University startup grant (023452-00001). Shuhao Zhang is the corresponding author.

References

[1] M. Tavallaee *et al.*, "A detailed analysis of the kdd cup 99 data set," in *CISDA*, 2009.

[2] S. Gong *et al.*, "Clustering stream data by exploring the evolution of density mountain," *Proc. VLDB Endow.*, oct 2018.

[3] K. Namitha *et al.*, "Concept drift detection in data stream clustering and its application on weather data," *Int. J. Agric. Environ. Inf. Syst.*, 2020.

[4] F. Cai *et al.*, "Clustering approaches for financial data analysis," in *Proceedings of the 8th International Conference on Data Mining (DMIN 2012), Las Vegas, NV, USA*, 2012.

[5] K. Mahadik *et al.*, "Fast distributed bandits for online recommendation systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20. New York, NY, USA: Association for Computing Machinery, 2020.

[6] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[7] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, 1982.

[8] M. Ester *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996.

[9] M. Carnein *et al.*, "An empirical comparison of stream clustering algorithms." *In Proceedings of the Computing Frontiers Conference.(CF' 17)*, 2017.

[10] S. Mansalis *et al.*, "An evaluation of data stream clustering algorithms," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 06 2018.

[11] M. Masud *et al.*, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[12] M. M. Masud *et al.*, "Addressing concept-evolution in concept-drifting data streams," in *2010 IEEE International Conference on Data Mining*, 2010.

[13] A. Haque *et al.*, "Efficient handling of concept drift and concept evolution over stream data," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, 2016.

[14] J. Lu *et al.*, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[15] B. Mukherjee *et al.*, "Network intrusion detection," *IEEE network*, 1994.

[16] T. Zhang *et al.*, "Birch: An efficient data clustering method for very large databases," *SIGMOD Rec.*, jun 1996.

[17] C. C. Aggarwal *et al.*, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003.

[18] L. Wan *et al.*, "Density-based clustering of data streams at multiple resolutions," *ACM Trans. Knowl. Discov. Data*, July 2009.

[19] M. Hahsler *et al.*, "Clustering data streams based on shared density between micro-clusters," *IEEE Transactions on Knowledge and Data Engineering*, 2016.

[20] Y. Chen *et al.*, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: Association for Computing Machinery, 2007.

[21] M. Borassi *et al.*, "Sliding window algorithms for k-clustering problems," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[22] M. R. Ackermann *et al.*, "Streamkm++: A clustering algorithm for data streams," *ACM J. Exp. Algorithmics*, May 2012.

[23] A. Bechini *et al.*, "Tsf-dbscan: a novel fuzzy density-based approach for clustering unbounded data streams," *IEEE Transactions on Fuzzy Systems*, 2020.

[24] D. Barbará, "Requirements for clustering data streams," *ACM SIGKDD Explorations Newsletter*, Jan. 2002.

[25] J. A. Silva *et al.*, "Data stream clustering: A survey," *ACM Comput. Surv.*, jul 2013.

[26] X. Wang *et al.*, "Data stream clustering: An in-depth empirical study," *Proc. ACM Manag. Data*, jun 2023.

[27] T. Zhang *et al.*, "Birch: A new data clustering algorithm and its applications," *Data mining and knowledge discovery*, 1997.

[28] J. Wu *et al.*, "Adapting the right measures for k-means clustering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: Association for Computing Machinery, 2009.

[29] H. Kremer *et al.*, "An effective evaluation measure for clustering on evolving data streams," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2011.

[30] Covertype. http://archive.ics.uci.edu/ml/datasets/Covertype.

[31] V. M. A. Souza *et al.*, "Challenges in benchmarking stream learning algorithms with real-world data," *Data Mining and Knowledge Discovery*, 2020.

[32] Sensor. https://www.cse.fau.edu/~xqzhu/stream.html.

[33] A. Bifet *et al.*, "Moa: Massive online analysis," *Journal of Machine Learning Research*, 2010.

[34] A. Metwally *et al.*, "Duplicate detection in click streams," in *Proceedings of the 14th international conference on World Wide Web*, 2005.

[35] A. Zhou *et al.*, "Tracking clusters in evolving data streams over sliding windows," *Knowledge and Information Systems*, 2008.

[36] F. Cao *et al.*, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 2006.