Streamify Time-Series Anomaly Detection Methods

Konstantina Gkritsa
Aristotle University of Thessaloniki
Thessaloniki, Greece
gkritsak@csd.auth.gr

Chrysoula Moschou
Aristotle University of Thessaloniki
Thessaloniki, Greece
cmosch@csd.auth.gr

Anastasios Papadopoulos
Aristotle University of Thessaloniki
Thessaloniki, Greece
apapadoi@csd.auth.gr

ABSTRACT

Anomaly detection is gaining importance, particularly within the domain of time series. Time series pose unique challenges for anomaly detection due to their inherent characteristics; they are fast-growing and are ubiquitous across different domains and settings. In this paper we propose different modifications to well-known algorithms specifically tailored to address anomaly detection in time series data. Specifically, we propose modifications for Isolation Forest, Histogram-Based Outlier Score, Polynomial Approximation, CNN and LSTM. For each algorithm, we implement (at least) two variations: a baseline (naive) batch approach technique and a more advanced version incorporating our modifications.

Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/moschouChry/DWS-MMD-Streamify-Time-Series-Anomaly-Detection-Methods.

1 DATASETS

In this section, we describe briefly the datasets that were used for the experiments presented in the paper. Specifically, the datasets originate from the TSB-UAD benchmark [11].

1.1 TSB-UAD

TSB-UAD is a new benchmark created for evaluation of time-series anomaly detection methods. Before TSB-UAD, available methods often relied on limited or biased data, leading to deceiving results. TSB-UAD addresses these restrictions, by collecting and processing over 13,000 datasets from multiple domains and sources, with high variability of anomaly types, ratios, and sizes.

1.1.1 Dataset selection. In the course of our experiments, we utilized the three datasets: Dodgers, MGAB, and NAB. These datasets served as the foundation for our research and enabled us to analyze and evaluate the performance of our methods under various conditions.

Table 1: Part of the table shown in [11]. Summary characteristics of the three datasets. R_c is the relative contrast a coefficient measuring the distribution of normal and abnormal points, with smaller values indicating relatively higher difficulty.

Dataset	Count	Average Length	Average # Anomalies	Average # Abnormal Points	R_c
Dodgers [7]	1	50400.0	133.0	5612.0	2.02
MGAB [12]	10	100000.0	10.0	200.0	27.64
NAB [1]	58	6301.7	2.0	575.5	2.67

As [11] mentions,

- Dodgers [7] is a loop sensor data for the Glendale on-ramp for the 101 North freeway in Los Angeles and the anomalies represent unusual traffic after a Dodgers game.
- MGAB [12] is composed of Mackey-Glass time series with non-trivial anomalies. Mackey-Glass time series exhibit chaotic behavior that is difficult for the human eye to distinguish.
- NAB [1] is composed of labeled real-world and artificial time series including AWS server metrics, online advertisement clicking rates, real time traffic data, and a collection of Twitter mentions of large publicly-traded companies.

1.2 Dataset generation

To investigate the impact of distribution shifts on anomaly detection performance, we constructed composite time series through the process of concatenation. This resulted in the creation of three distinct normality categories. Normality 1, which encompasses a singular time series, reflecting the absence of distributional shifts. Normality 2, which comprises the concatenation of two time series originating from different domains and datasets, introducing a single distribution shift. Normality 3, which represents the concatenation of three time series from diverse domains and datasets, incorporating two distribution shifts.

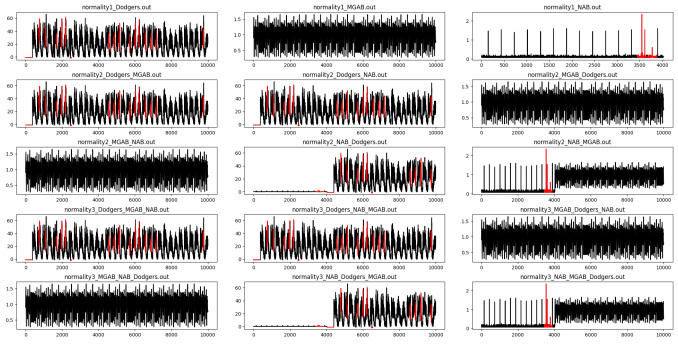

Figure 1: A sample of the generated datasets. The points in red color represent anomalies.

2 ISOLATION FOREST

For the purpose of this paper the Isolation Forest (iForerst) algorithm was modified to operate on streaming settings.

2.1 Variation 1

For variation 1, we trained an isolation forest model on each batch of arriving data and calculated the anomaly score. We evaluated five different batch sizes: 1000, 1500, 2000, 2500, and 3000. For each

Table 2: This figure analyzes the average Area Under the Curve (AUC) value and execution time of our iforest naive approach across different batch sizes. The algorithm run on all our files, calculating the average AUC and execution time for each batch size.

Batch size	Average AUC	Average execution time (sec)
1000	0.479	16.740
1500	0.474	11.657
2000	0.490	9.099
2500	0.482	7.345
3000	0.435	6.257

batch size, we recorded the average Area Under the Curve (AUC) and the execution time in seconds. The results are presented in Table 2.

2.2 Variation 2

In the second variant, we created an isolation forest (iForest) algorithm with a sliding window to achieve anomaly detection in streaming data, taking inspiration from the research presented in [6]. This approach entails defining a window size. Once the window is filled with data points, an initial iForest model is trained on that specific window, and the corresponding anomaly score is stored. Subsequently, new data is continuously fed into the model until the window reaches its capacity again. At this point, the anomaly rate within the sliding window is evaluated against a predetermined threshold. If the anomaly rate is below the drift threshold, the existing iForest model is retained. Conversely, if the anomaly rate surpasses the threshold, a retraining process is initiated using the data contained within the current window. This iterative process of training and evaluation continues for the entirety of the incoming data stream. A workflow can be seen in Figure 2.

To further investigate the effectiveness of the aforementioned approach, we conducted experiments with two variations. The first variation involved training the model on a fixed, predetermined number of data points, before using the sliding window approach described earlier. The second variation entirely used a sliding window. Additionally, we assessed the impact of different anomaly threshold, drift threshold and window size values on the model's performance.

Specifically, we evaluated the anomaly detection performance using various parameter combinations. Window sizes ranging from 25 to 2000 data points. Anomaly thresholds of 0.4, 0.5, and 0.6 were investigated to determine where the point is an anomaly or not. Data points exceeding this threshold were flagged as anomalies. We also evaluated drift thresholds of 0.4, 0.5, and 0.6 to determine the need for model retraining. Average anomaly score within a window exceeding this threshold, means a potential shift in the data distribution and a need for model retraining. Finally, we considered the initial training argument to define whether the model passed an initial training phase or not. The detailed results obtained through these evaluations are presented in Figure 3.

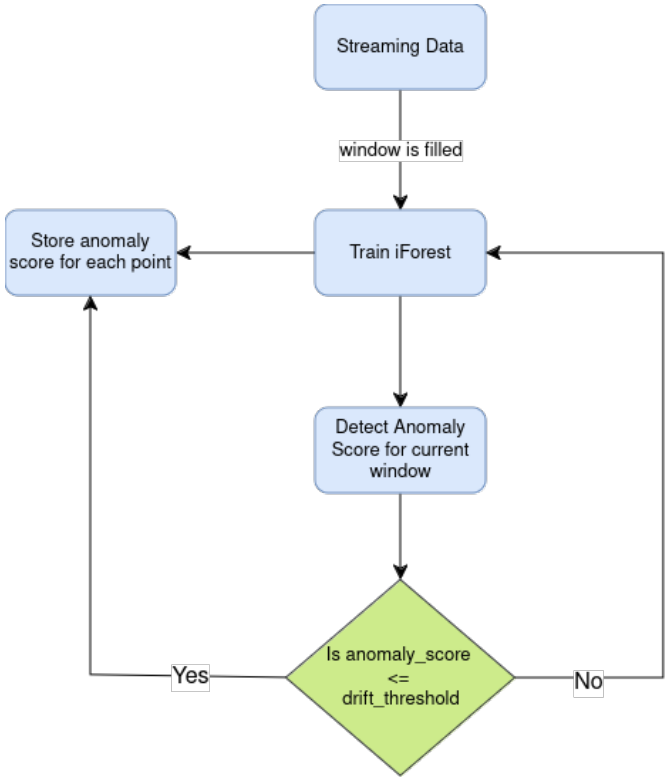

Figure 2: Process flow to visualize the second variant of Isolation Forest algorithm.

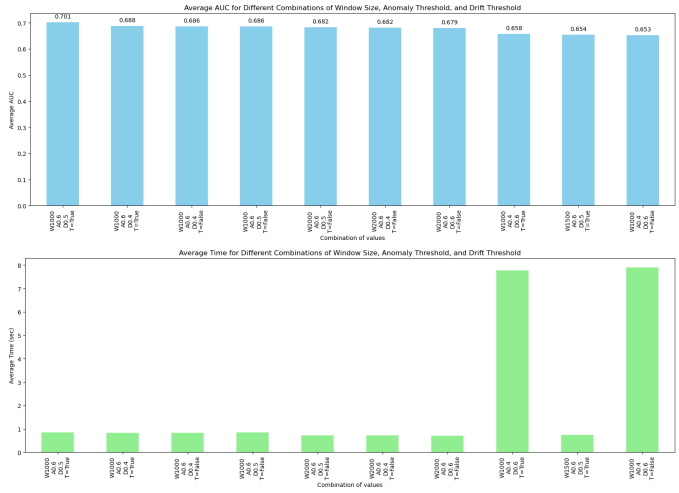

Figure 3: Average AUC and execution time for different combinations of iForest's implementation parameters. A is for anomaly threshold, W is for window size, D is for drift threshold and T indicates whether an initial training phase exists.

3 POLYNOMIAL APPROXIMATION

Regarding the polynomial approximation again we evaluate a naive streaming approach with data arriving at batches, with a fixed batch size set to 2000 points.

3.1 Variation 2

For the (main) proposed variant of POLY, the data still arrive in batches with a fixed batch size set to 2000 points. The difference between the initial POLY implementation is that in order to calculate the scores when a new window arrives, the data are approximated with all the previous fitted polynomials per window and the scores are averaged for each point across polynomials. The motivation behind this is to see how well the new data can be reconstructed concerning previous data, a concept that might help when normality changes. The proposed variation can be used in two ways, with having all the previous polynomials or just having the previous polynomials in the current window, where in the first case the method has larger space requirements. In any case, the method requires storing the polynomial coefficients, where for a polynomial of degree 3 typically available memory space allows for a quite large look-back buffer.

3.2 Variation 3

We also tried to train the polynomial coefficients in an unsupervised way based on [8] using the Binary information gain loss, where essentially a polynomial of degree 3 corresponds to a neuron as shown in Figure 4 where the coefficients are learned incrementally or with another reference time series. The idea was to let the model learn the normal and abnormal class of points without explicit labels. The idea did not work so well with scores having the same value most of the time or decreasing as we go towards the end of the time series (perhaps with unsupervised training in another time series or only training at the start of the time series and not continuously as new points arrive would yield better results). A relative "good" example is shown in Figure 5.

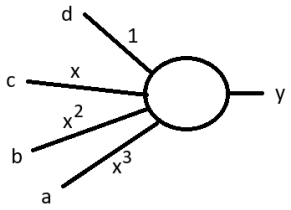

Figure 4: The corresponding neuron of POLY using the BINGO optimization objective. x is the input timestamp index and a,b,c,d are model weights, while y is the output.

4 HISTOGRAM-BASED OUTLIER SCORE

For the HBOS method apart from the data arriving in batches of 2000 points, we utilized algorithms from [3] that allow computing of approximate histograms from streaming data. The data again arrive in batches of 2000 and the histogram is updated incrementally, while the anomaly score is calculated as the initial implementation.

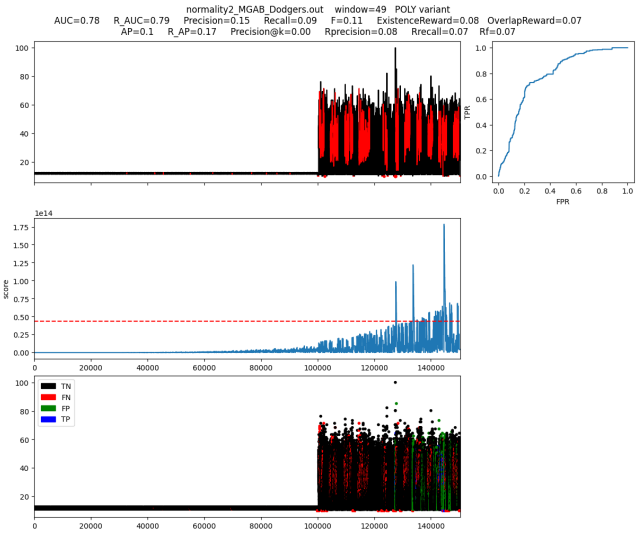

Figure 5: A good example for the POLY algorithm with BINGO optimization.

This approach again tries to keep up with the data trends as new batches arrive. An interesting alternative would be to compute a different histogram for each batch and apply a weighting scheme like the SAND algorithm [5]. Also, this method is space-efficient for streaming data as we only need to store the bin heights and bin edges.

5 CONVOLUTIONAL NEURAL NETWORK

For the CNN method apart from the data arriving in batches of 2000 points and training a different network on each of the batches, we modify the method by replacing the Adam optimizer with Stochastic Gradient Descent [2], where a single model is maintained across the whole time series and for each new batch (with size 64 in this case) the model first tries to forecast the upcoming points to compute the anomaly score and then it is trained incrementally on the points of the batch.

6 LONG SHORT TERM MEMORY NETWORKS (SUPERVISED)

Although labeled data in time series, especially for anomaly detection, is not commonly available, we decided to explore and implement a supervised method for the sake of this project. It's important to note that this method is not directly comparable to other unsupervised methods, as it assumes some prior knowledge about the time series labels.

Our objective was to train a model that, given a predefined number of previous points and the current one, could predict whether the current point is normal (label 0) or an anomaly (label 1). We used TensorFlow for training. Specifically, we utilized the first 15% of the total points from the time series with normality 1 for the train and validation set. From that 15% we used the 80% to train for each case a Long Short Term Memory (LSTM) neural network model, employing the Adam optimizer and the rest as a validation

Table 3: Information from the training process of each time-series.

Time-series	Dodgers	MGAB	NAB
Total points in train and validation set	7485	15000	604
Number of abnormal points in train and validation set	486	0	0
Training time (s)	119	168.3	17.9

set. For the loss computation, we used the 'binary_crossentropy' loss function and a 'sigmoid' activation function in the last Dense layer of the model.

We used a fixed window size of 50 points in time to predict the label of the 50th point. For the training, we set the total number of epochs to 1000 with an early stopping patience of 5 epochs and a 'ReduceLROnPlateau' for the learning rate with an initial value of 0.001, patience of 3 epochs, and a factor of 0.5. The batch size for the training was chosen to be 256 samples. This setup results in a many-to-one model that, given a window of points, predicts the label of the last one. It actually predicts a number between 0 and 1, so the higher the number, the higher the probability that the point is an anomaly.

Our datasets have different normalities, so our goal was to see how a model trained on data with a specific normality would behave when applied to another dataset with a different normality. We trained three initial models using the first 10% of the total points from the time series with normality 1, as mentioned earlier. For the datasets with normality greater than 1, we used the base model that corresponds to the first time series of the concatenated datasets with normality 2 and normality 3.

Table 3 shows the number of samples in the training set and the number of anomalies as well. Notably, for the datasets MGAB and NAB, there were no anomalies in the training set. This provides an interesting opportunity to investigate how a model trained on normal data behaves when applied to anomalous data. The last line of the table shows also the training time.

For the offline implementation of this method, we assume that the test set is delivered in a single, large batch equivalent to the entire length of the test set. In contrast, for the online implementation, the first variation (naive streaming) handles the data in smaller batches of 1000 points each.

6.1 Variation 2

For this variation, we implemented an active learning approach. In this scenario, after processing 10 batches, we assigned labels based on the trained model's predictions. Points with the highest confidence in being anomalies were labeled accordingly.

Given that the model outputs scores ranging from 0 to 1, we considered a point to be an anomaly if its score exceeded a specific threshold. The threshold we chose was 0.8. We then used these now-labeled data to further train the model, trying to improve its accuracy.

6.2 Variation 3

For this variation, we trained a general model on all of the training sets of the three base datasets. We did not apply any normalization to the data before using them for training, as we wanted to see if a model could learn to predict labels adequately for multiple data distributions. Of course, even in this scenario where we take information from multiple sources, we assume that our test data will belong to one of the three initial distributions. This assumption may be extreme in some applications but can also be valid in certain cases. For example, if we want to detect anomalies in heart rate recordings from a smartwatch, we can know the different distributions of the data for the various activities a person can perform throughout the day (sleeping, standing, running, etc.).

The training parameters were the same as before; only the training data changed. The total training time for this model was 535 s.

Table 4: Comparing the two iForest variation based on the different normalities. The best performance is shown in bold.

Algorithm	Normality 1	Normality 2	Normality 3
Variant 1 (iForest)	0.496	0.541	0.440
Variant 2 (iForest)	**0.575**	**0.727**	**0.704**

7 EXPERIMENTAL EVALUATION

7.1 Isolation Forest

In this section we choose the best performing case for both variations and averaged their AUC value for each normality. The results are displayed in Table 4. It is evident that our modification of iForest improved the AUC by 20% for normalities 2 and 3, and by 8% for normality 1.

Furthermore, in Table 5 we present a comprehensive comparison of our two proposed variations with both the naive (offline) streaming baseline (where the entire dataset is used as input, simulating immediate knowledge of the future) and the SAND algorithm. This comparison includes execution time, in seconds, alongside the results.

7.2 LSTM

Table 6 shows the AUC values for the different variations, the time needed to process the test set, and the average AUC values for each method. The best performance for each time series is displayed in bold.

For the SAND method, we did not rerun the method on the test set as only a small fraction of the time series is used for training. We believe that the results of the SAND method for the whole time series are a good representation of its behavior on the test set.

7.3 HBOS, POLY and CNN

For POLY and HBOS we evaluate both with and without per batch normalization of scores (without means scaling on the final list of produced scores) and keep the best across each time series. For CNN we always apply normalization on final list of scores. Results can be found in Table 7.

Table 5: Experimental Evaluation results for the iForest models. The best performance for each time series is displayed in bold.

Time series / AUC / Time	SAND	Naive Streaming	Variation 1	Variation 2
Normality 1				
normality1_NAB	**0.99** / 20.8 sec	0.505 / 0.145 sec	0.494 / 0.225 sec	0.494 / 0.115 sec
normality1_Dodgers	**0.73** / 284.9 sec	0.637 / 1.396 sec	0.562 / 3.249 sec	0.779 / 1.349 sec
normality1_MGAB	**0.72** / 1809.7 sec	0.537 / 2.811 sec	0.439 / 7.184 sec	0.459 / 2.757 sec
Normality 2				
normality2_NAB_MGAB	0.54 / 1850.9 sec	**0.817** / 2.688 sec	0.687 / 7.047 sec	0.652 / 2.854 sec
normality2_MGAB_NAB	0.67 / 1788.0 sec	**0.819** / 2.706 sec	0.676 / 7.208 sec	0.689 / 2.918 sec
normality2_NAB_Dodgers	0.71 / 339.9 sec	0.639 / 1.494 sec	0.563 / 3.546 sec	**0.750** / 1.489 sec
normality2_Dodgers_MGAB	0.74 / 2485.6 sec	**0.882** / 4.057 sec	0.424 / 10.173 sec	0.795 / 3.277 sec
normality2_Dodgers_NAB	0.71 / 315.8 sec	0.636 / 1.532 sec	0.542 / 3.609 sec	**0.779** / 1.436 sec
normality2_MGAB_Dodgers	0.64 / 3417.5 sec	**0.876** / 4.049 sec	0.395 / 10.160 sec	0.652 / 4.157 sec
Normality 3				
normality3_NAB_Dodgers_MGAB	0.71 / 2533.6 sec	**0.884** / 4.267 sec	0.421 / 10.216 sec	0.751 / 3.314 sec
normality3_Dodgers_MGAB_NAB	0.73 / 2534.3 sec	**0.874** / 4.269 sec	0.452 / 10.554 sec	0.779 / 3.294 sec
normality3_Dodgers_NAB_MGAB	0.75 / 2456.8 sec	**0.877** / 4.217 sec	0.460 / 10.351 sec	0.838 / 3.377 sec
normality3_MGAB_NAB_Dodgers	0.57 / 3527.6 sec	**0.880** / 4.192 sec	0.449 / 10.132 sec	0.652 / 4.162 sec
normality3_MGAB_Dodgers_NAB	0.61 / 3244.7 sec	**0.875** / 4.143 sec	0.390 / 10.197 sec	0.630 / 4.162 sec
normality3_NAB_MGAB_Dodgers	0.69 / 2893.2 sec	**0.875** / 4.310 sec	0.426 / 10.795 sec	0.658 / 4.332 sec
Average	0.70 / 1966.8 sec	**0.77** / 3.085 sec	0.492 / 7.645 sec	0.69 / 2.866 sec

Table 6: Experimental Evaluation results for the LSTM models. The best performance for each time series is displayed in bold.

Time series / AUC / Time	SAND	Naive Streaming	Variation 2	Variation 3
Normality 1				
normality1_NAB	**0.99** / 20.8 sec	0.44 / **3.0** sec	0.38 / 4.0 sec	0.39 / 5.0 sec
normality1_Dodgers	0.73 / 284.9 sec	**0.81** / **37.0** sec	0.65 / 117.0 sec	0.79 / 60.0 sec
normality1_MGAB	**0.72** / 1809.7 sec	0.42 / **74.0** sec	0.40 / 231.0 sec	0.65 / 131.0 sec
Normality 2				
normality2_NAB_MGAB	0.54 / 1850.9 sec	0.42 / 132.0 sec	0.45 / 202.0 sec	**0.65** / **87.0** sec
normality2_MGAB_NAB	0.67 / 1788.0 sec	0.78 / **72.0** sec	0.78 / 214.0 sec	**0.86** / 136.0 sec
normality2_NAB_Dodgers	0.71 / 339.9 sec	0.66 / **66.0** sec	0.69 / 100.0 sec	**0.80** / 77.0 sec
normality2_Dodgers_MGAB	0.74 / 2485.6 sec	**0.92** / **103.0** sec	0.50 / 151.0 sec	**0.92** / 213.0 sec
normality2_Dodgers_NAB	0.71 / 315.8 sec	**0.79** / **36.0** sec	0.63 / 301.0 sec	0.75 / 72.0 sec
normality2_MGAB_Dodgers	0.64 / 3417.5 sec	0.84 / **103.0** sec	0.58 / 317.0 sec	**0.91** / 191 sec
Normality 3				
normality3_NAB_Dodgers_MGAB	0.71 / 2533.6 sec	0.81 / 200.0 sec	0.56 / 491.0 sec	**0.93** / **120.0** sec
normality3_Dodgers_MGAB_NAB	0.73 / 2534.3 sec	**0.92** / 191.0 sec	0.55 / 308.0 sec	**0.92** / **123.0** sec
normality3_Dodgers_NAB_MGAB	0.75 / 2456.8 sec	**0.92** / 192.0 sec	0.56 / 309.0 sec	**0.92** / **107.0** sec
normality3_MGAB_NAB_Dodgers	0.57 / 3527.6 sec	0.83 / 178.0 sec	0.64 / 454.0 sec	**0.90** / **117.0** sec
normality3_MGAB_Dodgers_NAB	0.61 / 3244.7 sec	0.83 / 188.0 sec	0.62 / 313.0 sec	**0.89** / **126.0** sec
normality3_NAB_MGAB_Dodgers	0.69 / 2893.2 sec	0.80 / 181.0 sec	0.55 / 515.0 sec	**0.91** / **125.0** sec
Average	0.70 / 1966.8 sec	0.746 / **117.1** sec	0.571 / 268.5 sec	**0.770** / 112.7 sec

Time series / AUC / Time	SAND	HBOS	POLY	CNN	N. HBOS	N. POLY	N. CNN	V. HBOS	V. POLY	V. POLY WAPPA	V. CNN
Normality 1											
NAB	**0.99** / 20.8 sec	0.96 / 0.34 sec	0.97 / 0.03 sec	0.52 / 8.4 sec	wo 0.98 / 0.38 sec	**wo 0.99** / **0.006 sec**	w 0.52 / 8.75 sec	wo 0.96 / 2.0 sec	wo 0.96 / **0.006 sec**	0.95 / 0.01 sec	0.51 / 11.2 sec
Dodgers	**0.73** / 284.9 sec	0.30 / 3.9 sec	0.69 / 0.10 sec	0.62 / 10.3 sec	w 0.33 / 4.4 sec	wo 0.58 / **0.02 sec**	w/wo 0.51 / 36.7/39.0 sec	w 0.35 / 30.5 sec	wo 0.58 / 0.03 sec	0.43 / 0.92 sec	0.67 / 39.8 sec
MGAB	**0.72** / 1809.7 sec	0.50 / 1.2 sec	0.49 / **0.14 sec**	0.57 / 22.7 sec	w 0.53 / 1.5 sec	w 0.49 / **0.14 sec**	wo 0.56 / 77.5 sec	w 0.52 / 9.9 sec	w 0.59 / 0.42 sec	0.39 / 34.8 sec	0.58 / 100.1 sec
Normality 2											
NAB_MGAB	0.54 / 1850.9 sec	**0.80** / 1.3 sec	0.24 / **0.15 sec**	0.64 / 25.6 sec	w 0.21 / 1.5 sec	w 0.43 / **0.15 sec**	wo 0.48 / 81.2 sec	w 0.19 / 9.9 sec	wo 0.30 / 0.42 sec	0.24 / 39.9 sec	0.69 $^{+}$ / 105.7 sec
MGAB_NAB	0.67 / 1788.0 sec	0.80 / 1.3 sec	0.24 / 0.16 sec	0.80 / 21.2 sec	w 0.43 / 1.5 sec	w 0.43 / **0.15 sec**	wo 0.47 / 81.8 sec	**wo 0.81** $^{+}$ / 10.1 sec	w 0.30 / 0.44 sec	0.14 / 37.5 sec	0.77 $^{+}$ / 109.0 sec
NAB_Dodgers	**0.71** / 339.9 sec	0.32 / 4.2 sec	0.68 / 0.10 sec	0.33 / 10.2 sec	w 0.35 / 4.8 sec	wo 0.60 / **0.03 sec**	w 0.49 / 39.6 sec	w 0.33 / 10.4 sec	wo 0.58 / **0.03 sec**	0.51 / 1.5 sec	0.59 / 52.7 sec
Dodgers_MGAB	0.74 / 2485.6 sec	0.74 / 11.1 sec	**0.83** / 0.28 sec	0.34 / 34.4 sec	wo 0.76 / 13.0 sec	wo 0.72 / **0.08 sec**	wo 0.54 / 126.1 sec	wo 0.77 $^{+}$ / 88.3 sec	wo 0.72 / 0.10 sec	0.75 $^{+}$ / 8.9 sec	0.48 / 155.7 sec
Dodgers_NAB	**0.71** / 315.8 sec	0.32 / 4.3 sec	0.64 / 0.10 sec	0.64 / 10.6 sec	w 0.39 / 4.8 sec	wo 0.56 / **0.03 sec**	wo 0.52 / 41.5 sec	w 0.37 / 32.5 sec	wo 0.56 / 0.04 sec	0.42 / 1.5 sec	0.65 / 47.2 sec
MGAB_Dodgers	0.64 / 3417.5 sec	0.67 / 1.9 sec	0.79 / 0.34 sec	0.79 / 31.6 sec	wo 0.78 / 2.2 sec	wo 0.72 / **0.19 sec**	wo 0.38 / 118.2 sec	wo 0.79 $^{+}$ / 14.9 sec	wo 0.72 $^{+}$ / 0.6 sec	**0.83** $^{+}$ / 83.1 sec	0.79 $^{+}$ / 145.5 sec
Normality 3											
NAB_Dodgers_MGAB	0.71 / 2533.6 sec	0.72 / 11.6 sec	**0.87** / 0.28 sec	0.29 / 49.8 sec	wo 0.72 / 14.1 sec	wo 0.71 / **0.08 sec**	wo 0.47 / 132.5 sec	w 0.75 $^{+}$ / 93.5 sec	wo 0.70 / 0.11 sec	0.69 / 10.3 sec	0.47 / 152.4 sec
Dodgers_MGAB_NAB	0.73 / 2534.3 sec	0.72 / 11.9 sec	**0.78** / 0.28 sec	0.30 / 47.6 sec	wo 0.72 / 13.4 sec	wo 0.68 / **0.08 sec**	wo 0.51 / 130.7 sec	w 0.74 $^{+}$ / 92.8 sec	wo 0.68 / 0.11 sec	0.71 / 9.8 sec	0.44 / 170.1 sec
Dodgers_NAB_MGAB	0.75 / 2456.8 sec	0.72 / 12.0 sec	**0.79** / 0.28 sec	0.33 / 43.2 sec	wo 0.72 / 13.4 sec	wo 0.68 / **0.08 sec**	wo 0.50 / 127.1 sec	wo 0.76 $^{+}$ / 91.7 sec	wo 0.69 / 0.11 sec	0.75 / 10.5 sec	0.44 / 166.3 sec
MGAB_NAB_Dodgers	0.57 / 3527.6 sec	0.65 / 2.0 sec	0.75 / 0.23 sec	0.62 / 44.3 sec	wo 0.74 / 2.4 sec	wo 0.69 / **0.20 sec**	wo 0.37 / 121.3 sec	**wo 0.79** $^{+}$ / 15.2 sec	wo 0.68 $^{+}$ / 0.63 sec	0.74 $^{+}$ / 86.2 sec	0.77 $^{+}$ / 168.5 sec
MGAB_Dodgers_NAB	0.61 / 3244.7 sec	0.65 / 2.0 sec	0.75 / 0.23 sec	0.79 / 40.8 sec	wo 0.74 / 2.4 sec	wo 0.67 / **0.19 sec**	wo 0.38 / 122.0 sec	wo 0.78 $^{+}$ / 15.1 sec	wo 0.67 $^{+}$ / 0.62 sec	**0.82** $^{+}$ / 87.3 sec	0.77 $^{+}$ / 171.2 sec
NAB_MGAB_Dodgers	0.69 / 2893.2 sec	0.65 / 2.0 sec	0.75 / 0.22 sec	0.33 / 43.1 sec	wo 0.74 / 2.4 sec	wo 0.69 / **0.20 sec**	wo 0.43 /124.8 sec	wo 0.74 $^{+}$ / 14.9 sec	wo 0.68 / 0.63 sec	**0.79** $^{+}$ / 94.3 sec	0.69 / 170.7 sec
Average	**0.70** / 1966.8 sec	0.63 / 4.7 sec	0.68 / 0.19 sec	0.52 / 29.5 sec	0.60 / 5.47 sec	0.64 / **0.1 sec**	0.47 / 91.3 sec	0.64 / 3.0 sec	0.62 / 0.28 sec	0.61 / 33.7 sec	0.62 / 117.7 sec

Table 7: Experimental evaluation results for HBOS, POLY and CNN algorithms. The best performance for each time series is displayed in bold. (w) sign means that normalization is applied in the scores of every batch independently of the others, (wo) means that a single normalization is applied in the final list of scores for the whole time series, (N) sign means naive, (V) proposed variant and V. POLY WAPPA is the proposed variant that uses all the previously calculated polynomials to produce anomaly scores. The green colour means that the proposed variant outperformed the naive streaming approach, while red means that it did not. Finally, a $^{+}$ means that the proposed variant outperformed SAND.

Table 8: Collective results for the streaming variations (average AUC values per algorithm per normality). The optimal values corresponding to each normality level are highlighted in bold.

Algorithms	Normality 1	Normality 2	Normality 3
SAND	**0.813**	0.668	0.676
iForest (Variation 1)	0.496	0.541	0.440
iForest (Variation 2)	0.575	**0.727**	0.704
POLY (Variation 1)	0.687	0.577	0.687
POLY (Variation 2)	0.710	0.53	0.683
POLY (Variation 3)	0.590	0.482	0.750
HBOS (Variation 1)	0.613	0.428	0.730
HBOS (Variation 2)	0.610	0.543	**0.760**
CNN (Variation 1)	0.53	0.48	0.443
CNN (Variation 2)	0.587	0.662	0.597
LSTM* (Variation 1)	0.556	0735	0.852
LSTM* (Variation 2)	0.477	0.605	0.580
LSTM* (Variation 3)	*0.610*	*0.815*	*0.912*

*supervised method cannot be directly compared to the rest

7.4 Conclusions

In Table 8, we present the results for every algorithm that was introduced in this paper for each normality level. We can observe that for normality 1, SAND was found to have the best results, for normality 2 the second variation of iForest and for normality 3 the second variation of HBOS has the optimal results.

We can observe that the AUC values for the LSTM methods are relatively high compared to the others. However, it's important to remember that LSTM is a supervised method, benefiting from training on labeled data. By using this method, we assume our model can detect anomalies based on previous time windows. Yet, different time series may need varying amounts of data to detect anomalies effectively—providing too much data might lead to distractions, while providing too little might deprive the model of essential information. Generally, these methods are not ideal for generalizing across datasets, and we must keep this limitation in mind.

REFERENCES

[1] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147. https://doi.org/10.1016/j.neucom.2017.04.070 Online Real-Time Learning Strategies for Data Streams.

[2] Shun-ichi Amari. 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing* 5, 4-5 (1993), 185–196.

[3] Yael Ben-Haim and Elad Tom-Tov. 2010. A Streaming Parallel Decision Tree Algorithm. *Journal of Machine Learning Research* 11, 2 (2010).

[4] 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.

[5] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J Franklin. 2021. SAND: streaming subsequence anomaly detection. *Proceedings of the VLDB Endowment* 14, 10 (2021), 1717–1729.

[6] Zhiguo Ding and Minrui Fei. 2013. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. *IFAC Proceedings Volumes* 46, 20 (2013), 12–17. https://doi.org/10.3182/20130902-3-CN-3020.00044 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013.

[7] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. 2006. Adaptive event detection with time-varying poisson processes. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, USA) *(KDD '06)*. Association for Computing Machinery, New York, NY, USA, 207–216. https://doi.org/10.1145/1150402.1150428

[8] Magdalena Klapper-Rybicka, Nicol N Schraudolph, and Jürgen Schmidhuber. 2001. Unsupervised learning in LSTM recurrent neural networks. In *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*. Springer, 684–691.

[9] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* 6, 1, Article 3 (mar 2012), 39 pages. https://doi.org/10.1145/2133360.2133363

[10] 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.

[11] 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.

[12] Markus Thill, Wolfgang Konen, and Thomas Bäck. 2020. *MarkusThill/MGAB: The Mackey-Glass Anomaly Benchmark*. https://doi.org/10.5281/zenodo.3762385