Online Event Detection in Streaming Time Series: Novel Metrics and Practical Insights

Janio Lima
CEFET/RJ & Petrobras
janio.lima@aluno.cefet-rj.br

Lucas Giusti Tavares
CEFET/RJ
lucas.tavares@aluno.cefet-rj.br

Esther Pacitti
INRIA & University of Montpellier
esther.pacitti@inria.fr

João Eduardo Ferreira
USP
jef@ime.usp.br

Ismael Santos
Petrobras
ismaelh@petrobras.com.br

Isabela Guimarães Siqueira
Petrobras
isabela.siqueira@petrobras.com.br

Diego Carvalho
CEFET/RJ
d.carvalho@ieee.org

Fabio Porto
LNCC
fporto@lncc.br

Rafaelli Coutinho
CEFET/RJ
rafaelli.coutinho@cefet-rj.br

Eduardo Ogasawara
CEFET/RJ
eogasawara@ieee.org

2024 International Joint Conference on Neural Networks (IJCNN) | 979-8-3503-5931-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IJCNN60899.2024.10650809

***Abstract*—Online event detection in streaming time series is a critical task with applications across various domains. For example, the right-on-time event detection for control systems is a key for correctly addressing the issues related to the events. However, events may not be identified right after their occurrence. Depending on the monitoring solution, a time difference may exist between the event's occurrence and detection. This problem raises research questions regarding the study of such a temporal gap. The paper introduces novel metrics (detection probability and detection lag) to address these questions. It explores the impact of configurable batches on detection performance. The experimental evaluation of diverse datasets reveals nuanced insights into the interplay between batch parameters, detection accuracy, and computational performance.**

I. Introduction

Time series events are characterized by the occurrence of a significant change in the behavior of a time series at a certain point or time interval [1, 2, 3]. Such events are commonly identified using change point or anomaly detection methods [4, 5, 6, 7]. Event detection methods are either offline or online [8, 9, 10, 11]. When offline, they rely on prior access to all historical data. When online, they have to detect occurrences by monitoring the streaming of process data while it is being generated [12, 9, 13].

Many control systems demand right-on-time detection to enable undesirable situations to be addressed promptly. In this way, continuously monitoring streaming time series raises an important fact. An observation that characterizes an event might not be detected when the observation appears in the time series. New observations might be appended before event detection methods discover them [14, 15, 16]. Thus, if an event occurs in time t but is only detected by an online detection method at time $t + k$, the detection lag is k.

Considering the detection lag problem, it is possible to raise the following research questions: How can one evaluate the elapsed time between when an observation in the time series is read and its detection as an event? Is it possible to assess the behavior of methods throughout the streaming process? How can information about the behavior of methods contribute to identifying changes in the streaming time series?

To explore these questions and enhance online event detection in time series, this paper introduces new metrics for analyzing the performance of event detection methods over streaming time series. The presented metrics enable assessing the probability of a detection corresponding to an event and lag in detections. Since existing studies do not correctly address these aspects, these are important contributions of this paper.

In addition to this introduction, this paper is divided into five more sections. Section II presents the theoretical framework for event detection. Section III discusses related works. Section IV introduces the general framework with the proposed probability and lag analysis in event detection. Section V discusses the results of the experiments, and Section VI presents the research conclusions.

II. Background

This section introduces fundamental event detection and subsequence analysis concepts in time series.

A. Event Detection

Event detection can be based on predicted values, where methods include a step for predicting the next values in the time series. Events are detected when there is a discrepancy between the actual value of a new observation and predicted values. Alternatively, events can be detected based on boundaries, defining upper and lower limits beyond which each new streaming observation is marked as an event [17, 18, 16, 7, 19].

There are two main modes for online event detection in streaming time series: static and dynamic. In static mode, the model detects events in the streaming time series using a pre-trained model with historical data. In dynamic mode, training is done once a warm-up size for the time series is achieved.

As new observations are received via streaming, the model is retrained incrementally [15].

B. Subsequence Analysis in Time Series

To analyze local properties in a time series X, subsequences of size p can be extracted as described by Equation 1. The sliding windows technique involves extracting subsequences that traverse the entire time series. For this, a set of subsequences of the same size is generated, represented as a matrix A containing all the time series X subsequences, as defined in Equation 2. The sliding window enables continuous analyses of the local properties of time series [20].

$$seq_{p,i}(X) =< x_i, x_{i+1}, \cdots, x_{i+(p-1)} >, \quad |seq_{p,i}(X)| = p, 1 \leq i \leq |X| - p \tag{1}$$

$$\forall a_i \in A, a_i = seq_{p,i}(X) \tag{2}$$

Event detection methods such as *Exponentially Weighted Moving Average* (EWMA) [21], and *Forward and Backward Inertial Anomaly Detector* (FBIAD) [22] use sliding windows either integrated or as an intermediate preprocessing [23]. Naturally, considering a scenario in which a new observation is appended one at a time and an event detection method is immediately triggered, the concept of sliding windows is seamlessly prepared for online event detection. For example, Zeileis et al. [24] present an R package called *strucchange* to detect change points in streaming time series through sliding windows.

However, when the data arrives very fast, analyzing all sliding windows each time a single observation arrives is not feasible. In these cases, grouping the arrival of new observations into batches makes sense. Although the term batch processing is typically associated with offline processing, even streaming data processing can occur by sending and receiving individual small batches [9, 5, 25]. For example, batches are used in the *Spark streaming* tool. The tool uses micro-batches, *i.e.*, small batches of data to be processed [5]. A similar intuition for dividing the time series into batches was explored for streaming processing to identify concept changes by Iwashita and Papa [26], Giusti et al. [27].

How batches are configured is a key for online event detection. The number of batches preserved in memory is also a parameter commonly studied. It gives the flexibility of enabling a connection between the two scenarios previously indicated. The case in which we define a batch of size 1, preserving m batches in memory, provides the same context of having sliding windows of size m. So, these concepts are not antagonistic.

III. Related Work

This section discusses online event methods and tools and ways of comparing them. Subsection III-A presents works that propose event detection methods and tools. Subsection III-B presents works comparing methods. Finally, Subsection III-C wraps up related work and presents identified research gaps.

A. Event Detection Methods in Streaming Time Series

Various methods have been developed for offline event detection, but works such as Talagala et al. [16] and Ariyaluran Habeeb et al. [15] indicate the challenges of these methods in handling streaming time series. However, there is no definitive evidence of the non-applicability of methods initially designed for offline detection in the streaming scenario. Considering their widespread use, they should not be overlooked [28, 29, 9, 30]. Some examples include *Holt-Winters* (HW) [31], FBIAD [22], *Generalized Autoregressive Conditional Heteroskedasticity* (GARCH) [32], EWMA [1], and *ChangeFinder* (CF) [33, 30]. There are also certain tools with hybrid applications developed for offline and online detection, including *Strucchange* [24] and *k-Nearest Neighbors - Conformal Anomaly Detector* (KNN-CAD) [17].

Rettig et al. [5] introduced an online event detection method targeting gradual and abrupt changes. Gradual changes in the data stream are identified using relative entropy, whereas abrupt changes with Pearson correlation [5]. Ahmad et al. [17] proposed a method for event detection in time series streaming. The Hierarchical Temporal Memory (HTM) uses machine learning to model the distribution of the time series and compares new observations with model predictions, marking anomaly events based on the discrepancy between actual and predicted values [17, 31]. Zhang et al. [7] presented the MF-*stacked* LSTM-EWMA method that employs a prediction step and a detection step based on the divergence between actual streaming observations and model predictions. The method is divided into three steps: (i) identifying obvious anomalies, (ii) predicting the next observations in the time series, and (iii) detecting events.

Ren et al. [19] implemented a service called Microsoft Anomaly Detector (MAD) on the Microsoft Azure cloud platform. MAD used the SR-CNN algorithm, a combination of the SR algorithm and a CNN [19]. SR performs a time series transformation, generating a saliency map representation where extreme values are accentuated. At the same time, CNN uses this transformed version of the time series to learn boundaries to determine event occurrences.

B. Comparative Studies of Methods

In addition to works focused on proposing methods for event detection, efforts were made to identify studies comparing these methods. Salles et al. [29] developed the *Harbinger* framework, which provides functions for event detection, evaluation, visualization, method combination, and comparison of detections. *Harbinger* enables the implementation and combination of different event detection methods but executes them offline.

Ahmad et al. [17] provides a framework named Numenta, the *Numenta Anomaly Benchmark* (NAB) dataset, and the NAB Score metric. Several methods were compared for online detection, and the three methods with the best-published performances were HTM, *Relative Entropy*, and *Twitter ADVec* [17]. The methods were compared using the NAB Score metric and microsecond latency time. Hasani [6] analyzes online

event detection methods and compares performance, execution time, and CPU usage.

Belacel et al. [11] present a proof of concept for using methods existing in the *Scikit-Multiflow* and *River* packages integrated into the *Kafka* streaming platform. The main contributions of the work relate to proposing a model for integrating methods with Kafka and reducing execution time in streaming scenarios compared to offline detection. In the analysis of comparative method studies, the use of traditional metrics for classification tasks [34, 29] and computational cost [6] is observed. Some metrics adapted for the time series context are also used [17, 30].

C. Summary

A synthesis of the analysis of related works is presented in Table I. An important aspect of identifying gaps in the literature is that most works do not provide an analysis of batch configuration during streaming processing and do not address a deeper discussion on evaluation metrics. Table I shows that using detection metrics adapted from classification tasks is common. While evaluating streaming, the focus is on computational performance, such as execution time, latency, and CPU usage. This paper explores these limitations through the methodology presented in the next section.

TABLE I
METRICS FOR COMPARISON IN STREAMING DETECTION

Work	Metrics
Numenta [17]	Latency
Harbinger [29]	No
MAD [19]	No
Strucchange [24]	No
Rettig et al. [5]	No
Hasani [6]	Execution Time, CPU Usage
Belacel et al. [11]	No

IV. METHODS

This section discusses the general scenario for online time series event detection in streaming. It also provides new metrics to support the evaluation of detectors. Section IV-A provides the basic formalization. Section IV-B presents how detections in streaming scenarios are carried out. Sections IV-C and IV-D formalize *Detection Probability* (DP) and *Detection Lag* (DL), respectively.

A. Formalization

Given a time series X of size n, it is partitioned into batches of size s. The number of batches of X is defined as $\lceil \frac{n}{s} \rceil$. Thus, each batch b_j ($1 \leq j \leq \lceil \frac{n}{s} \rceil$) has a sequence of s observations $< x_{(j-1)\cdot s+1}, \ldots, x_{j\cdot s} >$.

For streaming processing, w is the number of batches to fill in memory (*warm-up*) to start the event detection process. Besides, m is the number of batches to preserve in memory. In steady state, for full memory (old batches are not discarded) $m \geq w$, whereas for partial memory (old batches are discarded) $m = w$.

B. Online Event Detection Process

We established a configurable batch for event detection workflow to conduct detections and develop probability and lag analysis metrics. The workflow is presented in Figure 1. It receives a time series X and additional parameters Pr for streaming event detection on the time series. These additional configuration parameters are streaming process parameters s, w, and m and the applied method M for detection with its respective hyperparameters. The workflow continues executing by sliding along the time series in batches of size s iteratively until the entire time series is traversed.

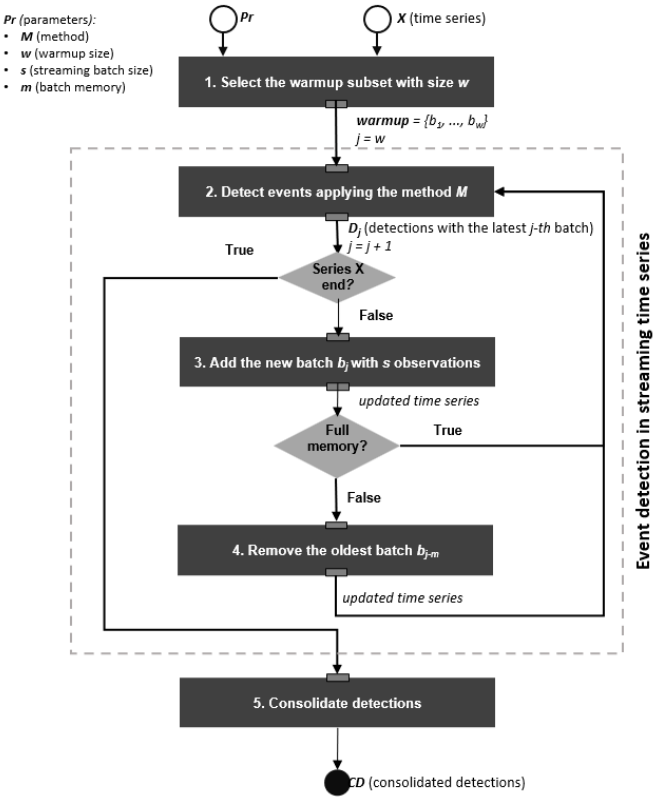

Fig. 1. Workflow for event detection in streaming time series

This process of analyzing the series iteratively through batches is described in the following steps that detail Figure 1. Initially, the warm-up subset is selected (Step 1). Then, the method M with the provided hyperparameters for event detection is applied in the warm-up (Step 2). This step generates a set of event detections with the latest j-th batch, denoted by D_j.

After detection is completed, if the series has not yet reached the end, a new batch b_j is included (Step 3). If the configurable batch is defined as full memory, *i.e.*, $m = 0$, the workflow maintains the memory of old batches. However, when $m \geq 1$ (partial memory), the oldest batch b_{j-m+1} is removed (Step 4) when a new batch b_{j+1} is appended in a steady state. At this point, j is incremented in one unit.

The process is repeated from step 2, i.e., method M is applied again to detect events, generating a new set of events D_j. Finally, the detection results from each stream processing iteration (D_j) are consolidated in the CD set (Step 5) to measure the events detected and their detection times correctly.

The goal of using configurable batches with full or partial memory approaches aims to balance the accuracy of the detection methods and the computational cost of analyzing the time series in streaming. In both cases, the batches are added continuously. Hence, the methods use the in-memory batches to learn the expected behavior of the time series. With access to observations, methods are applied to detect events.

In full-memory detection, batches are added continuously, and the time series is analyzed cumulatively. This way, there is more information for use by the methods that can help increase accuracy. However, it is natural for a gradual increase in computational cost, and old observations might interfere too much with event detection.

With partial memory, old batches are removed as new batches are added. Thus, only the most recent batches equivalent to the memory size m defined in the execution parameters remain in memory. It is expected to evaluate whether this option generates the opposite effect of full-memory preservation, *i.e.*, with lower computational cost, greater influence of recent observations on event detection, and their effects on detection accuracy.

C. Detection Probability

Consider a time series X; the assessment of the DP requires the mapping of each observation x_i while present in batches during processing. Assuming that x_i appeared in batch b_j, such $j = \lceil \frac{i}{s} \rceil$, x_i is available for time series detection $bf(x_i)$ times. In steady state, Equation 3 characterizes the batch frequency for an observation x_i, considering partial and full memory cases.

$$bf(x_i) = \begin{cases} m, & \text{for the partial memory case} \\ \lceil \frac{n}{s} \rceil, & \text{for the full memory case.} \end{cases} \tag{3}$$

Let B_i be set of the most recent batches an observation x_i is presented in the batch memory, such that $B_i = \{b_j, \ldots, b_{j+bf(x_i)}\}$, a detector M has $bf(x_i)$ evaluations to check if x_i is an event or not. It is possible to define as a definition frequency $df(x_i)$ as the number of times x_i was marked by M as an event. From this definition, $0 \leq df(x_i) \leq bf(x_i)$.

Under this vein, it is possible to define the detection probability $DP(x_i)$, presented by Equation 4, as the ratio between detection frequency and batch frequency for x_i.

$$DP(x_i) = \frac{df(x_i)}{bf(x_i)} \tag{4}$$

To exemplify how DP can be applied to enhance the analysis of online event detections, two hypothetical observations, x_3 and x_5 in a series X, can be assumed. Considering that when analyzing the series through a batch size 1 with the batch memory size 7 ($m = 7$), method M detects x_3 as event 2 times. In contrast, the observation x_5 is detected as event 6 times, so $df(x_3) = 2$ and $df(x_5) = 6$. Considering, furthermore, that both observations appear in 7 batches, hence $bf(x_3) = bf(x_5) = 7$. Thus, $DP(x_3) = 0.29$ and $DP(x_5) = 0.86$.

D. Detection Lag

The DL of each detection is evaluated to assess the early detection capability of DL-executed methods. For this purpose, DL is considered the lag in detection, *i.e.*, the number of batches elapsed between the first reading of the observation and the first batch in which this observation is detected as an event. Formally, considering an observation x_i in the time series X, sb_i as the start of reading this observation in batches (*start batch*), and fdb_i as the first batch in which the observation x_i is detected as an event (*first detection batch*), we have Lag_i^s for a batch size s obtained by Equation 5. The DL identifies how many batches the method took to detect an event, helping to assess the methods' early event detection capability.

$$Lag_i^s = fdb_i - sb_i \tag{5}$$

To illustrate the use and interpretation of Lag_i^s, one can consider a hypothetical observation in time 9 (x_9) for a time series of size 12. Assuming a batch size 3, the start batch sb_9 equals 3. Besides, if we consider that the detector M only found x_9 as an event in batch 4, *i.e.*, $fdb_9 = 4$. In this way, Lag_9^3 equals 1.

Considering that it is possible to configure batches with different sizes (s), it is not recommended to compare the DL obtained in batches using Equation 5 between executions of different batch sizes. As an alternative, for cases in which this type of comparison is desired, obtaining the DL value in the number of observations is possible using Equation 6.

$$Lag_i = (fdb_i - sb_i + 1) \cdot s \tag{6}$$

V. Experimental Evaluation

This section comprehensively analyzes the online event detection process and proposed novel metrics. Firstly, Section V-A introduces the experimental setup, laying the foundation for subsequent discussions. Following this, an in-depth experimental evaluation is presented to address three key aspects: (i) the comparison of various batch parameter combinations (Section V-B), (ii) an assessment of the utility of the detection probability (Section V-C), and (iii) an exploration of the detection lag associated with the methods employed (Section V-D).

A. Experimental Setup

Datasets representing a wide diversity of domain areas and time series behaviors were selected to explore the studied methods properly. These datasets include Yahoo Labs [19], Numenta [17], and RARE [35]. The datasets used in the

experiments were made available in a public repository[1] to facilitate reproducibility.

The time series contained in the Yahoo Labs, NAB, and RARE datasets are from domain areas related to technology, such as network and internet data traffic, cloud computing, and computer equipment monitoring [17, 35]. The Yahoo Labs and NAB time series are extensively used in research comparing event detection methods [6, 29, 35]. Thus, all the time series from these datasets were included in the analysis.

The selected detection methods were CF [1], FBIAD [22], ARIMA [28], GARCH [32], and LSTM [7]. This choice aims to explore a suitable diversity of event detection approaches. Regarding the types of events, the CF method detects change points, and GARCH detects volatility anomalies. In contrast, the other methods detect anomalies in general.

The parameters used for each method are displayed in Table II. These parameters represent the default values of each method as implemented in the Harbinger framework [29]. In the experiments, no exploration of different parameter values for the methods was conducted to maintain uniform behavior and vary only the parameters related to their online execution. Thus, the experiments enable a more suitable evaluation of the impact of batches in memory on detection results for each method, with different batch sizes and preservation of complete or partial data memory across batches during execution.

TABLE II
PARAMETERS PER METHOD

Method	Parameter	Value
FBIAD	sw (sliding window size)	30
ARIMA	mdl (model)	Autoarima
	$ARIMA(p, d, q)$	order(1,6,2)
CF	mdl (model)	Linear regression
	ma (moving average size)	30
GARCH	$variance.model$, $mean.model$	sGARCH: garchOrder $= (1, 1)$, armaOrder $= (1, 1)$
LSTM	$inputsize$, $epochs$, normalization	4 neurons, 10000 epochs, Global Min-Max

Since the methods employed have different operational mechanisms, many parameters do not have equivalence or similarity. However, when more than one method uses similar parameters, the values are kept the same for better comparability of results. In Table II, it is possible to verify similar parameters with equal values in the rows of sw and ma for the FBIAD and CF methods, respectively, as well as the $alpha$ parameters for the FBIAD and GARCH methods.

The parameters used in the batches are the warm-up size, batch size, and batch memory (w, s, m). The options follow a uniform proportion for the warm-up and batch size values given in the number of time series observations. For batch memory, expressed in the number of batches, the used options are complete memory of all batches and partial memory (the last three or nine batches, depending on the dataset and methods).

[1]Available at https://github.com/cefet-rj-dal/event_datasets

The selected values for batch size (s) enable observation-to-observation analysis execution (size = 1) up to larger batches with 243 observations. The range of value options was defined to enable the proper execution of methods requiring a minimum number of observations. Different value options also aimed to enable experiments with larger time series, where very small batches could make experiments too slow and computationally expensive.

Values for defining batch size (s) start at one. They follow a uniform progression of multiples of three from the second option. However, considering that the selected methods may use central tendency techniques, the first value was the one closer to 30 ($s = 27$), which is known as a sample size that it may be possible to assume normal distribution considering the Central Limit Theorem. Moreover, values below $s = 81$ might be insufficient for methods requiring more model training observations or greater values of w and m. For fairness in execution conditions and result comparison, when this occurs for a specific method, the experiment parameters are configured to start from the minimum value that works for all methods.

Regarding values for batch memory (m), in addition to exploring the complete time series by preserving the memory of all batches, attention was given to situations where memory is incompatible with the number of batches in the experiment. For example, the number of preserved batches must equal or smaller than the total number of batches analyzed. It also does not make sense for the number of batches to be very close to the total number of time series batches since the effect of both data volume for analysis and computational cost would be very close or equal to full memory execution.

The experiments were implemented using the R programming language. They were executed on a computer with an Intel i7-7820X CPU with 16 cores and 128 GB of main memory. The datasets were loaded into memory from a repository on the GitHub platform[2].

B. Batches in Memory

This subsection details the detection performance of different batch parameters and method combinations to explore the behavior of each parameter and the different methods. The average results for each metric across all datasets are presented in Table III and IV. The values were calculated by obtaining the performance average of each metric for all batch parameter combinations in each dataset.

TABLE III
AVERAGE ACCURACY FOR METHODS BY DATASETS

Method	Yahoo	Numenta	RARE
ARIMA	93.28	92.78	92.12
CF	97.01	96.15	94.31
FBIAD	97.01	82.41	93.00
GARCH	97.56	97.65	93.71
LSTM	93.32	92.84	91.44

[2]Available at https://github.com/cefet-rj-dal/event_datasets

No significant differences were found at a significance level of 5% among different combinations within each dataset for detection performance metrics. For batch execution time, executions with partial memory showed statistical significance compared to their full-memory counterparts. The batch size variable proved relevant and positively correlated with this metric. The execution time results per method and batch parameter are described in Table V.

TABLE IV
AVERAGE F1 SCORE FOR METHODS BY DATASETS

Method	Yahoo	Numenta	RARE
ARIMA	8.66	1.79	7.04
CF	1.75	2.81	1.65
FBIAD	1.31	0.78	6.97
GARCH	10.80	3.68	4.41
LSTM	7.69	1.12	3.84

TABLE V
TIME PER BATCH FOR EACH COMBINATION, METHOD, AND DATASET

Set.	Dataset	ARIMA	CF	FBIAD	GARCH	LSTM
1, 81, F	Yahoo	3.54	-	0.98	16.82	4.43
	Numenta	13.98	-	4.11	31.60	6.30
	RARE	1.85	-	5.80	41.97	4.60
1, 81, P*	Yahoo	1.56	-	0.40	13.96	5.66
	Numenta	0.91	-	0.47	12.11	8.02
	RARE	0.80	-	0.97	13.04	4.31
1, 243, F	Yahoo	12.57	2.43	3.96	62.95	11.86
	Numenta	39.32	11.83	13.51	96.14	12.56
	RARE	5.83	21.81	17.28	130.69	11.49
9, 27, F	Yahoo	1.48	0.20	0.35	5.71	2.47
	Numenta	5.06	2.27	1.44	10.77	3.78
	RARE	0.69	2.41	1.91	15.26	3.21
1, 243, P	Yahoo	7.93	1.44	2.44	51.18	11.45
	Numenta	5.77	1.50	2.56	33.80	14.26
	RARE	3.56	3.27	5.38	51.60	10.42

* The CF method does not support window size 81

Considering that most batch parameters may not influence detection accuracy according to these results, partial memory may be an effective option to reduce batch execution time. Moreover, some methods, such as the GARCH and LSTM, may be more complex than others. Nevertheless, considering the results presented and the purpose of this paper, the configuration of ($w = 1$, $s = 243$, $m = 0$) was used as the base for the upcoming experiments involving DP and DL. This combination was chosen because it was faster to execute and also provided more steps in the analysis, which are important for the metrics presented in the next sections.

C. Analysis of Detection Probability

This section analyses the impact of the detection probability on the detection performance in online event detection. Specifically, a new DP threshold is proposed based on the times each observation is detected and the total amount of times the observations are read. The purpose is to test whether the detection performance could improve with this new threshold.

The results showed that the hypothesis that non-zero thresholds may improve the detection performance is still unclear. Specifically, of the three datasets used in the experiments, only the Yahoo time series showed a significant improvement at thresholds 0.5 and 0.8 compared to 0 for both accuracy and F1. The results are presented in Table VI and VII.

The behavior of methods, expressed by the accuracy values for different thresholds, might be seen in Figure 2. The RARE dataset showed no difference between thresholds 0.5 and 0.8. Numenta only performed better in F1 for 0.5 and accuracy for 0.5 and 0.8. This dataset showed the biggest difference in accuracy when compared to 0, as illustrated by the abrupt growth in Figure 2.

TABLE VI
ACCURACY METRICS FOR EACH THRESHOLD GROUP AND DATASET

Dataset	Threshold		
	0	0.5	0.8
Yahoo	91.93	92.48	93.61
Numenta	80.90	90.50	91.55
Rare	90.76	91.09	91.50
Overall	90.73	91.29	92.31

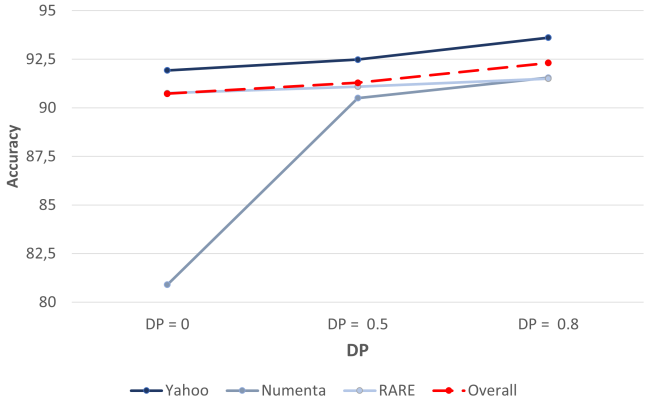

Fig. 2. Visual analysis of accuracy metrics for each threshold

TABLE VII
F1 METRICS FOR EACH THRESHOLD GROUP AND DATASET

Dataset	Threshold		
	0	0.5	0.8
Yahoo	9.78	10.25	11.32
Numenta	11.59	13.25	17.42
Rare	5.83	5.73	5.58
Overall	4.85	5.11	5.72

Regarding the performance gain in cases where it occurred, the highest gains were around 5% F_1 and 10% Accuracy. These results indicate that the proposed detection probability metric can be a good alternative to optimize the performance of univariate time series event detection. However, it is impossible to generalize to all databases, making it necessary to tune different detection probability values in each case. Specifically, the improvements achieved with the detection probability may

be explained by reducing false positives. Nevertheless, this improvement may not always hold with a high incidence of false negatives.

D. Analysis of Detection Lag

This subsection analyses the detection lag metric of all methods. Specifically, it shows how many batches each method takes to make a detection. To the best of our knowledge, these results may generate important information about the performance of the methods that were not available in any other known metric.

The results with the DL variable indicate that the values may depend significantly on the datasets. Specifically, the Yahoo dataset had a median DL of one window for all methods, indicating that real events tend to be detected faster in this dataset. Table VIII presents the DL analysis of the three datasets.

TABLE VIII
COMPARISON OF DL FOR YAHOO, NAB, AND RARE DATASETS

Parameters: $w = 1, s = 243, m = 0$.

Method	Yahoo	NAB	RARE
FBIAD	1	9	10
ARIMA	1	11	11
GARCH	1	4	6
CF	1	13	8
LSTM	1	5	10

Values: Median for all time series in each dataset

When there are no differences in this aspect, selecting the best method can be based on the balance between detection accuracy and execution time. However, as in the NAB and RARE time series, where DL values are quite distinct between methods, it is necessary to evaluate which methods achieve the best combination of accuracy, processing time, and lower lags in detections.

For NAB, a high value is observed for the median of DL, mainly due to some time series whose median lags are very high, all with $Lag_i^s > 10$. However, in the time series that originated the median calculation, the DL generally was close to $Lag_i^s = 3$. With such differences, as in NAB, the importance of evaluating the joint aspects of accuracy, time, and mentioned lag is emphasized.

The RARE dataset has peculiar characteristics, such as many time series without changes over time, which are considered anomalies according to the labels. There are also punctual changes in some time series intervals, without the labels of events coinciding with the moments when the greatest changes occur. Thus, many false negatives exist in detections and high lags, as seen in Table VIII, where only two methods have a median $Lag_i^s \leq 10$.

Among the three technology datasets, NAB has the greatest difference between Lag_i^s values. As an exercise to show the intuition of balancing the available metrics in the detection, it is noted that GARCH and LSTM have better Lag_i^s results.

Table IX confirms a difference in DL values with statistical significance between GARCH and LSTM. The same test was performed for the DL and accuracy metrics for a complete evaluation. This test compares all results and generates a value called the effect size, the magnitude of the size of the difference, which is classified on a scale as nonexistent, small, moderate, or large. In this comparison, only the difference in DL was classified as small. At the same time, accuracy has differences that are classified as large.

TABLE IX
COMPARISON BETWEEN GARCH AND LSTM USING WILCOX EFFECT SIZE STATISTICAL TEST IN NAB DATASET

Parameters: $w = 1, s = 243, m = 0$

Metric	Difference	*Effect size*	Magnitude
DL	Yes	0.054	Small
Accuracy	Yes	0.728	Large

Considering real-world scenarios, it is important to evaluate whether DL obtained by detection methods is appropriate for response time reaction by domain experts. For example, in an oil drilling and exploration [36] context, 15 minutes could be the right time to stop oil production in case of an incident. If DL could lead to an elapsed time higher than the time constraint, novel detection methods should be explored.

The different usage scenarios also deepen the importance of information on the behavior of streaming methods provided by DL and DP metrics. As demonstrated, especially in exploring the NAB time series, this information on the behavior of detections over streaming enriches the foundations for the proper selection of methods. According to the authors' best knowledge, this type of detail about the streaming of time series is a unique characteristic of this paper and reinforces its relevance.

VI. CONCLUSION

This paper proposed a comprehensive approach to online event detection in streaming time series. It introduces novel metrics for evaluating detection performance. The configurable batch methodology was formalized, enabling a balance between accuracy and computational cost.

The new metrics, Detection Probability (DP) and Detection Lag (DL), provide insights into the behavior of detection methods during streaming processing. DP measures the probability of detecting an event at each observation, and DL evaluates the lag between observing an event and its detection.

The experimental evaluation covered diverse datasets and methods. The analysis explored different batch parameter combinations, revealing that, in general, the choice of batch parameters does not significantly affect detection accuracy. However, partial memory configurations clarify the potential to reduce batch execution time without compromising accuracy.

The insights provided by DP and DL metrics contribute to a more informed decision-making process when deploying online event detection systems. In future work, using these metrics to evaluate event detection methods can be further explored to improve detection performance and evaluate whether the DL is acceptable for the specific problem.

ACKNOWLEDGMENT

The authors thank CNPq, CAPES, and FAPERJ for partially sponsoring this research.

REFERENCES

[1] V. Guralnik and J. Srivastava, "Event Detection from Time Series Data," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 33–42.

[2] L. Perelman, J. Arad, M. Housh, and A. Ostfeld, "Event detection in water distribution systems from multivariate water quality time series," *Environmental Science and Technology*, vol. 46, no. 15, pp. 8212 – 8219, 2012.

[3] L. Zhao, "Event Prediction in the Big Data Era: A Systematic Survey," *ACM Computing Surveys*, vol. 54, no. 5, 2021.

[4] A. Gensler and B. Sick, "Performing event detection in time series with SwiftEvent: an algorithm with supervised learning of detection criteria," *Pattern Analysis and Applications*, vol. 21, no. 2, pp. 543–562, 2018.

[5] L. Rettig, M. Khayati, P. Cudre-Mauroux, and M. Piorkowski, "Online anomaly detection over Big Data streams," in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 2015, pp. 1113 – 1122.

[6] Z. Hasani, "Anomaly detection algorithms for streaming data: Performance comparison," *Journal of Computer Science*, vol. 16, no. 7, pp. 950 – 955, 2020.

[7] M. Zhang, J. Guo, X. Li, and R. Jin, "Data-driven anomaly detection approach for time-series streaming data," *Sensors (Switzerland)*, vol. 20, no. 19, pp. 1 – 17, 2020.

[8] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A Review on Outlier/Anomaly Detection in Time Series Data," *ACM Computing Surveys*, vol. 54, no. 3, 2021.

[9] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, 2020.

[10] Z. Wang, X. Lin, A. Mishra, and R. Sriharsha, "Online Changepoint Detection on a Budget," in *IEEE International Conference on Data Mining Workshops, ICDMW*, vol. 2021-December, 2021, pp. 414 – 420.

[11] N. Belacel, R. Richard, D. P. Rangavajjala, and R. Adhaduk, "Online Anomaly Detection for Streaming Data Implemented on Top of Kafka, Scikit-Multiflow and River," in *Lecture Notes in Networks and Systems*, vol. 360 LNNS, 2022, pp. 826 – 836.

[12] V. H. A. Ribeiro and G. Reynoso-Meza, "Online anomaly detection for drinking water quality using a multi-objective machine learning approach," in *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion*, 2018, pp. 1 – 2.

[13] P. Boniol, J. Paparrizos, T. Palpanas, and M. J. Franklin, "Sand in action: Subsequence anomaly detection for streams," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2867 – 2870, 2021.

[14] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 339 – 367, 2017.

[15] R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, I. A. Targio Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A Survey," *International Journal of Information Management*, vol. 45, pp. 289 – 307, 2019.

[16] P. D. Talagala, R. J. Hyndman, K. Smith-Miles, S. Kandanaarachchi, and M. A. Muñoz, "Anomaly Detection in Streaming Nonstationary Temporal Data," *Journal of Computational and Graphical Statistics*, vol. 29, no. 1, pp. 13 – 27, 2020.

[17] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017.

[18] M. Munir, S. A. Siddiqui, M. A. Chattha, A. Dengel, and S. Ahmed, "FuseAD: Unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models," *Sensors (Switzerland)*, vol. 19, no. 11, 2019.

[19] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at Microsoft," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 3009 – 3017.

[20] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD '03*, 2003, pp. 2 – 11.

[21] K. M. Carter and W. W. Streilein, "Probabilistic reasoning for streaming anomaly detection," in *2012 IEEE Statistical Signal Processing Workshop, SSP 2012*, 2012, pp. 377 – 380.

[22] J. Lima, R. Salles, F. Porto, R. Coutinho, P. Alpis, L. Escobar, E. Pacitti, and E. Ogasawara, "Forward and Backward Inertial Anomaly Detector: A Novel Time Series Event Detection Method," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2022-July, 2022, pp. 1–8.

[23] R. Salles, K. Belloze, F. Porto, P. H. Gonzalez, and E. Ogasawara, "Nonstationary time series transformation methods: An experimental review," *Knowledge-Based Systems*, vol. 164, pp. 274 – 291, 2019.

[24] A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber, "Strucchange: An R package for testing for structural change in linear regression models," *Journal of Statistical Software*, vol. 7, pp. 1 – 38, 2002.

[25] B. R. Hiraman, M. C. Viresh, and C. K. Abhijeet, "A Study of Apache Kafka in Big Data Stream Processing," in *2018 International Conference on Information, Communication, Engineering and Technology, ICICET 2018*, 2018.

[26] A. S. Iwashita and J. P. Papa, "An Overview on Concept Drift Learning," *IEEE Access*, vol. 7, pp. 1532 – 1547, 2019.

[27] L. Giusti, L. Carvalho, A. T. Gomes, R. Coutinho, J. Soares, and E. Ogasawara, "Analyzing flight delay prediction under concept drift," *Evolving Systems*, 2022.

[28] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, 2009.

[29] R. Salles, J. Lima, L. Baroni, A. Castro, L. Carvalho, H. Borges, D. Carvalho, R. Coutinho, E. Bezerra, E. Pacitti, F. Porto, and E. Ogasawara, "harbinger: An Unified Time Series Event Detection Framework," jul 2023. [Online]. Available: https://cran.r-project.org/web/packages/harbinger/index.html

[30] L. Escobar, R. Salles, J. Lima, C. Gea, L. Baroni, A. Ziviani, P. Pires, F. Delicato, R. Coutinho, L. Assis, and E. Ogasawara, "Evaluating Temporal Bias in Time Series Event Detection Methods," *Journal of Information and Data Management*, vol. 12, no. 3, oct 2021.

[31] Z. Hasani, "Robust anomaly detection algorithms for real-time big data: Comparison of algorithms," in *2017 6th Mediterranean Conference on Embedded Computing, MECO 2017 - Including ECYPS 2017, Proceedings*, 2017.

[32] R. Carmona, *Statistical Analysis of Financial Data in R*. Springer Science & Business Media, dec 2013.

[33] J.-I. Takeuchi and K. Yamanishi, "A unifying framework for detecting outliers and change points from time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 4, pp. 482 – 492, 2006.

[34] J. Han, J. Pei, and H. Tong, *Data Mining: Concepts and Techniques*, 4th ed. Cambridge, MA: Morgan Kaufmann, oct 2022.

[35] F. Lomio, D. M. Baselga, S. Moreschini, H. Huttunen, and D. Taibi, "RARE: A labeled dataset for cloud-native memory anomalies," in *MaLTeSQuE 2020 - Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation, Co-located with ESEC/FSE 2020*, 2020, pp. 19 – 24.

[36] K. Bach, O. Gundersen, C. Knappskog, and P. Öztürk, "Automatic case capturing for problematic drilling situations," in *Lecture Notes in Computer Science*, vol. 8765, 2014, pp. 48–62.