Energy-Efficient Change Point Detection Algorithm for Resource-Constrained Devices

Ruitao Xue*
R.Xue5@newcastle.ac.uk
Newcastle University
Newcastle upon Tyne, UK

Kamil Faber
kfaber@agh.edu.pl
AGH University of Krakow
Kraków, Poland

Wojciech Kalka
wkalka@agh.edu.pl
AGH University of Krakow
Kraków, Poland

Devki Nandan Jha
dev.jha@newcastle.ac.uk
Newcastle University
Newcastle upon Tyne, UK

Bartlomiej Sniezynski
bartlomiej.sniezynski@agh.edu.pl
AGH University of Krakow
Kraków, Poland

Roberto Corizzo
rcorizzo@american.edu
American University
Washington, DC, United States

Rajiv Ranjan
Raj.Ranjan@newcastle.ac.uk
Newcastle University
Newcastle upon Tyne, UK

Tomasz Szydlo
tomasz.szydlo@newcastle.ac.uk
Newcastle University
Newcastle upon Tyne, UK

Abstract

This paper presents microWATCH, an efficient change point detection (CPD) algorithm designed for resource-limited IoT devices. Our proposed microWATCH integrates lightweight statistical measures with targeted computational optimizations to reduce computational overhead and ensure efficient execution. Tested on 42 real-world datasets, it achieved state-of-the-art accuracy (F1 score) among algorithms capable of running on ultra-low-power microcontrollers, balancing high accuracy with significantly lower energy. This makes microWATCH highly suitable for IoT edge deployment. We also introduce CPDPerf, an open-source framework for evaluating resource-constrained CPD, demonstrating a practical path toward reliable on-device intelligence.

CCS Concepts

• **Computing methodologies** → **Anomaly detection**; • **Computer systems organization** → **Sensor networks**; • **Software and its engineering** → *Power management*.

Keywords

Change Point Detection, Edge Computing, Energy Consumption, Benchmarking

1 Introduction

Data quality is critical in the Internet of Things (IoT), where decisions in predictive maintenance [1], real-time monitoring [2], smart city [3], and risk management rely on accurate sensor data [4]. Poor data from sensor malfunctions or environmental shifts can lead to flawed insights and costly failures [5]. Change Point Detection (CPD) is a crucial tool for maintaining data integrity by automatically detecting significant shifts in the underlying data distribution of a time series [6]. By identifying data drift and anomalies at the source, CPD ensures the reliability of IoT-driven analytics and improves operational efficiency.

However, the practical deployment of CPD is often limited by device constraints. Sending raw data to the cloud for analysis is frequently infeasible due to unacceptable latency [7], intermittent network autonomy, prohibitive data costs, and significant privacy risks [8]. This makes embedding intelligence directly onto resource-constrained microcontrollers (MCUs) like the ESP32 a necessity [9]. Yet, this paradigm creates a fundamental conflict: many state-of-the-art CPD algorithms, particularly those using Bayesian inference [10] or kernel methods [11], are too resource-intensive for MCUs, relying on extensive historical data or complex matrix operations that exceed available memory and processing power.

To address these challenges, we propose **microWATCH**, a CPD algorithm specifically designed for resource-constrained devices. Our approach incorporates multiple optimizations to minimize memory usage and computational overhead, ensuring efficient execution on resource-constrained hardware. Additionally, we provide a MicroPython implementation compatible with CPython, enabling seamless deployment across a wide range of devices, from microcontrollers to more powerful edge computing systems.

Moreover, we provide **CPDPerf**, a standardized benchmarking framework for comparing change point detection algorithms across resource-constrained IoT devices. The framework allows for the assessment of the efficiency and power consumption of CPD algorithms in environments where computing, memory, and energy resources are extremely limited. The source code for microWATCH and CPDPerf is publicly available. [1]

2 Related Work

Change Point Detection (CPD) is a critical task for monitoring time series data in applications ranging from industrial quality control [12, 13] to continual learning [14, 15]. CPD methods can be categorized [16] as online or offline, and as univariate or multivariate. This work focuses on online methods suitable for near real-time processing on resource-constrained devices.

Classic statistical approaches include CUSUM [17], which monitors cumulative deviations from a mean, and Geometric Moving

[1] https://github.com/firepond/microWATCH

Average (GMA) [18], which uses a weighted average to detect gradual shifts. More complex methods like Bayesian Online Change Point Detection (BOCPD) [10] offer a principled probabilistic framework for identifying change points, with extensions for model selection [19] and robustness to outliers [11]. Recently, the WATCH algorithm [20] introduced a Wasserstein distance-based approach for high-dimensional data, which serves as the foundation for our work.

However, existing CPD methods and evaluation frameworks like TCPDBench [16] primarily focus on detection accuracy, neglecting the computational and energy costs. This omission presents a significant barrier for deploying CPD on edge devices. Our work addresses this gap by developing and benchmarking energy-efficient algorithms specifically for this constrained environment.

3 The Design of microWATCH

In this section, we detail the design of **microWATCH**. We begin by summarizing the original WATCH algorithm [20], then detail our targeted optimizations for resource-constrained environments, and conclude with an ablation study that quantifies the impact of each contribution

3.1 The WATCH Algorithm: A Foundation for Change Point Detection

The WATCH algorithm is a state-of-the-art online CPD method that detects changes in the distribution of high-dimensional time series data. Its core principle is to maintain a representation of the current 'stable' data distribution and compare incoming mini-batches of data against it using the Wasserstein distance.

Its core principle is to maintain a representation of the current 'stable' data distribution and compare incoming mini-batches of data against it using the Wasserstein distance.

It is important to note that this approach is designed to detect distributional shifts between segments of data under the assumption that the data points within each stable segment are independent and identically distributed (i.i.d.) .

It is composed of three stages:

(1) Split time series data I_i into mini-batches B_i. Learn and extract a representation from a small set of batches to form the initial distribution D. Then the threshold of distance η is determined by

$$\eta(D, \epsilon) = \max_{B \in D} W_A(B_j, D)$$

where ϵ is the threshold ratio for the distance value.

(2) As new data points arrive in mini-batches, each batch B_i is compared with D using the Wasserstein distance: $v = W_A(B_i, D)$.

(3) If the computed distance v exceeds a predefined threshold, a change point is detected and a new distribution is created using the new batches. Otherwise, if the distance is below the threshold, the new batch is incorporated into the existing distribution.

While highly effective, WATCH was designed for powerful computing systems and presents several critical bottlenecks for deployment on resource-constrained MCUs:

(1) High Computational Cost: The Wasserstein distance, especially for multivariate data, involves sorting and complex calculations that are computationally expensive.

(2) High Memory Overhead: Storing the entire history of data points in the distribution buffer (D) can exceed the limited RAM of an MCU, especially for long-running processes.

(3) Implementation Dependencies: The original Python/R implementations rely on libraries that are unavailable in environments like MicroPython.

Algorithm 1: Pseudo-code of the **microWATCH**

Data: $\mathbf{I} = (I_1, I_2, ..., I_T)$ – Time Series Data
$\kappa \in \mathbb{N}$ – Min points required to start detection
$\mu \in \mathbb{N}$ – Max points stored for a distribution
$\epsilon \in \mathbb{R}$ – Threshold ratio for the distance value
$\omega \in \mathbb{N}$ – Mini-batch size
Result: R: Set of change points

Initialization: $D \leftarrow \emptyset$ $S = 0$;
Process **I** into sequence $B_1, B_2, .., B_{\frac{T}{\omega}}$ of non-overlapping batches B_i of size ω;
for $B_i \in \mathbf{B}$ **do**
  if $|D| < \kappa$ **then**
    $D \leftarrow D \cup B_i$;
    $S = S + \sum B_i$;
    if $|D| \geq \kappa$ **then**
      $\overline{D} = \frac{S}{|D|}$ <– Mean from running sum;
      $\eta(D, \epsilon) = \epsilon \max_{B \in D} dist(B, \bar{D})$
    end
  else
    $\overline{D} = \frac{S}{|D|}$ <– Mean from running sum;
    $v \leftarrow dist\left(B_i, \overline{D}\right)$ <– Uses lightweight distance
    if $v > \eta$ **then**
      $C \leftarrow i * \omega$;
      $R \leftarrow R \cup \{c\}$;
      $D \leftarrow \{B_i\}$;
    else
      if $|D| < \mu$ **then**
        $D \leftarrow D \cup B_i$;
        $\eta(D, \epsilon) = \epsilon \max_{B \in D} dist(B, \bar{D})$
      end
    end
  end
end

3.2 microWATCH: Optimizations for Energy-Efficient Execution

Our microWATCH retains the core algorithmic loop of WATCH — comparing incoming data batches to a reference distribution — but systematically re-engineers four key aspects of the implementation to overcome the previously identified bottlenecks and ensure efficient execution on MCUs.

Table 1: Ablation study results quantifying the impact of optimizations

Stage	Avg. Speedup	Primary Contribution
0→1	34.5x	Different distance metric, native Python
1→2	1.80x	Reduces distribution mean recalculation
2→3	1.29x	Ensures a fixed memory footprint
0→3	**73.4x**	**All Combined**

1. Lightweight Distance Metrics: The primary modification is the replacement of the Wasserstein distance. microWATCH is architected to use a variety of lightweight statistical distance functions. In this work, we evaluate 39 such measures (e.g., Euclidean, Chebyshev[2]) that rely on simple arithmetic, drastically reducing computational load. The distance $dist(B, D)$ between a batch B and the distribution D is now calculated between B and the mean of D: $dist(B, \bar{D})$.

2. Fixed-Memory with Running-Sum Mean: To eliminate the costly recalculation of the distribution mean at every step, we replace this operation with a computationally trivial running-sum update. This is complemented by replacing the dynamic Python list D with a pre-allocated, fixed-size NumPy array, which guarantees a constant and predictable memory footprint.

3. Self-Contained MicroPython Implementation: Finally, the algorithm is implemented in pure MicroPython with a minimal dependency on NumPy (or ulab in MicroPython), making it directly deployable on MCUs without the need for the extensive libraries required by the original WATCH.

3.3 Ablation Study and Comparing with WATCH

To quantify the impact of our design choices, we performed an ablation study comparing the original WATCH (Stage 0) against progressively optimized versions of microWATCH. The optimized versions are a naive Python port using Chebyshev distance (Stage 1), an implementation with a running sum for mean calculation (Stage 2), and finally, **microWATCH** with a pre-allocated NumPy buffer (Stage 3). To ensure the most rigorous comparison, the original WATCH was configured with its own optimal hyperparameters, while all **microWATCH** stages (1-3) shared a separate set of optimal parameters, both determined through an extensive grid search.

The performance evaluation was conducted on the eight largest datasets (four univariate, four multivariate) from the TCPD benchmark. As summarized in Table 1, transitioning from the R implementation to a pure Python version (Stage 0→1) yielded the most significant performance gain, with an average speedup of 34.5x. Our subsequent algorithmic and memory optimizations provided further improvements: introducing a running sum (Stage 1→2) offered an additional 1.80x speedup by eliminating redundant calculations, while the final move to a pre-allocated buffer (Stage 2→3) provided another 1.29x speedup. More importantly, this final step guarantees a fixed memory footprint, a critical requirement for deployment on resource-constrained devices. While this study uses a representative distance metric (Chebyshev), these fundamental optimizations—a running sum and a fixed memory buffer—provide performance benefits across all distance measures evaluated in Section 5

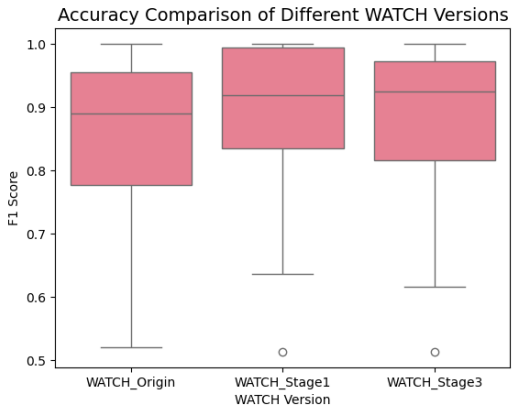

Figure 1: F1 Scores of different WATCH versions

In terms of detection accuracy, an evaluation across all 42 datasets (Fig. 1) shows that our optimized microWATCH (Stage 3) achieves an F1 score comparable to the original WATCH, confirming our optimizations do not compromise effectiveness. Minor output differences between Stage 1 and Stage 3 are attributed to expected floating-point arithmetic variations between their respective mean calculation methods. Overall, this study validates that our optimizations provide substantial performance gains essential for edge deployment without sacrificing detection accuracy.

4 Experiment Setup

This section describes the user case study and real-world validation of our proposed **CPDPerf** system.

4.1 Devices and Power Measurements

To evaluate the performance of our CPD algorithms, we conducted experiments on two distinct devices, representing different levels of computational capabilities and power consumption profiles: a Raspberry Pi 4B (1.5 GHz A72, 4GB RAM), and ESP32-C6 MCU (160 MHz RISC-V, 512KB RAM).

4.2 Algorithm Implementation and Tuning

In addition to **microWATCH**, we have implemented several CPD algorithms (Table 2) in MicroPython environment for comparison. And they are compatible with **CPython**. Many other modern CPD algorithms proved computationally intractable or failed to run due to memory constraints on the ESP32. To ensure a fair comparison, all algorithms, including our proposed methods and the baselines, were tuned via an extensive grid search to optimize their F1 score on each dataset. [3]

[2]Details about other distance measures can be found in the GitHub repository: ACC, Add. Sym. χ^2, Bhattacharyya, Bray-Curtis, Canberra, Chebyshev (Max), Chebyshev (Min), Clark, Czekanowski, Divergence, Euclidean, Google, Gower, Hellinger, Jeffreys, Jensenshannon Divergence, Jensen Difference, K Divergence, Kl Divergence, Kulczynski d, Lorentzian, Manhattan, Matusita, Max Symmetric χ^2, Minkowski (p=2), Motyka, Neyman χ^2, Nonintersection, Pearson χ^2, Penroseshape, Sorgel, Squared χ^2, Squared-chord, Taneja, Tanimoto, Topsoe, Vicis Symmetric χ^2, Vicis Wave Hedges, Wave Hedges

[3]The hyperparameter settings are available in our GitHub repo.

Table 2: The CPD algorithms implemented and tested in the benchmark

Name	Method	Device
BOCPD[10]	Bayesian Online CPD	ESP/Pi
microWATCH	Our work: optimized WATCH[20]	ESP/Pi
BOCPDMS[19]	BOCPD with Model Selection	Pi
PELT[11]	Kernel CPD with Penalty	ESP/Pi
CUSUM[17]	Cumulative Sum Control Chart	ESP/Pi

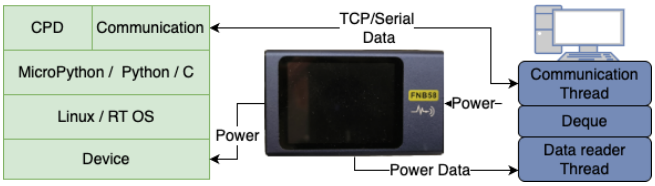

Figure 2: Our test framework.

The algorithms are fine-tuned to optimize parameters that yield the highest F1 score. Subsequently, these optimized parameters are employed in the subsequent experiments. Grid search is utilized for selecting the optimal parameters.

4.3 Test-case Setup

4.3.1 Datasets details. Our evaluation is based on the comprehensive Turing Institute Change Point Detection (TCPD) benchmark [6], which contains 38 univariate and 4 multivariate time series. We leveraged the entire benchmark to conduct the robust Pareto analysis (Fig. 3) that identified the optimal microWATCH variants. For the subsequent head-to-head comparisons against other algorithms, we present detailed results on the 4 multivariate datasets and a representative subset of 4 univariate datasets (centralia, nile, ozone, gdp japan) due to space limitations. This subset was chosen to represent diverse signal types. Complete results for all 42 datasets are available in our public repository. It is important to note that many datasets within the TCPD benchmark, while derived from real-world processes, are well-modeled by a sequence of segments with distinct i.i.d. distributions. This characteristic makes the benchmark particularly suitable for evaluating methods like microWATCH, which are designed to detect such distributional shifts rather than changes in processes with strong temporal dependencies.

4.3.2 Test Framework. To evaluate CPD algorithms, we developed a testing framework and communication protocol for collecting performance and energy data (Fig. 2). The host computer orchestrates the evaluation by configuring the target device with a specific dataset and algorithm. It then commands the device to execute the task while recording the cumulative energy consumption from an external power meter. This automated process allows for precise calculation of the average energy consumed per execution across multiple runs. Full details of the framework and protocol are provided in our public repository.

4.4 Memory Considerations and Hardware Constraints

Memory efficiency plays a critical role in determining whether algorithms are practical for deployment on IoT devices. Existing CPD algorithms, such as BOCPD, BOCPDMS, and PELT, typically require substantial memory resources. For instance, these methods store extensive historical data, posterior distributions, or complex internal state representations, resulting in memory complexity that scales unfavorably with data length or dimensionality.

In contrast, our proposed algorithm and simpler methods like CUSUM maintain minimal statistical summaries (e.g., means, variances, and counts), thus requiring only a small, fixed amount of memory at each timestep. This yields a linear, predictable memory footprint—critical for efficient and stable real-time operation on memory-limited devices.

Due to these inherent memory constraints, we perform initial experiments on a Raspberry Pi, which has sufficient RAM. However, our method is explicitly developed for extremely resource-constrained microcontrollers such as the ESP32. Our algorithm runs efficiently and reliably on such devices, demonstrating their practical suitability for deployment in real-world low-power IoT scenarios.

We tried to implement the following methods, but they can not run on ESP32 due to limited resources: CPNP , RBOCPDMS [21], RFPOP[22], PROPHET[23], WBS [24], CoCPD [25]

5 Results and Discussion

In this section, we discuss the experimental results of **microWATCH**. We compare its performance against selected CPD algorithms in terms of accuracy and energy efficiency. Our analysis, supported by visualizations such as box plots and Pareto front analysis, highlights the trade-offs inherent in deploying CPD algorithms on resource-constrained devices.

5.1 Evaluation microWATCH Distance Measures

To systematically identify the optimal trade-offs between detection accuracy (F1 score) and energy efficiency, we evaluated 39 statistical distance measures within **microWATCH** across all 42 datasets. We ranked each measure's performance and plotted the average F1 ranking against the average energy ranking to conduct a Pareto front analysis. Figure 3 shows this analysis, highlighting the measures that represent optimal trade-offs where no improvement in one criterion is possible without compromising the other. Our analysis identified `WATCH-v7` (**Chebyshev-Min**), `WATCH-v11` (**Euclidean**), and `WATCH-v3` (**Bhattacharyya**) as key representatives for achieving the highest accuracy, best power efficiency, and best overall balance, respectively.

5.2 Comparison with Selected Algorithms on IoT Devices

We compare the selected representative distance measures (`WATCH-v7`, `WATCH-v11` and `WATCH-v3`) against other CPD algorithms we implemented: BOCPD, PELT, BOCPDMS and CUSUM. Due to the limitations of baseline methods on multivariate datasets, the analysis is separated into univariate and multivariate datasets.

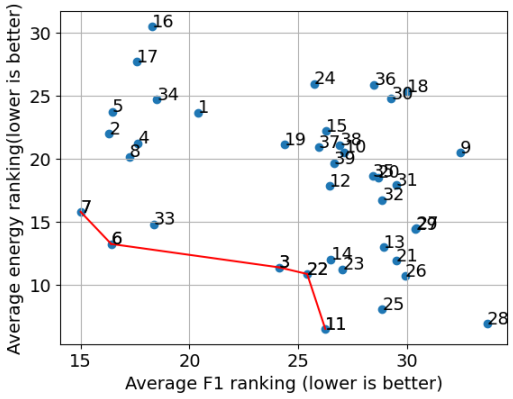

Figure 3: Pareto analysis of all microWATCH distance measures based on rankings. Key variants discussed in the text are v3 (Bhattacharyya), v7 (Chebyshev-Min), and v11 (Euclidean), corresponding to points 3, 7, and 11, respectively.

Table 3: F1 scores on multivariate datasets

Dataset	(WATCH)v3	v7	v11	PELT	BOCPDMS
apple	1	1	1	0.8	0.583
bee waggle	0.929	0.929	0.929	0.929	0.651
occupancy	0.857	0.974	0.621	0.683	0.583
run log	0.966	0.966	0.966	0.667	0.778

Table 4: Energy consumption (mJ) on Raspberry Pi, multivariate datasets

Dataset	(WATCH)v3	v7	v11	PELT	BOCPDMS
apple	2.4	1.4	1.1	293	1570
bee waggle	52.6	22.3	23.7	182,400	301,000
occupancy	63.5	35.3	16.3	3318	56400
run log	2.5	3.9	2.8	374.0	5370

5.2.1 Analysis on Multivariate Datasets. We evaluated our selected **microWATCH** distance measures against baseline algorithms supporting multivariate data, specifically PELT and BOCPDMS. We performed it on the Raspberry Pi, as it was the only device capable of running all algorithms. Due to the significant memory and computational requirements of PELT and BOCPDMS, they failed to run on the ESP32. The Raspberry Pi thus serves as a necessary baseline platform to estimate the relative power efficiency across all methods, highlighting the orders-of-magnitude advantage of microWATCH. The accuracy results are shown in Table 3. The energy results are shown in Table 4.

Our data shows that the selected **microWATCH** distance measures significantly outperform PELT and BOCPDMS in both detection accuracy (F1 score) and energy efficiency. Notably, **microWATCH** consumes roughly 1/100 the energy of PELT and 1/1000 that of BOCPDMS, while simultaneously providing substantially higher F1 scores. Thus, our approach is distinctly superior for multivariate data.

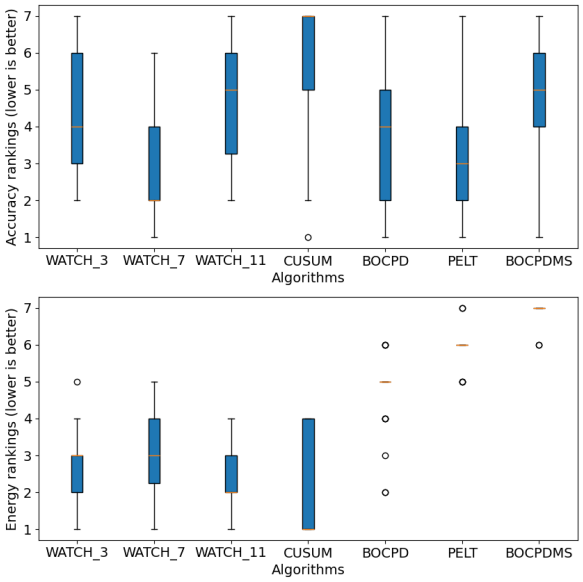

Figure 4: Box plot of cumulative rankings on accuracy and energy for microWATCH and selected algorithms.

Within the selected **microWATCH** distance measures, a clear accuracy-energy trade-off emerges:

- `WATCH-v7` offers the best accuracy at a slightly higher energy cost.
- `WATCH-v11` excels in energy efficiency but achieves slightly lower accuracy.
- `WATCH-v3` provides a balanced choice between accuracy and energy efficiency.

5.2.2 Analysis of Univariate Data. In addition to multivariate datasets, we also analyzed the performance of our algorithms on all 38 univariate datasets from the TCPD benchmark. To summarize the findings, we calculated cumulative rankings for both F1-score and energy consumption. Figure 4 visualizes these rankings as box plots, showing the distribution of performance for each algorithm across all datasets. Furthermore, Figure 5 plots the average energy ranking against the average accuracy ranking to reveal the critical performance-efficiency trade-offs. Detailed per-dataset results are available in our public repository.

Key observations from the analysis are:

(1) `WATCH-v7` achieves the highest F1 score, outperforming all algorithms and the other WATCH variants. It also surpasses BOCPD, BOCPDMS, and PELT in energy efficiency.
(2) `WATCH-v11` and CUSUM exhibit the lowest energy consumption. However, `WATCH-v11` shows better detection accuracy compared to CUSUM, making it preferable in energy-critical applications.
(3) `WATCH-v3` offers a balanced trade-off between accuracy and energy usage. It achieves moderate F1 scores while consuming significantly less energy than heavy algorithms like BOCPDMS, and PELT.

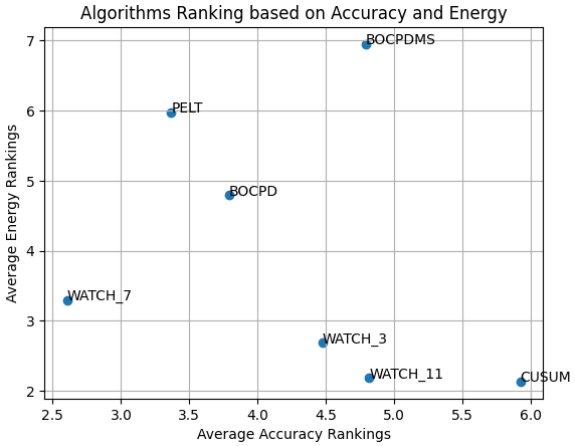

Figure 5: Average energy vs. accuracy rankings for microWATCH variants and other algorithms on univariate datasets.

These results illustrate that the **microWATCH** algorithm variants provide flexible performance choices, enabling practitioners to select the most suitable distance measure according to specific application requirements—be it maximizing accuracy, minimizing energy, or balancing both objectives.

These results highlight the strength and versatility of our proposed algorithm for practical IoT deployments, offering clear guidance for practitioners based on their specific accuracy and resource constraints.

6 Conclusion

In this work, we introduced **microWATCH**, an energy-efficient CPD algorithm tailored for resource-constrained IoT devices. By systematically evaluating statistical distance measures within our benchmarking framework, **CPDPerf**, we identified variants that map out a clear performance-efficiency trade-off.

We acknowledge that **microWATCH** is currently a method for detecting changepoints for i.i.d. data and is not explicitly designed for time series with strong temporal dependencies (e.g., ARIMA processes) and may produce false positives in such scenarios. Future work should focus on extending our lightweight, energy-efficient framework to handle such data, potentially by incorporating models of temporal structure while maintaining a minimal resource footprint.

Our results demonstrate that **microWATCH** is not only competitive with complex baselines in univariate settings but vastly superior in multivariate scenarios, making it a practical and effective solution for enabling reliable on-device monitoring in IoT.

Acknowledgments

This study was supported by the UKRI EPSRC EP/Y028813/1 "National Edge AI Hub for Real Data: Edge Intelligence for Cyberdisturbances and Data Quality".

References

[1] Roksana Haque, Ammar Bajwa, Noor Alam Siddiqui, and Ishtiaque Ahmed. Predictive maintenance in industrial automation: A systematic review of iot sensor technologies and ai algorithms. *American Journal of Interdisciplinary Studies*, 5(01):01–30, 2024.

[2] Yohanes Yohanie Fridelin Panduman, Nobuo Funabiki, Evianita Dewi Fajrianti, Shihao Fang, and Sritrusta Sukaridhoto. A survey of ai techniques in iot applications with use case investigations in the smart environmental monitoring and analytics in real-time iot platform. *Information*, 15(3):153, 2024.

[3] Sultan Altarrazi, Tomasz Szydlo, Devki Jha, and Rajiv Ranjan. Automating data quality monitoring for environmental air quality system. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, pages 1612–1616, 2025.

[4] Lina Zhang, Dongwon Jeong, and Sukhoon Lee. Data quality management in the internet of things. *Sensors*, 21(17), 2021.

[5] Maroua Bahri, Albert Bifet, João Gama, Heitor Murilo Gomes, and Silviu Maniu. Data stream analysis: Foundations, major tasks and tools. *WIREs Data Mining and Knowledge Discovery*, 11(3):e1405, 2021. _eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1405.

[6] Gerrit J. J. van den Burg and Christopher K. I. Williams. An evaluation of change point detection algorithms.

[7] Saurabh Shukla, Mohd Fadzil Hassan, Duc Chung Tran, Rehan Akbar, Irving Vitra Paputungan, and Muhammad Khalid Khan. Improving latency in internet-of-things and cloud computing for real-time data transmission: a systematic literature review (slr). *Cluster Computing*, pages 1–24, 2023.

[8] Jianbing Ni, Kuan Zhang, Xiaodong Lin, and Xuemin Shen. Securing fog computing for internet of things applications: Challenges and solutions. *IEEE Communications Surveys & Tutorials*, 20(1):601–628, 2017.

[9] Riku Immonen and Timo Hämäläinen. Tiny machine learning for resource-constrained microcontrollers. *Journal of Sensors*, 2022(1):7437023, 2022.

[10] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection.

[11] Jeremias Knoblauch, Jack E Jewson, and Theodoros Damoulas. Doubly robust bayesian inference for non-stationary streaming data with \beta-divergences. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] Olivia A Grigg, VT Farewell, and DJ Spiegelhalter. Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts. *Statistical methods in medical research*, 12(2):147–170, 2003.

[13] Zimo Wang, Satish T.S. Bukkapatnam, Soundar R.T. Kumara, Zhenyu Kong, and Zvi Katz. Change detection in precision manufacturing processes under transient conditions. *CIRP Annals*, 63(1):449–452, 2014.

[14] Kamil Faber, Roberto Corizzo, Bartlomiej Sniezynski, and Nathalie Japkowicz. Vlad: Task-agnostic vae-based lifelong anomaly detection. *Neural Networks*, 165:248–273, 2023.

[15] Radek Svoboda, Sebastián Basterrech, Jędrzej Kozal, Jan Platoš, and Michał Woźniak. A natural gas consumption forecasting system for continual learning scenarios based on hoeffding trees with change point detection mechanism. *Knowledge-Based Systems*, 304:112482, 2024.

[16] Gerrit JJ Van den Burg and Christopher KI Williams. An evaluation of change point detection algorithms. *arXiv preprint arXiv:2003.06222*, 2020.

[17] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[18] SW Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1):97–101, 2000.

[19] Jeremias Knoblauch and Theodoros Damoulas. Spatio-temporal bayesian on-line changepoint detection with model selection. In *International Conference on Machine Learning*, pages 2718–2727. PMLR, 2018.

[20] Kamil Faber, Roberto Corizzo, Bartlomiej Sniezynski, Michael Baron, and Nathalie Japkowicz. Watch: Wasserstein change point detection for high-dimensional time series data. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4450–4459. IEEE, 2021.

[21] Jeremias Knoblauch and Theodoros Damoulas. Spatio-temporal bayesian on-line changepoint detection with model selection.

[22] Paul Fearnhead and Guillem Rigaill. Changepoint Detection in the Presence of Outliers. *Journal of the American Statistical Association*, 114(525):169–183, January 2019.

[23] Sean J. Taylor and Benjamin Letham. Forecasting at Scale. *The American Statistician*, 72(1):37–45, January 2018.

[24] Piotr Fryzlewicz. Wild binary segmentation for multiple change-point detection.

[25] Xiangyu Bao, Liang Chen, Jingshu Zhong, Dianliang Wu, and Yu Zheng. A self-supervised contrastive change point detection method for industrial time series. *Engineering Applications of Artificial Intelligence*, 133:108217, July 2024.