BURST: Rendering Clustering Techniques Suitable for Evolving Streams

Apostolos Giannoulidis
Computer Science Department,
Aristotle University of Thessaloniki
Thessaloniki, Greece
agiannous@csd.auth.gr

Anastasios Gounaris
Computer Science Department,
Aristotle University of Thessaloniki
Thessaloniki, Greece
gounaria@csd.auth.gr

John Paparrizos
The Ohio State University and
Aristotle University of Thessaloniki
Thessaloniki, Greece
jopa@csd.auth.gr

ABSTRACT

Identifying patterns or clusters in streaming time-series data is crucial for decision-making, and underpins applications such as anomaly detection, forecasting, and data quality monitoring. While numerous clustering algorithms have been proposed, many remain unexplored in the time-series domain, and others are unsuitable for streaming scenarios. Moreover, many effective methods require prior knowledge of the number of clusters, a significant limitation when dealing with evolving data streams. To address these challenges, we propose BURST, a principled and general-purpose framework that enables the application of partition-based clustering methods in streaming time-series settings. At its core, BURST integrates AutoKC, a novel, adaptive algorithm for automatically estimating the number of clusters, enhancing robustness to evolving time-series streams. Experimental analyses show that BURST is a robust strategy for real-time time-series clustering, effectively generalizing across different partitioning methods, and achieving state-of-the-art performance compared to existing algorithms.

PVLDB Reference Format:
Apostolos Giannoulidis, Anastasios Gounaris, and John Paparrizos. BURST: Rendering Clustering Techniques Suitable
for Evolving Streams. PVLDB, 18(11): 4054 - 4063, 2025.
doi:10.14778/3749646.3749675

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/agiannoul/BURST-Clustering.

1 INTRODUCTION

Pattern detection in real-time, is essential for monitoring and exploring time-series data, as it underpins unsupervised data exploration and drives actionable insights. Applications such as anomaly detection [5, 6, 8, 29, 38, 39, 55], predictive maintenance [16, 22, 27], electricity consumption forecasting [26], and data quality monitoring [19] rely on continuously capturing evolving patterns. As data continuously flows in dynamic environments, capturing evolving patterns in real-time demands novel methods that adapt to streaming setting challenges. A key methodology to extract time-series patterns in real-time is by performing online clustering [58].

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749675

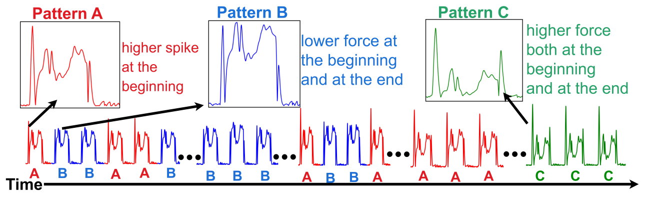

Figure 1: Evolving time-series from an industrial case study alternating between A, B, and C patterns.

Although extensive research has advanced static time-series clustering methods [2, 20, 25, 32, 37, 42, 47, 50, 51], online time-series clustering [58], remains significantly under-explored despite its practical importance on real-world problems. Online clustering inherits the challenges of streaming applications, such as concept drift, which leads to the continuous evolution of clusters [21, 56, 66, 67]. Cluster evolution in turn encompasses phenomena where data gradually transition from one cluster to another, or where an entire cluster abruptly ceases to appear. Consequently, online time-series clustering must be robust against concept drifts, through accommodating cluster merging, deletion, and updates.

Consider, for example, a cold-forming press operation in a smart factory [22]. In this scenario, sensors continuously record the force exerted when press contacts a metal strip, generating time-series signals. As illustrated in Figure 1, force signals exhibit dynamic behavior, alternating between known patterns (e.g., A and B), and occasionally introducing new dominant patterns (e.g., pattern C). Monitoring these evolving patterns is essential for understanding press operation and enabling predictive maintenance [22]. While existing state-of-the-art partition-based clustering, such as k-Shape [41], effectively discovers time-series patterns in press historical data, real-world applications require online pattern discovery. In practice, newly detected patterns are evaluated by experts, while real-time pattern detection continuously verifies whether the press is functioning as expected or showing signs of potential faults.

The above example generalizes to many other scenarios where monitored assets exhibit dynamic behavior, with new patterns continuously emerging. This evolution limits the effectiveness of static clustering techniques, which rely solely on historical data, highlighting the need for real-time clustering algorithms. While existing dynamic clustering approaches [58] have made progress in handling evolving data streams, they have not been explicitly designed neither tested for online clustering on time-series [47].

Motivated by (i) the aforementioned challenges and limitations, (ii) the real-life need to deal with continuous time-series data, and (iii) the fact that effective static clustering solutions exist for time-series, we propose the Batch-Updating Real-time assignment STrategy (BURST), a general framework for enabling real-time time-series partition-based clustering. BURST incrementally applies partition-based clustering methods (e.g., k-means [32], k-medoids [51], k-Shape [41]) to incoming data batches, enabling effective clustering through mechanisms to support cluster merging, emergence, and deletion, while also addressing scalability via batch-wise incremental processing. A common limitation of partition-based methods is the need to specify the number of clusters (k). We introduce AutoKC, a generic and model-agnostic k-estimation algorithm for partition-based clustering methods. Unlike prior approaches tied to specific algorithms (e.g., hierarchical clustering in [62] or k-means in [52]), AutoKC estimates k in a principled manner by combining adaptive thresholding with hierarchical cluster merging. It requires no external information, such as the reference sets used by gap-statistic methods [57, 62].

Our findings show that BURST can transform static clustering algorithms like k-Shape [41], k-Means [32], and k-Medoids [51], to perform effectively in an online setting. We assess the performance of the BURST framework alongside 22 variants of online clustering methods on UCR archive time-series datasets, thoroughly evaluating them on both evolving and stationary time-series streams.

We begin with background knowledge and related work in Section 2. Our key contributions are as follows:

- We present BURST, a novel framework for efficient real-time clustering in evolving data streams (Section 3).
- We propose AutoKC, an automated algorithm for identifying the most representative clusters without requiring a predefined number of clusters (k) (Section 3.2).
- We evaluate our approach alongside 22 baseline clustering methods variants presented in Section 4, on both evolving and stationary time-series streams (Section 5).
- We demonstrate BURST's generalizability in adapting partition based clustering techniques to real-time time-series clustering, and discuss its scalability (Section 6).

Finally we conclude and discuss future work in Section 7.

2 BACKGROUND AND RELATED WORK

Data streams are defined as ordered sequences of measurements, represented as an evolving sequence of real-valued numbers $T_i \in \mathbb{R}$, where i denotes the timestamp at which a particular measurement is observed. These measurements are continuously recorded, resulting in data streams of potentially infinite length. By collecting a fixed number $l \in \mathbb{N}$ of consecutive measurements, we can form fixed-sized univariate time-series $T_{i,l} = \{T_i, T_{i+1}, \dots, T_{i+l-1}\} \in \mathbb{R}^l$. The resulting time-series can be either overlapping, containing one or more common timestamps (e.g., for $l = 10$, the time-series $T_{i,10}$ and $T_{i+1,10}$ share 9 measurements), or non-overlapping, without having any shared timestamps (e.g., $T_{i,10}$ and $T_{i+10,10}$). Our goal is to discover clusters (common patterns) of time-series observed over time. Clusters representation can be vary depending on the technique, but commonly in partition-based clustering, a cluster $C^i \in C$ (where C is the set of all clusters) contain a set of similar time series ($ts \in C^i | ts \in \mathbb{R}^l$), and is represented by a center $c^i \in \mathbb{R}^l$.

Subsequently, we describe static and online clustering algorithms, either designed for general vector data or tailored to time-series.

2.1 State-of-the-art Overview

Traditional static clustering algorithms [20, 25, 32, 51] can be performed over static data to find patterns and groups in (historical) data. Such well known algorithms, like k-means [32] and k-medoids [51], can be applied to time-series considering them as multidimensional vectors. Another way to apply them is by transforming time-series to multidimensional vectors through performing feature extraction, using time-series representations [4, 17, 40], symbolic approximations [60], or extracting the catch22 [31] features, a feature set selected after highly comparative time-series analysis [31]. The work in [47] provides a taxonomy of static time-series clustering methods, categorizing them into deep learning [33], partitional [32, 41, 51], kernel-based [14, 34], density-based [20], distribution-based [13], and hierarchical [9] ones. Moreover, [47, 48] evaluates a plethora of static time-series clustering algorithms, using several distance measures [15, 18, 44, 46, 49]. Since we are interested in online clustering, static algorithms could naively used in streams, by instantiating them over an initial batch of data and then, continuously matching new time-series with the pre-calculated clusters.

Online (real-time) clustering is mainly applied to data streams. The survey in [58] provides a comprehensive taxonomy of existing streaming clustering methods [1, 10, 23, 36], categorizing them based on characteristics such as windowing strategies for stream data, outlier detection capabilities, and data structures used to summarize cluster information. However, the specific challenge of clustering univariate time-series data remains underexplored. Most existing methods are designed for vector-based data, where features observed at the same timestamps form multidimensional points. In our setting, such algorithms can be applied on either raw time-series data or extracted features from the the time series.

2.2 Static vs. Online Clustering of Time-series

In a static setting, one of the most effective methods for clustering time-series data is k-Shape [41]. This method leverages *Shape-Based Distance (SBD)*, a more robust distance measure [15, 45], to group similar time series into clusters. However, k-Shape cannot be applied to a streaming data. The algorithm computes cluster centroids, by identifying the time series that minimizes the SBD to all others within the same cluster. This process requires the full dataset, making it infeasible for real-time updates in an evolving data stream.

On the other hand, processing streaming time series can follow two approaches. The first, *batch processing*, involves collecting a fixed number of time series or waiting for a specific time period and analyzes them as a batch. In contrast, *real-time processing* [11], as defined in our work, processes each newly-arrived time series immediately upon observation, without waiting to accumulate a batch, enabling instant cluster assignment. BURST integrates instant cluster assignment, while periodically refining clusters by processing a batch of time-series together, making it a hybrid solution.

An attempt to adapt k-Shape for streaming data is found in the SAND method [7, 8]. SAND focuses on anomaly detection in time-series data [28, 30, 43] by applying k-Shape incrementally to data arriving in batches. It merges batch-level clusters with

Algorithm 1: BURST

```
Data: $KC$: A static partition-based $K$-clustering method
      $Bs$ : Batch size (integer)
1  $Init \leftarrow False$ // Indicating whether clusters have initialized.
2  $Md \leftarrow \{\}$ // Metadata
3  $C \leftarrow \{\}$ // Clusters set
4  $T_{batch} \leftarrow \{\}$ // Current batch of time series
5  foreach $T_{i,l} \in T$ do
6      $T_{batch}.append(T_{i,l})$ // Store data of current batch
7      if $Init$ then
8          $clusterAssigment(T_{i,l}, C)$ // Assign cluster
9      if $|T_{batch}| \geq Bs$ then // Calculate and merge clusters
10         $C, Md \leftarrow BatchUpdate(KC, T_{batch}, C, Md)$
11         $Init \leftarrow True,$
12         $T_{batch} \leftarrow \{\}$
```

existing clusters to deal with concept drifts, while anomalies are detected based on cluster weights that quantify cluster activity levels. Although SAND applies k-Shape in a streaming setting, it primarily targets anomaly detection and batch processing rather than real-time clustering. Critically, SAND for the clustering task can be seen as the streaming version of k-Shape, but it does not support real-time cluster assignment and requires prior knowledge of the number of clusters. While automated estimation of k has been explored in the literature, existing methods either target a single clustering technique [3], or rely on selecting the best silhouette score using k-means as an intermediate step [1], i.e., it is limited to a single partition-based clustering method and is computationally intensive. BURST addresses these limitations by enabling real-time cluster assignment and automated k estimation through AutoKC.

3 BURST DESCRIPTION

To address the challenges of applying state-of-the-art clustering methods in an online setting and handling evolving time-series data, we introduce BURST, detailed in Algorithm 1. BURST contains two core functions: (i) real-time cluster assignment and (ii) periodic batch-based cluster updates. New time series are assigned to clusters immediately for fast pattern matching, while cluster updates occur in batches to ensure efficiency. To handle infinite streams without storing raw data, BURST maintains a compact metadata structure, Md, that encodes essential cluster characteristics.

Real-time Cluster Assignment: This task is achieved by using centroids c^i calculated by the underlying clustering method. When a new time series arrives, it is assigned to its nearest cluster centroid.

Cluster Updates. Updating the set of clusters ($C^i \in C$) in a streaming context poses challenges, especially when static clustering algorithms like k-Shape are involved. BURST updates clusters upon receiving each batch of time-series data by applying the baseline algorithm (e.g., k-Shape) to the batch. After initialization, BURST merges the new batch clusters $C^{ni} \in C_{new}$ with existing clusters $C^{oi} \in C_{old}$, based on their similarity and characteristics. To address the constraints of a streaming setting, BURST avoids retaining all past time series, while it calculates weights W for each

Algorithm 2: BatchUpdate

```
Data: $KC$: A static partition-based $k$-clustering method
      $T_{batch}$: Batch of time-series data
      $C_{old}$ : Existing cluster's centers
      $Md$ : Meta Data of existing clusters
1  $C \leftarrow AutoKC(T_{batch}, KC)$
   // Match new clusters with old
2  $MC \leftarrow Match\_Clusters(C_{old}, C_{new}, C_{subs})$ // MC[j] contain
       the clusters from $C_{new}$ that are matched to $C^j_{old}$
3  $C_{merged} \leftarrow \{\}$
4  foreach $C^j_{old} \in C_{old}$ do // Merge matched clusters
       // If old cluster is matched with new batch clusters
5      if $MC[j]$ then // merge old with new clusters
6          $C^m, Md \leftarrow KC.merge(Md, C_{old^j}, \{C^i_{new} \in MC[j]\})$
7          $C_{merged} \leftarrow C_{merged} + C^m$ // Add merged cluster
8      else
9          $C_{merged} \leftarrow C_{merged} + C^j_{old}$ // Retain unmatched
               clusters
10 foreach $C^i_{new} \notin MC$ do // For each unmatched cluster
11     $C_{merged} \leftarrow C_{merged} + C^i_{new}$
12     $Md \leftarrow UpdateMetaData(Md, C^i_{new})$
   // Delete Obsolete Clusters based on weights
13 $C_{merged}, Md \leftarrow DeleteObsolete(C_{merged}, Md)$
14 return $C_{merged}, Md$
```

cluster to measure its relevance to the current data, which are used to prune outdated clusters, ensuring adaptivity and efficiency.

3.1 Cluster Updating

BURST's cluster updating involves several steps: automated batch clustering, matching new to old clusters, merging matches, adding unmatched clusters, updating weights, and removing obsolete clusters. These steps are summarized in Algorithm 2 and detailed below.

Automated Batch Clustering uses a static clustering method KC to find clusters in a batch of time-series T_{batch}, without requiring a predefined number of clusters (k parameter). To achieve that, we propose the AutoKC algorithm, which applies the static clustering method with a big k and then merges redundant clusters internally. More details on AutoKC are provided in Section 3.2.

Cluster Matching matches the clusters derived from the current batch (C_{new}) with the existing clusters (C_{old}). When a new batch of time-series arrives, the algorithm calculates clusters for the batch (T_{batch}). To determine whether a new cluster C^i_{new} matches an existing cluster C^j_{old}, the intra-cluster average distance $\tau_j = \sum_{T_{i,l} \in C^j_{old}} dist(T_{i,l}, C^j_{old})/|C^j_{old}|$ is used as a threshold. If the distance between the centers of a new cluster c^i_{new} and an old cluster c^j_{old} is smaller than τ_j, the two clusters are matched. The value of τ_j is computed in $UpdateMetaData()$ and stored in Md. In case of merging the updated τ is given by weighted sum of the merged clusters τ values, with weights being equal to the size of each cluster.

Merging matched clusters involves updating cluster centers by combining the existing cluster centers with their corresponding matched clusters from the new batch. Additionally, unmatched clusters from the new batch are treated as new clusters, with their meta-data (e.g., τ, W) calculated and added to the cluster set C_{merged}. For matched clusters, new centers are computed using information from both the old and the new clusters, ensuring that the updated clusters accurately reflect the latest data.

We use the example of k-Shape to provide more details regarding this step. k-Shape solves an optimization problem to derive in final cluster centers. Specifically, the only component of k-Shape optimization that depends on the raw time-series is the matrix that stores the sum of the dot products between all time-series within the same cluster (referred to as S matrix) [8, 41]. But, when merging clusters from older batches, we do not have access to their time-series. To solve that, instead of storing the time series of previous batches, we store only the matrix $S \in \mathbb{R}^{l \times l}$ for each cluster, as part of the meta-data, i.e., space requirements remain constant. Then, for merging two clusters, the S matrix is updated just adding the dot products of time series from the new cluster: $S^* = S_{old} + \sum_{T_i \in C^i_{new}} T_{i,l} \cdot T^T_{i,l}$. The updated S matrix is then used in the standard k-Shape procedure to compute the new cluster centers.

Weighting quantifies the relevance of each cluster to the current data. Higher weights indicate clusters with greater activity or importance, while lower weights signify clusters that are becoming obsolete. The weight w^j of a cluster C^j is defined as $w^j = \frac{|C^j|^2}{D^j * \sum_{C^j \in C} dist(c^i, c^j)}$ where $dist$ is the distance function used for time series (e.g., SBD for k-Shape), $|C_j|$ is the size of the cluster, and D_j is a decay factor. The decay factor is computed as: $D^j = 1 + max(0, current_t - Bs - last_j)$. Here, $last_j$ is the last time-series index assigned to C^j, $current_t$ is the index of the most recent time series, and Bs is the batch size. Clusters containing recent time series will have a decay factor of 1, while older clusters will have larger decay factors, resulting in lower weights. Such weights are computed for all clusters after the merging step.

Regarding updating the weights of old clusters w_{old} (potentially merged or not), to ensure the weights reflect long-term cluster relevance rather than being overly biased by the latest batch, BURST combines old and new weights using a weighted sum: $w^j = (1 - \alpha) * w^j_{old} + \alpha * w^j_{new}$ where α controls the rate of change. Finally weights are normalized to lie between 0 and 1 after computation.

Cluster Deletion improves the efficiency of the algorithm by freeing memory occupied by obsolete cluster data. This is important in evolving data streams, where a large number of obsolete patterns can accumulate over time. Since cluster weights quantify the relevance of clusters to the observed time-series, clusters with low weights (e.g., below 0.1) are considered no longer significant. These clusters, i.e., their meta-data, are deleted to optimize resource usage while ensuring that only relevant clusters are retained.

3.2 Automated k-Clustering with AutoKC

Reflect that automated determination of clusters is a crucial step for dynamically identifying time-series patterns, eliminating the need for a user-defined k parameter. AutoKC offers such a solution for partition-based clustering methods.

AutoKC initially performs a static clustering algorithm with a large initial value of k ($bigk$), which yields an initial set of clusters

Algorithm 3: AutoKC

```
Data: $B_l$ : A set of time series $B_l = \{T_{0,l}, ..., T_{m,l}\}$
      $bigk = max(20, \sqrt{|B_l|})$: Big initial k
      $KC$: A static partition-based $k$-clustering method
1  $C \leftarrow KC(B_l, bigk)$ // baseline clustering
2  $CL \leftarrow \{\{c^j\} \forall C^j \in C\}$// consider centers as clusters
3  $HD \leftarrow \{\}, step \leftarrow \{CL\}$// Keep track of merges
4  while $|CL| \neq 1$ do
5      $hd, cl_a, cl_b \leftarrow nearest\_pair(CL)$ // Merge distance: hd
6      $HD.append(hd)$
7      $CL \leftarrow CL - \{\{cl_a\}, \{cl_b\}\} + \{cl_a, cl_b\}$
8      $step.append(CL)$
9  $th \leftarrow mean(HD) + 2 * std(HD)$// 2T thresholding
10 $HD' \leftarrow \{hd \;\forall\; hd \in HD \mid hd < th\}$
11 $th' \leftarrow mean(HD') + 2 * std(HD')$
12 while $th' \neq th$ do
13     $th \leftarrow th'$
14     $HD' \leftarrow \{hd \;\forall\; hd \in HD \mid hd < th\}$
15     $th' \leftarrow mean(HD') + 2 * std(HD')$
   // Optimal center merges are those of the step which produced
   the last $HD'$ merging distance
16 $Groups \leftarrow step[HD.indexOf(HD'[-1])]$
17 $C_{final} \leftarrow \{\}$
18 foreach $G \in Groups$ do // Merge clusters of the same group
19     $C^m \leftarrow KC.innerMerge(G_{centers}, G_{ts})$
       $C_{final} \leftarrow C_{final} + \{C^m\}$
20 return $C_{final}$
```

C. Subsequently, an inner-merging of these clusters is performed using hierarchical clustering over cluster centers, as outlined in Algorithm 3. Then, the algorithm treats each cluster center c^j (which is a time series) in C as an individual cluster. Hierarchical clustering is then performed iteratively, merging the two closest clusters at each step and recording the merging distances in the list HD. Cluster distances are calculated using complete linkage (maximum linkage), which measures the farthest distance between any two points from the two clusters [35].[1] This iterative merging continues until all clusters are combined into a single cluster, resulting in an ordered sequence of merging distances stored in HD.

To determine the optimal number of clusters, we use the $2T$ adaptive thresholding method [61], originally proposed for thresholding outlier scores. This method separates the merging distances in HD into "normal" and "outlier" distances. Specifically, a statistical threshold is calculated as $th = mean(HD) + 2 \cdot std(HD)$, and distances greater than th are iteratively excluded from HD. The threshold is recalculated after each iteration until all remaining distances are below th. This iterative process ensures the threshold is not influenced by outliers. The final threshold th determines the optimal clustering step—corresponding to the last step in which the merging distance remains below th. Finally, clusters within each group identified by hierarchical clustering are merged. Coming

[1]Utilizing single linkage resulted in same performance but we leave this topic out of our scope due to space constraints.

Table 1: Algorithms used for real-time time-series clustering (22 competitors in total).

Method	Need k	Real-Time	Configuration
k-means [32]	✓	Static	L2, catch22
k-medoids [51]	✓	Static	L2, SBD, catch22
DTC [33]	✓	Static	DL, catch22
k-Shape [41]	✓	Static	SBD
Clustream [1]	✓	Stream	L2, catch22
StreamKM [36]	✓	Stream	L2, catch22
MBKmeans [54]	✓	Stream	L2, catch22
DbStream [23]	✗	Stream	L2, catch22
DenStream [10]	✗	Stream	L2, catch22
BirchK [64]	✓	Stream	L2, catch22
Birch [64]	✗	Stream	L2, catch22
BURSTK k-Shape	✓	Stream	SBD
BURST k-Shape	✗	Stream	SBD

back to the example of k-Shape, this involves recalculating the S matrix using the time-series of each group (denoted as G_{ts}) and apply k-Shape optimization for cluster center calculation [41].

In summary, AutoKC presents a principled and flexible approach to automated clustering by integrating hierarchical clustering insights with adaptive thresholding into the partition-based clustering paradigm. By eliminating the need to predefine the number of clusters and supporting a wide range of k-based algorithms, AutoKC delivers a robust, method-agnostic solution adaptable to both static and streaming data contexts, with the ability to preserve the consistency of the original clustering method.

4 BASELINES

The selected baselines originate from two main sources i) online clustering and ii) static clustering as discussed in Section 2.1.

Online clustering: To cluster streaming time-series, we adopt classical methods originally designed for vector data. Streaming clustering methods are extensively reviewed in [58] and include several well-established approaches. DenStream [10] is a density-based method that extends DBSCAN to streaming data by maintaining and refining micro-clusters over time. Similarly, DBSTREAM [23] organizes data into micro-clusters but introduces a shared density graph to improve cluster evolution tracking. CluStream [1] maintains micro-clusters while continuously updating their statistical properties, enabling dynamic adaptation to evolving data distributions. Lastly, StreamLS [36] simplifies streaming clustering by retaining only cluster centers from batches and iteratively refining them. All these methods are considered in our evaluation.

Static clustering: We employ state-of-the-art techniques designed for static time-series clustering, following the benchmarking framework of Odyssey [47]. We select the four highest-performing methods and adapt them for real-time processing. k-Shape [41] leverages SBD for improved time-series similarity measurement. k-means [32] iteratively assigns time-series to the nearest cluster center and updates the centers as the mean of assigned series. k-medoids [51] follows a similar approach but selects actual time-series as cluster centers instead of their means. Deep Temporal Clustering (DTC) [33] is a deep learning-based approach that jointly learns feature representations and cluster assignments. To integrate static clustering methods into our real-time setting, we first compute clusters using the initial batch and subsequently assign newly observed time-series to the precomputed clusters.

We categorize the clustering techniques used in our evaluation based on key characteristics, as summarized in Table 1. One distinction is streaming capability (real-time operation) or static clustering (operating on the full dataset). Another factor is the need of parameter k to define the number of clusters. Moreover, some techniques are distance-independent (e.g., k-medoids), while others are constrained by the properties of their distance functions (e.g., k-Shape with SBD and k-means requiring triangle inequality). Lastly, we explore clustering after applying catch22 [31] feature extraction on the time-series rather than directly applied to raw data.

5 EVALUATION

We use the UCR archive time-series datasets [12] to assess the performance of various clustering approaches. Each dataset consists of a collection of time-series, normalized using z-score, and tagged with ground truth classes. These are treated as streams of time-series data. The UCR time-series benchmark poses a significant challenge for clustering, as indicated by low overall silhouette scores (Euclidean: median = 0.03, IQR = [0.00–0.12]; SBD: median = 0.07, IQR = [0.00–0.23]). To demonstrate the capabilities of clustering algorithms we categorize the datasets into two main categories:

(1) Not Evolving streams: Time-series streams where the initial batch (fixed at 10% of the complete dataset) contains at least one time-series from each class in the dataset.
(2) Evolving streams: Time-series streams, where the initial batch (10%) does not contain samples from all classes.

Evolving streams are used specifically to evaluate the ability of clustering methods to discover new clusters over time. In total, there are 60 evolving and 68 non-evolving data stream.

For a fair evaluation, we further categorize techniques based on their dependency on prior knowledge of the number of clusters. Specifically, we evaluate methods that require a predefined number of clusters k (*KC*) separately from those that do not (*AC*). Clustering methods requiring a predefined k (*KC* methods) are evaluated with parameter k equal to the number of clusters observed in the initial batch. Within this category, we also include BURSTK, which represents BURST without the use of AutoKC algorithm.

5.1 Evaluation Setup

In our evaluation, we consider 22 variants of clustering methods, derived from a combination of online algorithms, static algorithms, feature transformations, and distance metrics, as presented in Section 4. A demonstration of the real-time time-series clustering setup, using BURST, is shown in Figure 2. For each stream, we segment the time series into non-overlapping samples of fixed length (l). The initial 10% of the data is used as a fitting set, which may involve identifying initial clusters or performing unsupervised training. Afterwards, new samples are processed sequentially, with each sample being assigned a cluster identifier. Depending on the algorithm, new clusters may be formed over time (e.g. Birch and Clustream).

In our setting, the real-time requirement for clustering algorithms applies only to cluster assignment, meaning that after the initial batch, each new observation is immediately assigned to a

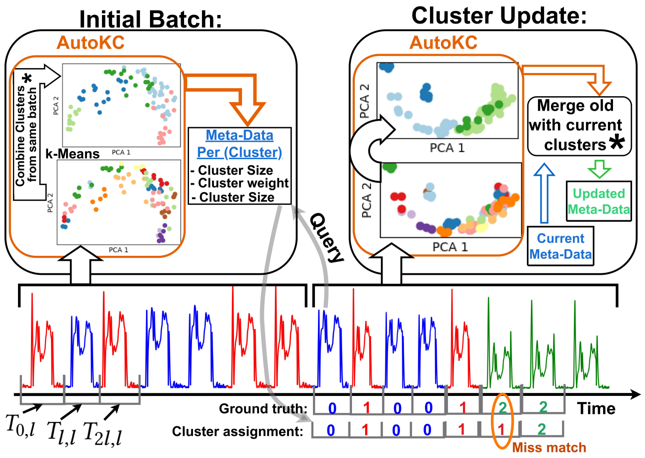

Figure 2: BURST with AutoKC adapts a static clustering method for evolving time series, assigning new data in real-time and updating clusters per batch. The required functionality for BURST compatibility is marked by *.

cluster. Beyond cluster assignment, many clustering methods also include cluster updates. These do not need to occur after every new time-series sample but can instead be performed in batch mode, after accumulating a predefined number of samples. For such algorithms, we set the update interval to one-tenth of the data stream length to ensure periodic refinement while maintaining efficiency.

To assess the performance of clustering and whether they could accurately discover and identify the existing patterns in the data stream, we compare the results of clustering with the ground truth classes of the time-series. We show two metrics, Purity [65], and Adjusted Rand Index (ARI) [24] in the manuscript. However, we have also calculated additional metrics (RI [53] and NMI [63]) that show similar results to the presented ones and are available along with our codebase at https://github.com/agiannoul/BURST-Clustering.

5.2 BURST Performance

Figure 3 presents the performance of the *KC* and *AC* methods across all time-series datasets using box plots. Additionally, it includes the performance of the best static clustering method (referred to as Oracle), applied to all available time-series data statically and using the correct number of clusters (k). Although Oracle violates the streaming clustering setting, we include its results to assess how well dynamic methods perform compared to an ideal scenario where all data and the correct k are provided. For both *KC* and *AC*, we report the best-performing variant among different choices regarding catch22 feature transformation and distance metric.

Our analysis shows that BURSTK consistently outperforms other *KC* methods in both ARI and Purity, while BURST achieves significantly higher ARI scores when the number of clusters (k) is not predefined (*AC* methods). The only instance where BURST ranks second among *AC* methods is regarding Purity, where the Birch surpasses BURST, even outperforming the Oracle. This performance discrepancy is due to Birch generating an extremely high number of clusters, over 20 times the number of real classes in the data. Predicting a large number of clusters results in high purity, as multiple time-series from the same original class tend to be grouped together (for reference, assigning each time-series to a separate cluster yields a purity of 1). However, this also leads to a low ARI, as the cluster assignments fail to align well with the true cluster structure. Adjusting Birch's parameters to reduce the predicted-to-actual cluster ratio lowers purity but does not improve ARI.

Our main emphasis is on cases of evolving patterns and techniques that do not assume predefined number of existing patterns (i.e. no k parameter). Figure 4 top panel, shows the ranking of such methods (with no need of setting the k parameter), when applied on a) evolving and b) non-evolving datasets. In evolving time seriers, BURST achieves higher ranking than the other techniques. This also, holds for non-evolving datasets, where results of BURST are statistically significant better than the remaining dynamic techniques, and without a statistically significant difference from Oracle.

Concerning the *KC* scenario, we compare the performance of clustering techniques versus BURSTK (BURST with predefined k instead of AutoKC). In Figure 4 bottom panel, we report the results for the best variant of such methods, on evolving a) and non-evolving b) datasets. BURSTK achieves the highest ranking among clustering methods on the former), while only k-Shape is better in non-evolving datasets, and without statistically significant difference from Oracle. Although BURSTK uses a static k parameter (in the *KC* setting), it can result in more than k total clusters in the end, since its mechanism dynamically forms new clusters during the batch update process. This holds also for Clustream and BirchK.

In summary, the BURST achieves state-of-the-art performance, consistently ranking first or among the best methods. Interestingly, well known dynamic clustering techniques, such as DBSTREAM and DenStream, show poor performance mainly due to the difficulty of tuning their parameters. In our experimental setting, we performed fine-tuning of parameters on 10 datasets (containing both evolving and non-evolving datasets), and then, we used this parametrization across all datasets, to estimate the robustness of techniques. BURST does not need fine-tuning; the only parameter is *bigk*, which does not impact the performance. By contrast, in methods like DenStreams, the parameter highly influences the performance (e.g., picking the right ϵ parameter, that is the maximum distance between two samples of the same cluster). Moreover, good parameters for one case, are not guaranteed that are valid in others.

5.3 BURST's Performance Sensitivity

To better understand when BURST excels or falters, we analyzed its performance relative to dataset compactness, measured by the silhouette score, where higher scores indicate more distinct, compact clusters. We selected two subsets from the evaluation datasets: the 30 with the highest and the 30 with the lowest silhouette scores. Figure 5 compares BURST with the second-best method, Birch, on these subsets. BURST performs better on datasets with high silhouette scores, consistently achieving top rankings in both ARI and Purity. The statistically significant ARI gap with Birch highlights BURST's strength in well-clustered data. In contrast, for the low silhouette score subset, BURST's performance drops, ARI scores approach 0, and Birch significantly outperforms BURST in Purity.

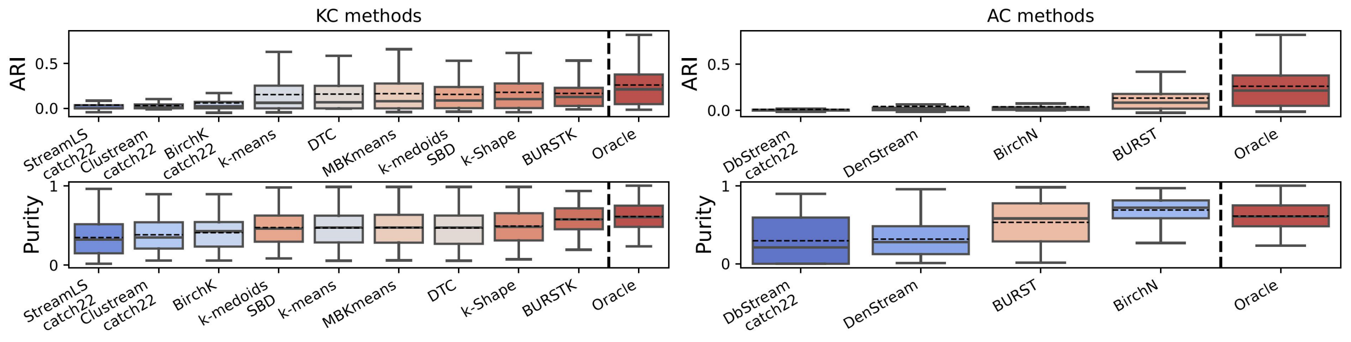

Figure 3: Box plots of ARI and Purity performance for best variant of *KC* and *AC* methods, sorted by median, along with performance of the best statically applied clustering, shown to the right of the dashed vertical line.

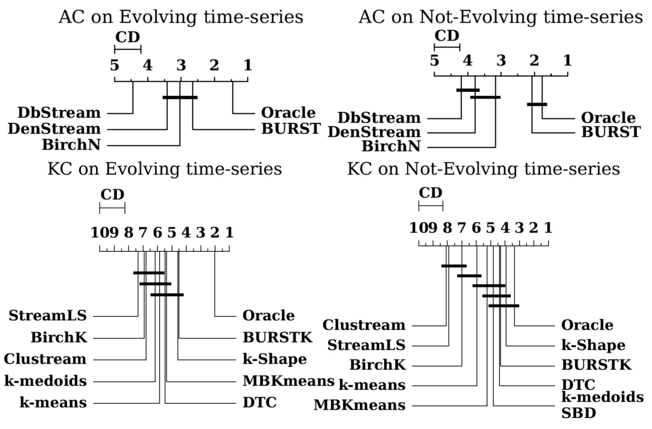

Figure 4: Ranking of *AC* (top) and *KC* (bottom) methods (ARI).

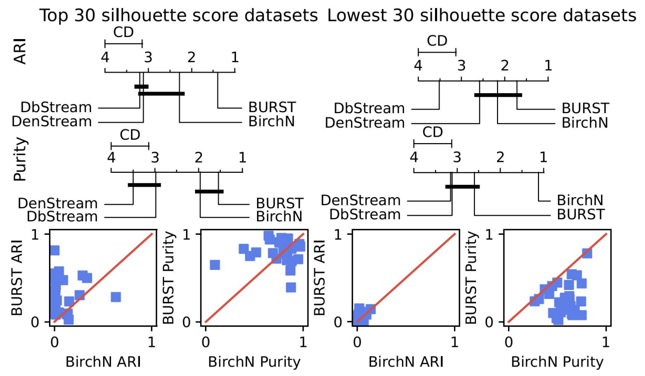

Figure 5: Cluster compactness impact.

Additionally, we conducted an ablation study to assess the contribution of each BURST module. Table 2 shows the impact of replacing key components for the top 30 datasets. Removing adaptive thresholding (replaced by a 2σ rule) or disabling AutoKC (setting k=2) while allowing the merging step to form new clusters led to performance drops in both ARI and Purity. Eliminating the merging mechanism entirely caused new clusters to form in every batch, significantly increasing the total number of clusters. This led to a sharp ARI drop (-45%), while the slight Purity gain is due to over-segmentation (see Section 5.2).

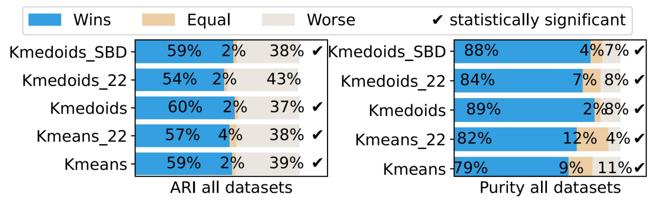

Figure 6: BURST performance when applied with variants of *k*-means and *k*-medoids versus the variant without BURST.

6 BURST GENERALIZATION

To demonstrate the robustness of the BURST framework, we implement Algorithm 1 using *k*-means and *k*-medoids as baseline clustering models. An illustration of the overall process is provided in Figure 2. BURST initializes clusters using AutoKC on the first batch. Then, real-time cluster assignment is performed based on the nearest-center principle. As new batches form, clusters are recalculated using AutoKC and merged with previously identified clusters. BURST performs two types of merging: AutoKC inner merging ($KC.innerMerge()$ in Algorithm 3) merges clusters within same batch, while cluster update merging ($Match\_Clusters()$ in Algorithm 2) combines clusters from the previous and current batch.

Implementing the merging steps requires modifications to the BURST meta-data structure. For *k*-means, $KC.innerMerge()$ recalculates merged cluster centers as the mean of their time-series. Since previous batch data are unavailable, $Match\_Clusters()$ uses meta-data instead of storing all time-series, maintaining only cluster counts to compute merged centers as their weighted sum.

For *k*-medoids, $KC.innerMerge()$ updates merged cluster centers by selecting the most central time-series of the merged clusters. Then, $Match\_Clusters()$ relies on meta-data, storing each cluster center's centrality as $\frac{|CL_j|}{avg(D_{all})}$, where $|CL_j|$ is the cluster size and $avg(D_{all})$ is the average distance from the center to all time-series in the cluster. During the merging process, the center with the highest centrality is selected as the center of the merged cluster.

Table 2: Impact of replacing BURST modules. Arrows indicate statistically significant changes.

BURST Components	ARI	Purity
No adaptive thresholding	-9 %	↓ -14 %
Without AutoKC	-8%	↓ -19%
Without Merging	↓ -45%	6%

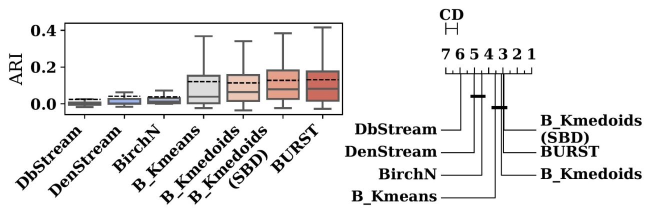

Figure 7: Performance of BURST (*B_*) strategy (with AutoKC) versus *AC* clustering algorithms.

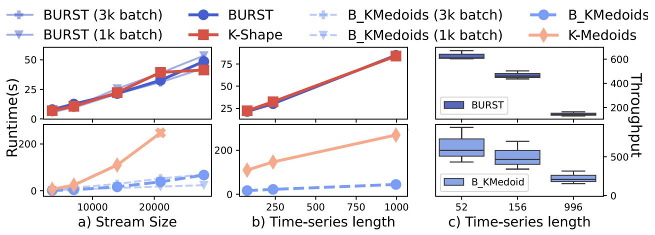

Figure 8: BURST runtime scalability.

6.1 BURST with k-means and k-medoids

We assess the performance of BURST using k-means and k-medoids as baseline methods, in both *KC* and *AC* settings. Figure 6 shows the performance improvement (in the *KC* setting) in terms of ARI and Purity when the BURST framework is applied to baseline variants. In all cases, BURST results in more wins (i.e., better performance) than the baseline methods. Moreover, in all but one case, BURST leads to statistically significantly better performance. To assess whether BURST results in statistically significant improvements, we perform pairwise comparisons (between performance with and without BURST) using the Wilcoxon test [59], testing whether the application of BURST yields significantly better results.

We also evaluate the two variants in the *AC* setting, incorporating the AutoKC algorithm. Figure 7 shows the performance of k-medoids and k-means variants with BURST (*B_*), compared to *AC* methods, using both box plots and a critical diagram. The results highlight the superiority of the BURST framework over traditional methods. Specifically, BURST with k-means and k-medoids performs similarly to BURST employing k-Shape (referred to simply as BURST), where all BURST variants ranking at top, without statistical difference between them.

6.2 Scalability

Figure 8 illustrates the runtime performance of BURST combined with two baseline methods: k-Shape (linear time complexity) and k-medoids (quadratic time complexity). We evaluate scalability across three setups: (1) clustering each batch independently, (2) applying the baseline statically to the full stream, and (3) using BURST.

BURST's runtime depends on the baseline's complexity, with added overhead from computing metadata, merging clusters, and executing the AutoKC algorithm. The main overhead stems from computing cluster centrality, which scales linearly with batch size. Figure 8 a), presents the runtime comparison between BURST along with baseline methods (k-Shape in the upper panel and k-medoids in the lower panel) across varying dataset sizes. For BURST, we report results with both fixed batch sizes (1k and 3k) and a dynamic batch size set to 10% of the dataset size, as used in our main evaluation. Column b) displays the runtime of each method when the dataset size and batch size are fixed, but the time-series length varies. The results demonstrate that, for linear-complexity clustering methods like k-Shape, BURST introduces only a modest overhead per batch, leading to overall runtimes that are similar to the static application. In contrast, for clustering algorithms with quadratic complexity, such as k-medoids, BURST can actually improve efficiency. This is because BURST processes data in smaller batches, thereby reducing the sample size per iteration. Finally, column c) shows the average throughput of BURST using k-Shape and k-medoids, again across varying time series lengths. All experiments ran on a single machine equipped with 16 GB of RAM and an Intel Core i5 processor with four cores running at 3.20 GHz. In that setting, BURST handles approximately 630 time series per second for a series length of 52, around 470 per second for length 159, and about 142 per second for length 996. The *B_KMedoids* method processes 630 time series per second for length 52, 500 for length 156, and 225 for length 996.

7 CONCLUSION AND FUTURE WORK

We address the problem of identifying clusters in evolving time series. While robust solutions exist for static time-series clustering, the streaming setting requires both incremental approaches and adaptability to evolving data. To this end, we propose BURST, which builds upon partition-based clustering algorithms and increases their effectiveness in real-time applications (and thus avoids re-inventing the wheel) through accounting for cluster merging, appearance and deletion without the need for setting a predefined number of clusters. Technically, it introduces a novel combination of hierarchical clustering and adaptive thresholding in a model-agnostic manner. Our approach demonstrates state-of-the-art performance in discovering clusters in time series hile remaining scalable. To further cover real-worlds cases, in the future we plan to extend the framework to multi-variate time-series, through incorporating distance functions specifically designed for these cases.

ACKNOWLEDGMENTS

The second author was co-funded by SECUR-EU. The SECUR-EU project funded under Grant Agreement 101128029 is supported by the European Cybersecurity Competence Center.

REFERENCES

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2003. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29* (Berlin, Germany) *(VLDB '03)*. VLDB Endowment, 81–92.

[2] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. Time-series clustering – A decade review. *Information Systems* 53 (2015), 16–38. https://doi.org/10.1016/j.is.2015.04.007

[3] Sahar Ahsani, Morteza Yousef Sanati, and Muharram Mansoorizadeh. 2023. DynamicCluStream: An algorithm Based on CluStream to Improve Clustering Quality. *International Journal of Web Research* 6, 2 (2023), 77–87.

[4] Peter Bloomfield. 2000. *Fourier Analysis of Time Series: An Introduction.* Wiley. https://doi.org/10.1002/0471722235

[5] Paul Boniol, Ashwin K Krishna, Marine Bruel, Qinghua Liu, Mingyi Huang, Themis Palpanas, Ruey S Tsay, Aaron Elmore, Michael J Franklin, and John Paparrizos. 2025. VUS: effective and efficient accuracy measures for time-series anomaly detection. *The VLDB Journal* 34, 3 (2025), 32.

[6] Paul Boniol, Qinghua Liu, Mingyi Huang, Themis Palpanas, and John Paparrizos. 2024. Dive into Time-Series Anomaly Detection: A Decade Review. *arXiv preprint arXiv:2412.20512* (2024).

[7] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J Franklin. 2021. Sand in action: subsequence anomaly detection for streams. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2867–2870.

[8] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. 2021. SAND: streaming subsequence anomaly detection. *Proc. VLDB Endow.* 14, 10 (jun 2021), 1717–1729. https://doi.org/10.14778/3467861.3467863

[9] Athman Bouguettaya, Qi Yu, Xumin Liu, Xiangmin Zhou, and Andy Song. 2015. Efficient agglomerative hierarchical clustering. *Expert Systems with Applications* 42, 5 (2015), 2785–2797. https://doi.org/10.1016/j.eswa.2014.09.054

[10] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. 2006. Density-Based Clustering over an Evolving Data Stream with Noise. In *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*, Joydeep Ghosh, Diane Lambert, David B. Skillicorn, and Jaideep Srivastava (Eds.). SIAM, 328–339. https://doi.org/10.1137/1.9781611972764.29

[11] Yixin Chen and Li Tu. 2007. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Jose, California, USA) *(KDD '07)*. Association for Computing Machinery, New York, NY, USA, 133–142. https://doi.org/10.1145/1281192.1281210

[12] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1977.tb01600.x

[14] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Seattle, WA, USA) *(KDD '04)*. Association for Computing Machinery, New York, NY, USA, 551–556. https://doi.org/10.1145/1014052.1014118

[15] Jens E d'Hondt, Haojun Li, Fan Yang, Odysseas Papapetrou, and John Paparrizos. 2025. A Structured Study of Multivariate Time-Series Distance Measures. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–29.

[16] Alberto Diez, Nguyen Lu Dang Khoa, Mehrisadat Makki Alamdari, Yang Wang, Fang Chen, and Peter Runcie. 2016. A clustering approach for structural health monitoring on bridges. *Journal of Civil Structural Health Monitoring* 6, 3 (2016), 429–445.

[17] Adam Dziedzic*, John Paparrizos* (*equal contribution), Sanjay Krishnan, Aaron Elmore, and Michael Franklin. 2019. Band-limited training and inference for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 1745–1754.

[18] Jens E d'Hondt, Odysseas Papapetrou, and John Paparrizos. 2024. Beyond the Dimensions: A Structured Evaluation of Multivariate Time Series Distance Measures. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 107–112.

[19] Lisa Ehrlinger and Wolfram Wöß. 2022. A survey of data quality measurement and monitoring tools. *Frontiers in big data* 5 (2022), 850611.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) *(KDD'96)*. AAAI Press, 226–231.

[21] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 37 pages. https://doi.org/10.1145/2523813

[22] Apostolos Giannoulidis, Anastasios Gounaris, Athanasios Naskos, Nikodimos Nikolaidis, and Daniel Caljouw. 2025. Engineering and evaluating an unsupervised predictive maintenance solution: a cold-forming press case-study. *J. Intell. Manuf.* 36, 3 (2025), 2121–2139. https://doi.org/10.1007/S10845-024-02352-Z

[23] Michael Hahsler and Matthew Bolaños. 2016. Clustering data streams based on shared density between micro-clusters. *IEEE transactions on knowledge and data engineering* 28, 6 (2016), 1449–1461.

[24] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2 (1985), 193–218.

[25] Leonard Kaufman and Peter J. Rousseeuw. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley. https://doi.org/10.1002/9780470316801

[26] Peter Laurinec and Mária Lucká. 2019. Interpretable multiple data streams clustering with clipped streams representation for the improvement of electricity consumption forecasting. *Data Mining and Knowledge Discovery* 33, 2 (01 Mar 2019), 413–445. https://doi.org/10.1007/s10618-018-0598-2

[27] Peng Lin, Bang Zhang, Yi Wang, Zhidong Li, Bin Li, Yang Wang, and Fang Chen. 2015. Data Driven Water Pipe Failure Prediction: A Bayesian Nonparametric Approach. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 193–202. https://doi.org/10.1145/2806416.2806509

[28] Qinghua Liu, Paul Boniol, Themis Palpanas, and John Paparrizos. 2024. Time-Series Anomaly Detection: Overview and New Trends. *Proceedings of the VLDB Endowment (PVLDB)* 17, 12 (2024), 4229–4232.

[29] Qinghua Liu, Seunghak Lee, and John Paparrizos. 2025. TSB-AutoAD: Towards Automated Solutions for Time-Series Anomaly Detection. *PVLDB* 18, 11 (2025), 4364–4379.

[30] Qinghua Liu and John Paparrizos. 2024. The Elephant in the Room: Towards A Reliable Time-Series Anomaly Detection Benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems.*

[31] Carl H Lubba, Sarab S Sethi, Philip Knaute, Simon R Schultz, Ben D Fulcher, and Nick S Jones. 2019. catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1821–1852.

[32] J MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press.* University of California Press, 281–297.

[33] Naveen Sai Madiraju. 2018. *Deep temporal clustering: Fully unsupervised learning of time-domain features.* Master's thesis. Arizona State University.

[34] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani (Eds.). MIT Press, 849–856. https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html

[35] Frank Nielsen. 2016. *Hierarchical Clustering.* Springer International Publishing, Cham, 195–211. https://doi.org/10.1007/978-3-319-21903-5_8

[36] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha. 2002. Streaming-Data Algorithms for High-Quality Clustering. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, Rakesh Agrawal and Klaus R. Dittrich (Eds.). IEEE Computer Society, 685–694. https://doi.org/10.1109/ICDE.2002.994785

[37] Ioannis Paparrizos. 2018. *Fast, scalable, and accurate algorithms for time-series analysis.* Ph.D. Dissertation. Columbia University.

[38] John Paparrizos, Paul Boniol, Qinghua Liu, and Themis Palpanas. 2025. Advances in Time-Series Anomaly Detection: Algorithms, Benchmarks, and Evaluation Measures. In *SIGKDD.*

[39] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S Tsay, Aaron Elmore, and Michael J Franklin. 2022. Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2774–2787.

[40] John Paparrizos and Michael J Franklin. 2019. GRAIL: efficient time-series representation learning. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1762–1777.

[41] John Paparrizos and Luis Gravano. 2016. k-Shape: Efficient and Accurate Clustering of Time Series. *SIGMOD Rec.* 45, 1 (jun 2016), 69–76. https://doi.org/10.1145/2949741.2949758

[42] John Paparrizos and Luis Gravano. 2017. Fast and Accurate Time-Series Clustering. *ACM Transactions on Database Systems (TODS)* 42, 2 (2017), 1–49.

[43] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. 2022. TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1697–1711.

[44] John Paparrizos, Haojun Li, Fan Yang, Kaize Wu, Jens E d'Hondt, and Odysseas Papapetrou. 2024. A Survey on Time-Series Distance Measures. *arXiv preprint arXiv:2412.20574* (2024).

[45] John Paparrizos, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2020. Debunking Four Long-Standing Misconceptions of Time-Series Distance Measures. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery,

New York, NY, USA, 1887–1905. https://doi.org/10.1145/3318464.3389760

[46] John Paparrizos, Chunwei Liu, Aaron J Elmore, and Michael J Franklin. 2023. Querying Time-Series Data: A Comprehensive Comparison of Distance Measures. *Data Engineering* (2023), 69.

[47] John Paparrizos and Sai Prasanna Teja Reddy. 2023. Odyssey: An Engine Enabling the Time-Series Clustering Journey. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 4066–4069. https://doi.org/10.14778/3611540.3611622

[48] John Paparrizos and Sai Prasanna Teja Reddy. 2025. Time-Series Clustering: A Comprehensive Study of Data Mining, Machine Learning, and Deep Learning Methods. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4380–4395.

[49] John Paparrizos, Kaize Wu, Aaron Elmore, Christos Faloutsos, and Michael J Franklin. 2023. Accelerating Similarity Search for Elastic Measures: A Study and New Generalization of Lower Bounding Distances. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2019–2032.

[50] John Paparrizos, Fan Yang, and Haojun Li. 2024. Bridging the Gap: A Decade Review of Time-Series Clustering Methods. *arXiv preprint arXiv:2412.20582* (2024).

[51] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* 36, 2 (2009), 3336–3341.

[52] Dan Pelleg and Andrew W. Moore. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 727–734.

[53] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.

[54] D. Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 1177–1178. https://doi.org/10.1145/1772690.1772862

[55] Emmanouil Sylligardos, Paul Boniol, John Paparrizos, Panos Trahanias, and Themis Palpanas. 2023. Choose Wisely: An Extensive Evaluation of Model Selection for Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3418–3432.

[56] Andreas Theissler, Judith Pérez-Velázquez, Marcel Kettelgerdes, and Gordon Elger. 2021. Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability engineering & system safety* 215 (2021), 107864.

[57] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63, 2 (2001), 411–423. https://doi.org/10.1111/1467-9868.00293 arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00293

[58] Xin Wang, Zhengru Wang, Zhenyu Wu, Shuhao Zhang, Xuanhua Shi, and Li Lu. 2023. Data Stream Clustering: An In-depth Empirical Study. *Proc. ACM Manag. Data* 1, 2, Article 162 (June 2023), 26 pages. https://doi.org/10.1145/3589307

[59] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968

[60] Fan Yang and John Paparrizos. 2025. SPARTAN: Data-Adaptive Symbolic Time-Series Approximation. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–30.

[61] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. 2019. Outlier Detection: How to Threshold Outlier Scores?. In *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing* (Sanya, China) *(AIIPCC '19)*. Association for Computing Machinery, New York, NY, USA, Article 37, 6 pages. https://doi.org/10.1145/3371425.3371427

[62] Antoine Zambelli. 2016. A data-driven approach to estimating the number of clusters in hierarchical clustering. *F1000Research* 5 (2016), 2809. https://doi.org/10.12688/F1000RESEARCH.10103.1

[63] Hui Zhang, Tu Bao Ho, Yang Zhang, and Mao Song Lin. 2006. Unsupervised Feature Extraction for Time Series Clustering Using Orthogonal Wavelet Transform. *Informatica (Slovenia)* 30, 3 (2006), 305–319. http://www.informatica.si/index.php/informatica/article/view/98

[64] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *SIGMOD Rec.* 25, 2 (June 1996), 103–114. https://doi.org/10.1145/235968.233324

[65] Ying Zhao and George Karypis. 2004. Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering. *Machine Learning* 55, 3 (01 Jun 2004), 311–331.

[66] Shuxin Zhong, William Yubeaton, Wenjun Lyu, Guang Wang, Desheng Zhang, and Yu Yang. 2023. RLIFE: Remaining Lifespan Prediction for E-Scooters. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (, Birmingham, United Kingdom,) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 3544–3553. https://doi.org/10.1145/3583780.3615037

[67] Jiaqi Zhu, Shaofeng Cai, Fang Deng, Beng Chin Ooi, and Wenqiao Zhang. 2023. METER: A Dynamic Concept Adaptation Framework for Online Anomaly Detection. *Proc. VLDB Endow.* 17, 4 (Dec. 2023), 794–807. https://doi.org/10.14778/3636218.3636233