RETHINKING HEAVY MODELS IN MULTIVARIATE TIME SERIES ANOMALY DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multivariate time series anomaly detection (MTS-AD) is widely used, but real-world deployments often face tight computational budgets that limit the practicality of deep learning. We revisit whether heavy deep models (high-FLOPs architectures) are necessary to achieve strong detection performance in such settings. We conduct a systematic, compute-aware comparison of statistical, classical machine learning, and deep learning methods across diverse MTS-AD benchmarks, measuring detection with AUROC (threshold-free, thus application-agnostic) and cost with FLOPs (a hardware-agnostic proxy enabling fair cross-method comparison). We find that traditional approaches often match or surpass deep models, which appear less frequently among the top performers, and that the effectiveness-efficiency trade-off commonly favors non-deep alternatives under limited budgets. These results indicate that deep learning is not uniformly superior for MTS-AD and that heavy architectures can be counterproductive in resource-constrained deployments. These findings offer practical guidance for practitioners designing anomaly monitoring systems under compute constraints, highlighting cases where lightweight models are sufficient and heavy deep models may be worth the cost.

1 INTRODUCTION

Time series anomaly detection is a fundamental task in machine learning with wide-ranging applications in domains such as industrial control systems, aerospace telemetry, and cyber security (Kim et al., 2023; Hundman et al., 2018; Landauer et al., 2025). In practice, anomalies are rare and difficult to label, which makes unsupervised anomaly detection methods trained on normal data an essential approach. Over the past decade, deep learning methods have gained prominence for anomaly detection, achieving impressive performance across a variety of benchmark datasets (Zamanzadeh Darban et al., 2024).

However, real-world deployment environments often impose severe hardware and operational constraints. For example, safety-critical systems may need to operate without external connectivity due to security restrictions, preventing the use of cloud-based solutions (Bhamare et al., 2020). Similarly, embedded monitoring devices may lack GPUs or operate under strict thermal and power limitations, making it impractical to deploy computationally intensive deep learning methods (Shuvo et al., 2023; Singh & Gill, 2023). In such cases, the assumption that deep learning is the universally superior solution becomes questionable. While recent research has emphasized novel neural architectures, comparatively little work has jointly examined both effectiveness and efficiency under constrained computing conditions. Most studies focus on accuracy alone (Jia et al., 2025), with only a few recent benchmarks considering accuracy together with runtime and memory usage (Qiu et al., 2025). In addition, several works have highlighted inconsistencies in evaluation protocols and metrics for time series anomaly detection in industry (Si et al., 2024).

This gap encourages us to consider two central questions: (i) What are the most effective options for time series anomaly detection under limited computational resources, and are deep learning methods always the best options? (ii) Does a trade-off between detection performance and computational cost truly exist in practice? Our own experience in industrial applications, including monitoring of air defense systems and equipment in manufacturing settings, has made clear the difficulty of balancing computational demands with detection performance. We believe that many practitioners working in real-world deployments encounter the same challenge.

Table 1: Summary of resource-constrained environments

Typical Domain	Representative Constraint	Common Approach	Key References
Limited memory			
IoT/Wireless Sensor Network nodes; streaming telemetry in manufacturing and equipment monitoring	devices cannot buffer long histories; models must adapt with small working sets under drift	online/streaming learning; feature selection; memory-efficient summaries/sketches	Bifet & Gavaldà (2007) Nancy et al. (2020) Jain et al. (2022) Chatterjee & Ahmed (2022) Mfondoum et al. (2024)
GPU unavailable			
ICS and OT; manufacturing cells; safety/certification-constrained environments	power/thermal, enclosure, and certification constraints preclude accelerators; inference must be CPU-only on-prem	quantization; pruning/compression; CPU-optimized runtime	Han et al. (2016) Jacob et al. (2018) Sipola et al. (2022) Das & Luo (2023) Singh & Gill (2023) Liu et al. (2024a) Fährmann et al. (2025)
Limited CPU capacity			
PLC/RTU-adjacent controllers; fanless industrial PCs; battery-powered sensing	very limited CPU cycles and RAM; strict cycle-time determinism	quantization; low-FLOPs model design; online/streaming updates; selective features	Liu et al. (2008) Goldstein & Dengel (2012) Singh & Gill (2023)
Restricted communication			
air-gapped ICS/OT; secure manufacturing cells; remote or intermittently connected sites	on-prem/offline operation and stringent latency disallow cloud round-trips; data egress may be restricted	local inference at the edge; federated/on-site adaptation; minimal upstream telemetry	Belenguer et al. (2022) Das & Luo (2023) Dehlaghi-Ghadim et al. (2023) Stouffer et al. (2023)

In this paper, we therefore address the questions by conducting a systematic comparative study of unsupervised anomaly detection methods that range from traditional approaches to deep learning methods. Unlike prior studies that have primarily emphasized accuracy, we introduce an evaluation framework that considers both detection performance and computational cost (Mejri et al., 2024). This perspective enables a fair comparison across different methodological approaches. Our evaluation covers diverse real-world datasets drawn from industrial, server, and aerospace domains, ensuring that our findings generalize across multiple application settings.

2 Literature Review

2.1 Resource Constrained Environments

The deployment of models in industrial system is not solely governed by algorithmic accuracy but is equally constrained by system-level limitations. As summarized in Table 1, the literature consistently highlights four recurring scenarios in resource-constrained environments, which encompass limited memory, GPU unavailable, limited CPU capacity, and restricted communication. These scenarios illustrate the historical progression of research toward resource-aware solutions and motivate the comparative analysis conducted in this study.

Limited memory Several studies have demonstrated that real-world industrial environments, including IoT nodes, wireless sensor networks, and manufacturing telemetry systems, often operate under severe limitations in storage and energy, making the buffering of long historical windows infeasible (Jain et al., 2022; Mfondoum et al., 2024; Chatterjee & Ahmed, 2022). In time series anomaly detection, such constraints require models to process data incrementally while maintaining only a limited working set. Consequently, prior research has emphasized memory-efficient representations, dimensionality reduction through feature selection (Nancy et al., 2020), and online or adaptive windowing techniques, which dynamically adjust to evolving conditions (Bifet & Gavaldà, 2007).

GPU unavailable In operational technology (OT) and industrial control system (ICS) environments, the use of GPU accelerators is often infeasible due to strict power, thermal, and certification constraints (Das & Luo, 2023; Liu et al., 2024a; Singh & Gill, 2023; Sipola et al., 2022). As a

result, inference is typically performed on CPUs, making model-level optimization essential. In this context, two advances stand out as particularly effective. Integer-only quantization executes inference entirely with INT8 arithmetic, while deep compression combines pruning and quantization to reduce both computation and memory requirements. These techniques together enable efficient CPU-centric optimization (Fährmann et al., 2025; Jacob et al., 2018; Han et al., 2016).

Limited CPU capacity A related constraint emerges when devices rely on modest CPUs, such as PLC/RTU-adjacent controllers, fanless industrial PCs, or battery-powered IoT nodes. In these settings, the need for cycle time determinism combined with limited computational throughput requires fundamentally low complexity designs. Traditional detectors such as Isolation Forest or histogram-based method remain attractive due to their favorable time and memory complexity (Liu et al., 2008; Goldstein & Dengel, 2012). Recent surveys on Edge AI further emphasize that such CPU-constrained deployments require models explicitly tailored to minimize FLOPs while preserving detection capability (Singh & Gill, 2023; Sipola et al., 2022).

Restricted communication In many industrial and critical infrastructure domains, data transfer to the cloud is either infeasible or prohibited due to latency requirements and strict security policies. Authoritative guideline from National Institute of Standard and Technology explicitly recommend isolation of OT networks, reinforcing the necessity of on-device or on-premise models (Stouffer et al., 2023). Research has therefore explored federated learning approaches to enable collaborative learning without raw data sharing, particularly in distributed industrial environments, as well as lightweight edge frameworks (Belenguer et al., 2022; Dehlaghi-Ghadim et al., 2023).

2.2 Time Series Anomaly Detection Algorithms

Early research on time series anomaly detection was largely grounded in basic statistical models, which assumed stationary distributions and relatively simple dependency structures. Among the most influential were multivariate monitoring methods such as Hotelling's T^2 statistic that leveraged distributional thresholds to detect deviations in industrial manufacturing processes (H.Hotelling, 1947; Ye & Chen, 2001; Zheng et al., 2016). However, their reliance on linearity and stationarity assumptions rendered them less effective when confronted with the high-dimensional, noisy, and non-stationary signals that characterize modern industrial systems.

The subsequent wave of research introduced non-parametric machine learning approaches that relaxed restrictive distributional assumptions. Distance and density-based detectors identified anomalies as local deviations within the data manifold (Angiulli & Pizzuti, 2002; Breunig et al., 2000), while clustering-based techniques grouped time series patterns to distinguish normal from abnormal behavior (He et al., 2003). Ensemble-based strategies, such as Isolation Forest, improved robustness and scalability through randomized partitioning and aggregation (Liu et al., 2008).

Driven by advances in representation learning, the field has recently shifted toward deep learning approaches that explicitly model sequential dependencies and nonlinear structures. Early works employed recurrent neural networks and detected anomalies by reconstructing temporal sequences (Malhotra et al., 2016; Park et al., 2018). This paradigm was later extended by probabilistic generative models, adversarially trained architectures, and attention-based models (Su et al., 2019; Li et al., 2019; Geiger et al., 2020; Zhou et al., 2019; Akcay et al., 2019; Tuli et al., 2022). Collectively, these approaches represent a clear trajectory toward increasingly complex and expressive models, often achieving state-of-the-art accuracy across widely used benchmarks. Nevertheless, their heavy reliance on GPU accelerators and large memory footprints has raised practical concerns regarding deployment in resource-constrained industrial environments.

To contextualize the performance of these approaches, several benchmark studies have systematically compared classical and deep learning methods (Han et al., 2022; Paparrizos et al., 2022; Si et al., 2024). However, most prior studies emphasize accuracy while giving limited attention to computational cost. In this work, we provide a more balanced evaluation by jointly examining detection performance and computational cost across both traditional and deep learning models. Further discussion of the scope and limitations for existing benchmarks is presented in the Appendix A.

3 EXPERIMENTAL DESIGN

3.1 PROBLEM DEFINITION

3.1.1 UNSUPERVISED ANOMALY DETECTION IN TIME SERIES

Unsupervised anomaly detection in multivariate time series is the task of identifying abnormal behaviors without access to anomaly labels during training (Pang et al., 2021). This setting is common in practice because anomalies are rare, labeling is costly (Blázquez-García et al., 2021).

A multivariate time series is defined as $x \in \mathbb{R}^{N \times D}$ and can be expressed as

$$X = \{x_t \in \mathbb{R}^D \mid t = 1, \ldots, N\} \tag{1}$$

where N denotes the sequence length and x_t is the D-dimensional observation at time t. Based on the learned representation of normal behaviors, the model provides a decision function that assigns each observation an anomaly score

$$s_t = F(x_t), \quad \{s_t\}_{t=1}^N \in \mathbb{R}^N \tag{2}$$

where observations with higher scores are regarded as anomalies, typically determined by calibrating a threshold from training scores (Su et al., 2019; Audibert et al., 2020; Xu et al., 2022).

The unsupervised formulation is particularly important in real-world applications, since annotated anomalies are typically unavailable, occur infrequently, or vary significantly across domains (Salehi & Rashidi, 2018). By modeling normality directly from unlabeled data, unsupervised approaches provide a practical and general framework for anomaly detection in time series.

3.1.2 RESEARCH QUESTIONS

Building on the above definition, we aim to investigate how unsupervised anomaly detection in multivariate time series can be evaluated not only in terms of effectiveness but also efficiency under realistic deployment settings. While prior work has primarily emphasized improving detection accuracy, comparatively less attention has been paid to the computational and operational feasibility of different methods. To address this gap, we define the following research questions:

(i) What are the most effective options for time series anomaly detection under limited computational resources, and are deep learning methods always the best options? This question is motivated by the observation that many deployment environments face practical constraints, including restricted computational capacity, memory, or energy availability. Although deep learning methods have shown strong benchmark performance, their dependence on substantial resources raises doubts about their universal applicability. It is therefore important to examine whether traditional statistical or machine learning methods may provide more practical alternatives under such constrained conditions.

(ii) Does a trade-off between detection performance and computational cost truly exist in practice? Deep learning models are generally associated with higher computational cost due to their larger architectures and resource requirements. The literature also shows that traditional statistical and machine learning methods can remain competitive in certain scenarios. This raises the question of whether such cost is justified by consistently superior detection performance. This question therefore seeks to clarify whether higher computational cost truly translates into superior performance, or whether certain approaches can offer a more balanced relationship that challenges the prevailing view.

3.2 EVALUATION PROTOCOL

3.2.1 CHOICE OF METHODS AND DATASETS

Algorithms We survey the historical development of unsupervised time series anomaly detection and curate a representative set of models for evaluation. The selection of models follow three principles. First, to ensure chronological coverage, we consider landmark contributions from the early multivariate statistical monitoring methods through contemporary deep learning methods, so that each major period is represented. Second, to reflect differences in operating mechanisms, we

partition the candidate methods into three families, namely statistical, one-class classification, and reconstruction-based, and we select representative exemplars from each family. Third, we focus on models that are well-established in the field and have been widely adopted in prior studies. Detailed descriptions of the included methods are provided in the Appendix B.1.

Datasets To support reproducible and deployment relevant evaluation, we prioritize datasets that capture real operational complexity and are widely used in the literature. We also focus on datasets from machinery and electronic system contexts where anomaly detection techniques are routinely deployed. Accordingly, we include SMD (Su et al., 2019) and PSM (Abdulaal et al., 2021) from server environments, SMAP and MSL from NASA engineering telemetry (Hundman et al., 2018), and SWaT (Mathur & Tippenhauer, 2016) and WADI (Ahmed et al., 2017) from industrial water treatment and distribution testbeds. Each dataset is multivariate and displays cross-channel dependencies and non-stationary dynamics, with rare anomalies as in real-world environments. Detailed descriptions of the included datasets are provided in the Appendix B.2.

3.2.2 METRICS

Detection Performance We compare detection performance across models using a threshold-agnostic metric, the Area Under the Receiver Operating Characteristic Curve (AUROC), to ensure fairness. Although certain methods are often paired with post hoc thresholding procedures such as Peaks Over Threshold, we do not apply such schemes to evaluate the intrinsic ranking quality of each model. To ensure a uniform basis of comparison, each implementation outputs a real-valued anomaly score at every time step, aligned with the original sequence length. AUROC is therefore computed at the same temporal resolution for all methods.

Computational Cost To enable a fair comparison of computational demands across both traditional algorithms and deep learning models, we adopt floating-point operations (FLOPs) as a unifying metric. FLOPs are model-agnostic and can be meaningfully related to hardware capabilities, making them particularly suitable for analyzing hardware-constrained scenarios. Unlike elapsed real time, FLOPs isolate algorithmic complexity from hardware variability. However, estimating FLOPs is nontrivial because the operations that dominate computational cost differ substantially across models and are highly sensitive to hyperparameter settings (e.g. tree depth in ensemble methods, number of neighbors in k-NN, or hidden dimension in neural networks). Consequently, prior work has rarely reported FLOPs for traditional machine learning methods, focusing only on deep learning models. We therefore derive closed-form or tight counting formulas for each traditional method and instantiate them with the exact hyperparameters used in our experiments. For deep learning models, we employed the PyTorch-based package `calflops`, which provides automatic FLOPs accounting given model architectures and input shapes (Ye, 2023).

In conducting FLOPs estimation, we adhere to the following principles:

- **Dataset specificity** FLOPs are computed separately for each dataset, as input dimensionality, sequence length, and sample size directly affect operation counts.
- **Training vs. inference** We compute FLOPs for both training and inference phases, reflecting their distinct computational characteristics.
- **Epoch sensitivity in deep learning models** Because training until convergence is ambiguous and dependent on hyperparameter optimization, we report FLOPs for a single epoch of training and the epoch at which the best AUROC is achieved during hyperparameter search.
- **Fair treatment of comparison operations** While FLOPs conventionally account only for addition and multiplication, algorithms such as k-NN and distance-based methods (ABOD, LOF) are dominated by comparison operations. Excluding these will unfairly understate their complexity, therefore, we count each comparison as a single FLOP.
- **Data-dependent structures** For models where computational complexity depends on data distribution such as clustering-based models, FLOPs are computed from the fitted model structure on the actual data rather than theoretical worst-case bounds.
- **Approximation for non-primitive operations** For procedures such as sorting, where exact FLOPs are impractical to enumerate, we adopt widely accepted complexity-based approximations (e.g. $O(n \log n)$, where n is the number of instances).

We summarize the final FLOPs formulations of the chosen methods in Table 2 and present their detailed derivations in Appendix C.

Table 2: Operation counts (FLOPs) for each method. We count additions, multiplications, and comparisons equally as 1 FLOP. The notation n_{tr}, n_{inf}, and d denote the number of training instances, the number of test instances, and the input dimension, respectively. Model-specific notations are explained in the Notation column.

Model	Type	FLOPs	Notation
Hotelling	Train	$2n_{\text{tr}}d^2 + 2n_{\text{tr}}d + d^3$	
	Inference	$n_{\text{inf}}(2d^2 + 2d - 1)$	
PCA	Train	$2n_{\text{tr}}d^2 + 2n_{\text{tr}}d + 3d^2$	• p: # of PCA components
	Inference	$n_{\text{inf}}(4pd - p + 2d - 1)$	
ABOD	Train	$1.5n_{\text{tr}}(n_{\text{tr}} - 1)d$ $+n_{\text{tr}}(n_{\text{tr}} - 1)\log_2(n_{\text{tr}} - 1)$ $+n_{\text{tr}}k(k-1)(d+2) + n_{\text{tr}}$	• k: # of neighbors
	Inference	$1.5n_{\text{inf}}(n_{\text{inf}} - 1)d$ $+n_{\text{inf}}(n_{\text{inf}} - 1)\log_2(n_{\text{inf}} - 1)$ $+n_{\text{inf}}k(k-1)(d+2) + n_{\text{inf}}$	
LOF	Train	$1.5n_{\text{tr}}(n_{\text{tr}} - 1)d$ $+n_{\text{tr}}(n_{\text{tr}} - 1)\log_2(n_{\text{tr}} - 1)$ $+n_{\text{tr}}k + n_{\text{tr}}(k+1) + 2n_{\text{tr}}k$	• k: # of neighbors
	Inference	$1.5n_{\text{inf}}(n_{\text{inf}} - 1)d$ $+n_{\text{inf}}(n_{\text{inf}} - 1)\log_2(n_{\text{inf}} - 1)$ $+n_{\text{inf}}k + n_{\text{inf}}(k+1) + 2n_{\text{inf}}k$	
CBLOF	Train	$n_{\text{tr}}I(3Cd + d - 1) + 3d((n_{\text{tr}} - \lvert LC\rvert)L + \lvert LC\rvert)$	• I: max iterations for clustering • C: # of clusters • L: # of large clusters • $\lvert LC\rvert$: # of instances in large clusters
	Inference	$n_{\text{inf}}I(3Cd + d - 1) + 3d((n_{\text{inf}} - \lvert LC\rvert)L + \lvert LC\rvert)$	
HBOS	Train	$2n_{\text{tr}}d + 5bd$	• b: # of bins
	Inference	$3n_{\text{inf}}d + 2bd$	
LODA	Train	$n_{\text{tr}}c(2\sqrt{d} + \log_2 b - 1)$	• b: # of bins • c: # of random cuts
	Inference	$n_{\text{inf}}(2c\sqrt{d} + c\log_2 b + 1)$	
Isolation Forest	Train	$T(2s\log_2 s)$	• T: # of estimators • s: max samples per estimator • γ: Euler–Mascheroni const.
	Inference	$n_{\text{inf}}(T(2\log s - 2 + \gamma) - 2(1 - 1/s) + (T + 2))$	
HS-Tree	Train	$T(\psi(h+1) + 5(2^{h+1} - 1))$	• T: # of estimators • h: max depth of tree • ψ: reference window size
	Inference	$n_{\text{inf}}T(5h + 7)$	

4 Results and Analyses

In this section we present three experiments to examine performance versus efficiency, estimated time comparison under hardware-constrained settings, and model scalability. The first experiment evaluates the trade-off between detection accuracy, measured by AUROC, and the computational cost quantified by training and inference FLOPs. For traditional models, training and inference FLOPs are given in Table 2. For deep learning models, we report training FLOPs per epoch, and full-training FLOPs correspond to the sum overall epochs. In subsequent analyses, we use full-training FLOPs for deep learning models. The second experiment leverages the fact that, compared to other efficiency metrics, FLOPs have the key advantage of being directly comparable to hardware performance, typically expressed as floating-point operations per second (FLOPS), where FLOPs denote algorithmic operation counts and FLOPS denote hardware throughput. Dividing algorithmic FLOPs by the FLOPS of a target hardware allows us to approximate training and inference time under GPU-free, hardware-constrained deployments. The third experiment examines scalability by

Table 3: Top five models for each dataset with AUROC and gigaFLOPs (GFLOPs). AUROC scores are reported as the mean over five runs with different random seeds.

Dataset	Model	Type	GFLOPs ↓			AUROC ↑
			Train	Inference	Full-training	
PSM	Hotelling	Statistical	0.17	0.11	–	0.77 ± 0.00
	LSTM-AE	Reconstruction	2.45	0.55	244.67	0.76 ± 0.01
	ABOD	Statistical	926.33	421.87	–	0.75 ± 0.00
	LOF	One-class	917.76	416.07	–	0.73 ± 0.00
	HBOS	Statistical	0.01	0.01	–	0.73 ± 0.00
MSL	CBLOF	One-class	0.89	2.60	–	0.65 ± 0.01
	HS-Tree	One-class	< 0.01	0.08	–	0.64 ± 0.03
	ABOD	Statistical	334.71	536.74	–	0.63 ± 0.00
	HBOS	Statistical	0.01	0.01	–	0.62 ± 0.00
	Isolation Forest	One-class	< 0.01	0.14	–	0.62 ± 0.01
SMAP	Isolation Forest	One-class	< 0.01	0.48	–	0.64 ± 0.01
	ABOD	Statistical	998.15	10281.98	–	0.64 ± 0.00
	LOF	One-class	996.76	10277.64	–	0.62 ± 0.00
	CBLOF	One-class	2.48	10.76	–	0.62 ± 0.01
	HBOS	Statistical	0.01	0.03	–	0.61 ± 0.00
SMD	LSTM-AE	Reconstruction	16.54	5.51	1653.89	0.77 ± 0.01
	Hotelling	Statistical	2.10	2.10	–	0.73 ± 0.00
	CBLOF	One-class	36.30	34.65	–	0.72 ± 0.01
	ABOD	Statistical	38426.94	38428.58	–	0.71 ± 0.00
	OmniAnomaly	Reconstruction	39.62	13.21	792.44	0.69 ± 0.02
SWaT	HBOS	Statistical	0.05	0.07	–	0.85 ± 0.00
	Isolation Forest	One-class	< 0.01	0.36	–	0.83 ± 0.00
	OmniAnomaly	Reconstruction	5.13	1.55	102.60	0.83 ± 0.00
	LODA	Statistical	1.39	1.39	–	0.82 ± 0.02
	PCA	Reconstruction	2.64	2.42	–	0.82 ± 0.00
WADI	HBOS	Statistical	0.19	0.06	–	0.74 ± 0.00
	Isolation Forest	One-class	< 0.01	0.33	–	0.74 ± 0.02
	LODA	Statistical	1.46	0.34	–	0.72 ± 0.04
	HS-Tree	One-class	< 0.01	0.20	–	0.63 ± 0.05
	OmniAnomaly	Reconstruction	71.70	5.26	1433.97	0.58 ± 0.00

varying the amount of data and observing how computational cost changes with dataset size. Model hyperparameters are chosen as those yielding the highest AUROC within the defined search space, with the detailed specification of the search space given in Appendix D.

4.1 Performance vs. Efficiency

Table 3 summarizes the main results, showing that the highest AUROC values are largely achieved by traditional models. Deep learning methods do not consistently surpass these baselines, and the performance gap is generally small. Notably, deep learning models appear only a few times among the top five performers overall, indicating that they are not dominant in terms of detection accuracy. Figure 1 further illustrates this trend by plotting AUROC against the total FLOPs, obtained by summing training and inference FLOPs, where the top five models from Table 3 are highlighted in orange. The highlighted points show that most of the best performing methods are traditional approaches, which achieve competitive or superior AUROC with substantially lower FLOPs, while deep learning models are generally concentrated on the higher cost side without clear performance gains. This consistent pattern across datasets underscores that the balance between effectiveness and

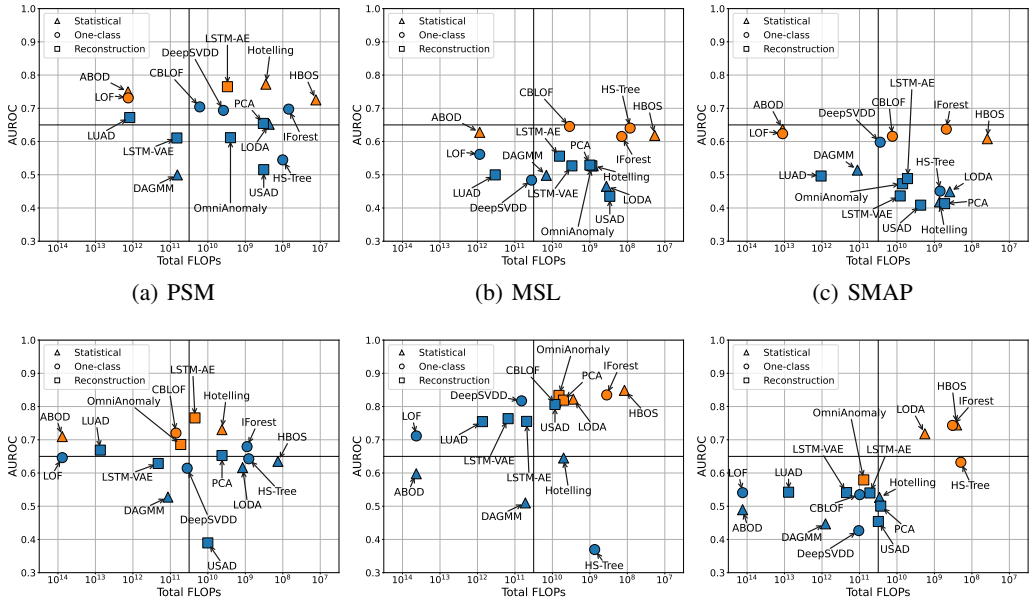


Figure 1: AUROC vs. Total FLOPs (sum of training and inference FLOPs) for each dataset. Orange markers denote the top five models in terms of AUROC.

efficiency often favors traditional models. Entire results for all models and datasets are provided in Appendix E.1.

4.2 Comparison of Estimated Time

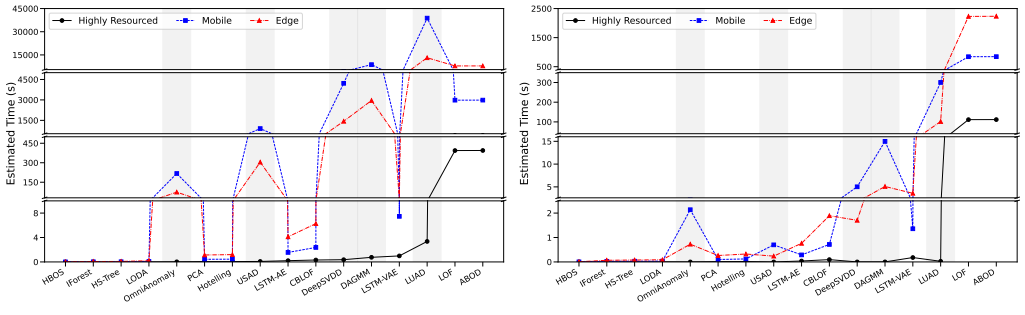


Figure 2: Estimated execution time of anomaly detection models under three hardware scenarios (Highly Resourced, Mobile, and Edge). Models are ordered in ascending runtime according to the Highly Resourced scenario, with deep learning models highlighted by gray shading. The results are averaged across datasets.

Estimate of the minimal execution time can be calculated as FLOPS/FLOPs. To capture variability across deployment conditions, we assume three hardware scenarios: a highly resourced environment represented by our experimental setup (Intel(R) Core(TM) i9-14900K CPU, NVIDIA GeForce RTX 5070 Ti GPU), a mobile environment corresponding to Samsung Galaxy A32 (Cortex-A75 & Cortex-A55 CPU, Arm Mali-G52 MC2 GPU), and a resource-constrained edge environment represented by Raspberry Pi 3B+ (Asutkar et al., 2023; Trilles et al., 2024). The FLOPS of each device

are computed using only the number of cores C and clock frequency f, as formalized in Equation 3.

$$\text{FLOPS}_{CPU}^{peak} = \sum_{t \in \{\text{Performance,Efficient}\}} C_t \times f_t, \quad \text{FLOPS}_{GPU}^{peak} = C \times f \tag{3}$$

A practically significant observation emerging from Figure 2 is that deep learning models are feasible only in highly resourced environments, where the abundance of GPU cores substantially mitigates their computational burden. In sharp contrast, under mobile or edge scenarios these models incur prohibitively high costs, making their deployment virtually infeasible. By comparison, tree-based algorithms such as Isolation Forest and HS-Tree, along with histogram-based algorithms such as HBOS and LODA consistently maintain extremely low computational overhead during both training and inference across all hardware settings. Finally, approaches that rely on k-NN, including ABOD and LOF, exhibit comparatively elevated costs, underscoring their limited practicality in data-abundant contexts. Results for individual datasets are provided in Appendix E.2.

4.3 Model Scalability

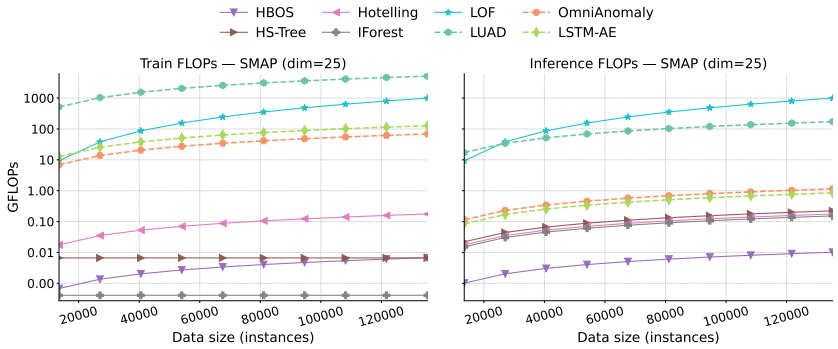


Figure 3: Training (left) and inference (right) FLOPs of eight representative models evaluated on the SMAP dataset. Both y-axes are presented on a logarithmic scale.

As illustrated in Figure 3, tree-based methods such as Isolation Forest and HS-Tree demonstrate high scalability, with computational requirements scaling sublinearly with respect to n, thereby making them well-suited for large-scale deployment. In contrast, methods using k-NN such as ABOD and LOF, exhibit a steep rise in both training and inference FLOPs, reflecting their quadratic dependence on n and highlighting their limited practicality for large datasets. Deep learning models exhibit a consistently linear increase in FLOPs with data size. Although they scale more favorably than k-NN approaches, their computational cost still poses a substantial burden as n becomes large. This divergence underscores the importance of considering algorithmic scalability when selecting anomaly detection models for real-world applications where data volumes can be substantial. Entire results for all models and datasets are provided in Appendix E.3.

5 Conclusion

This work presented a systematic comparison of traditional and deep learning methods for unsupervised time series anomaly detection under hardware-constrained settings. By jointly evaluating AUROC and FLOPs, we explored two central questions concerning the effectiveness of models under limited resources and whether and how performance differences arise between traditional and deep learning methods. Our results showed that traditional models often rank among the top performers, deep learning models do not consistently surpass them, and the balance between effectiveness and efficiency favors traditional approaches. Estimated time analysis further revealed that while deep learning models are feasible only in highly resourced environments, traditional models remain practical under resource-constrained settings while offering comparable detection performance. Overall, these findings challenge the view that deep learning is always the superior choice and emphasize the continued viability of traditional methods for real-world deployment. In doing so, we aim for this work to inspire new avenues of research while also providing practitioners with a useful point of reference when building anomaly detection systems under real-world constraints.

Reproducibility Statement

We have undertaken several efforts to ensure the reproducibility of our work. FLOPs derivations for all algorithms are provided in Appendix C, and the definition of hyperparameter search spaces is given in Appendix D. Extensive experimental results across datasets are presented in Appendix E. The supplementary material contains our full experimental pipeline, including model implementations, configuration files, and data preprocessing scripts, thereby facilitating independent reproduction of our findings.

References

Ahmed Abdulaal, Zhuanghua Liu, and Tomer Lancewicki. Practical approach to asynchronous multivariate time series anomaly detection and localization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pp. 2485–2494, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467174.

Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P. Mathur. Wadi: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, CySWATER '17, pp. 25–28, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349758. doi: 10.1145/3055366.3055375.

Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In C. V. Jawahar, Hongdong Li, Greg Mori, and Konrad Schindler (eds.), *Computer Vision – ACCV 2018*, pp. 622–637. Springer International Publishing, 2019. ISBN 978-3-030-20893-6.

Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen (eds.), *Principles of Data Mining and Knowledge Discovery*, pp. 15–27, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45681-0.

Supriya Asutkar, Chaitravi Chalke, Kajal Shivgan, and Siddharth Tallur. Tinyml-enabled edge implementation of transfer learning framework for domain generalization in machine fault diagnosis. *Expert Systems with Applications*, 213:119016, 2023. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.119016.

Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3395–3404, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403392.

Aitor Belenguer, Javier Navaridas, and Jose A. Pascual. A review of federated learning in intrusion detection systems for iot, 2022.

Deval Bhamare, Maede Zolanvari, Aiman Erbad, Raj Jain, Khaled Khan, and Nader Meskin. Cybersecurity for industrial control systems: A survey. *Computers & Security*, 89:101677, 2020. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.101677.

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*, pp. 443–448, 2007. doi: 10.1137/1.9781611972771.42.

Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A review on outlier/anomaly detection in time series data. *ACM Comput. Surv.*, 54(3), April 2021. ISSN 0360-0300. doi: 10.1145/3444690.

Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pp. 93–104, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335388.

Ayan Chatterjee and Bestoun S. Ahmed. Iot anomaly detection methods and applications: A survey. *Internet of Things*, 19:100568, 2022. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2022.100568.

Ronit Das and Tie Luo. Lightesd: Fully-automated and lightweight anomaly detection framework for edge computing. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 150–158, 2023. doi: 10.1109/EDGE60047.2023.00032.

Alireza Dehlaghi-Ghadim, Tijana Markovic, Miguel Leon, David Söderman, and Per Erik Strandberg. Federated learning for network anomaly detection in a distributed industrial environment. In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 218–225, 2023. doi: 10.1109/ICMLA58977.2023.00038.

Daniel Dobos, Tien Thanh Nguyen, Truong Dang, Allan Wilson, Helen Corbett, John McCall, and Phil Stockton. A comparative study of anomaly detection methods for gross error detection problems. *Computers & Chemical Engineering*, 175:108263, 2023. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2023.108263.

Jin Fan, Zhentao Liu, Huifeng Wu, Jia Wu, Zhanyu Si, Peng Hao, and Tom H. Luan. Luad: A lightweight unsupervised anomaly detection scheme for multivariate time series data. *Neurocomputing*, 557:126644, 2023. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.126644.

Daniel Fährmann, Malte Ihlefeld, Arjan Kuijper, and Naser Damer. Resource-efficient anomaly detection in industrial control systems with quantized recurrent variational autoencoder. *IET Collaborative Intelligent Manufacturing*, 7(1):e70032, 2025. doi: https://doi.org/10.1049/cim2.70032.

Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Tadgan: Time series anomaly detection using generative adversarial networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pp. 33–43, 2020. doi: 10.1109/BigData50022.2020.9378139.

Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track 9*, 2012.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 32142–32159. Curran Associates, Inc., 2022.

Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9):1641–1650, 2003. ISSN 0167-8655. doi: https://doi.org/10.1016/S0167-8655(03)00003-5.

H.Hotelling. Multivariate quality control illustrated by air testing of sample bombsights. *Techniques of statistical analysis*, pp. 111, 1947.

Thi Kieu Khanh Ho and Narges Armanfard. Contaminated multivariate time-series anomaly detection with spatio-temporal graph conditional diffusion models. In *The 41st Conference on Uncertainty in Artificial Intelligence*, 2025.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 387–395, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219845.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Prarthi Jain, Seemandhar Jain, Osmar R. Zaïane, and Abhishek Srivastava. Anomaly detection in resource constrained environments with streaming data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(3):649–659, 2022. doi: 10.1109/TETCI.2021.3070660.

Xudong Jia, Peng Xun, Wei Peng, Baokang Zhao, Haojie Li, and Chiran Shen. Deep anomaly detection for time series: A survey. *Computer Science Review*, 58:100787, 2025. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2025.100787.

Bedeuro Kim, Mohsen Ali Alawami, Eunsoo Kim, Sanghak Oh, Jeongyong Park, and Hyoungshick Kim. A comparative study of time series anomaly detection models for industrial control systems. *Sensors*, 23(3), 2023. ISSN 1424-8220. doi: 10.3390/s23031310.

Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 444–452, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581934. doi: 10.1145/1401890.1401946.

Max Landauer, Florian Skopik, Branka Stojanović, Andreas Flatscher, and Torsten Ullrich. A review of time-series analysis for cyber security analytics: from intrusion detection to attack prediction. *International Journal of Information Security*, 24(1):3, 2025.

Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, pp. 703–716, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30490-4.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008. doi: 10.1109/ICDM.2008.17.

Hou-I Liu, Marco Galindo, Hongxia Xie, Lai-Kuan Wong, Hong-Han Shuai, Yung-Hui Li, and Wen-Huang Cheng. Lightweight deep learning for resource-constrained environments: A survey. *ACM Comput. Surv.*, 56(10), June 2024a. ISSN 0360-0300. doi: 10.1145/3657282.

Qinghua Liu and John Paparrizos. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 108231–108261. Curran Associates, Inc., 2024.

Zhe Liu, Xiang Huang, Jingyun Zhang, Zhifeng Hao, Li Sun, and Hao Peng. Multivariate time-series anomaly detection based on enhancing graph attention networks with topological analysis. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, pp. 1555–1564, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400704369. doi: 10.1145/3627673.3679614.

Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148, 2016.

Aditya P. Mathur and Nils Ole Tippenhauer. Swat: a water treatment testbed for research and training on ics security. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pp. 31–36, 2016. doi: 10.1109/CySWater.2016.7469060.

Nesryne Mejri, Laura Lopez-Fuentes, Kankana Roy, Pavel Chernakov, Enjie Ghorbel, and Djamila Aouada. Unsupervised anomaly detection in time-series: An extensive evaluation and analysis of state-of-the-art methods. *Expert Systems with Applications*, 256:124922, 2024. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2024.124922.

Roland N. Mfondoum, Antoni Ivanov, Pavlina Koleva, Vladimir Poulkov, and Agata Manolova. Outlier detection in streaming data for telecommunications and industrial applications: A survey. *Electronics*, 13(16), 2024. ISSN 2079-9292. doi: 10.3390/electronics13163339.

Periasamy Nancy, S. Muthurajkumar, S. Ganapathy, S.V.N. Santhosh Kumar, M. Selvi, and Kannan Arputharaj. Intrusion detection using dynamic feature selection and fuzzy temporal decision tree classification for wireless sensor networks. *IET Communications*, 14:888–895, 2020. doi: 10.1049/iet-com.2019.0172.

Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), March 2021. ISSN 0360-0300. doi: 10.1145/3439950.

John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S. Tsay, Themis Palpanas, and Michael J. Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proc. VLDB Endow.*, 15(8):1697–1711, April 2022. ISSN 2150-8097. doi: 10.14778/3529337.3529354.

Daehyung Park, Yuuna Hoshi, and Charles C. Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018. doi: 10.1109/LRA.2018.2801475.

Tomáš Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

Xiangfei Qiu, Zhe Li, Wanghui Qiu, Shiyan Hu, Lekui Zhou, Xingjian Wu, Zhengyu Li, Chenjuan Guo, Aoying Zhou, Zhenli Sheng, Jilin Hu, Christian S. Jensen, and Bin Yang. TAB: Unified benchmarking of time series anomaly detection methods. In *Proc. VLDB Endow.*, 2025.

Ferdinand Rewicki, Joachim Denzler, and Julia Niebling. Is it worth it? comparing six deep and classical methods for unsupervised anomaly detection in time series. *Applied Sciences*, 13(3), 2023. ISSN 2076-3417. doi: 10.3390/app13031778.

Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4393–4402. PMLR, 10–15 Jul 2018.

Mahsa Salehi and Lida Rashidi. A survey on anomaly detection in evolving data: [with application to forest fire risk prediction]. *SIGKDD Explor. Newsl.*, 20(1):13–23, May 2018. ISSN 1931-0145. doi: 10.1145/3229329.3229332.

Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, May 2022. ISSN 2150-8097. doi: 10.14778/3538598.3538602.

Md. Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2023. doi: 10.1109/JPROC.2022.3226481.

Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. *Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering*, 2003.

Haotian Si, Jianhui Li, Changhua Pei, Hang Cui, Jingwen Yang, Yongqian Sun, Shenglin Zhang, Jingjing Li, Haiming Zhang, Jing Han, Dan Pei, and Gaogang Xie. Timeseriesbench: An industrial-grade benchmark for time series anomaly detection models. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 61–72, 2024. doi: 10.1109/ISSRE62328.2024.00017.

Raghubir Singh and Sukhpal Singh Gill. Edge ai: A survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023. ISSN 2667-3452. doi: https://doi.org/10.1016/j.iotcps.2023.02.004.

Tuomo Sipola, Janne Alatalo, Tero Kokkonen, and Mika Rantonen. Artificial intelligence in the iot era: A review of edge ai hardware and software. In *2022 31st Conference of Open Innovations Association (FRUCT)*, pp. 320–331, 2022. doi: 10.23919/FRUCT54823.2022.9770931.

Keith Stouffer, Keith Stouffer, Michael Pease, CheeYee Tang, Timothy Zimmerman, Victoria Pillitteri, Suzanne Lightman, Adam Hahn, Stephanie Saravia, Aslam Sherule, et al. Guide to operational technology (ot) security. *National Institute of Standards and Technology*, 2023.

Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pp. 2828–2837, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330672.

Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pp. 1511–1516. AAAI Press, 2011. ISBN 9781577355144.

Sergio Trilles, Sahibzada Saadoon Hammad, and Ditsuhi Iskandaryan. Anomaly detection based on artificial intelligence of things: A systematic literature mapping. *Internet of Things*, 25:101063, 2024. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2024.101063.

Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad: deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6):1201–1214, February 2022. ISSN 2150-8097. doi: 10.14778/3514061.3514067.

Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022.

Zheng Xu, Yumeng Yang, Xinwen Gao, and Min Hu. Dcff-mtad: A multivariate time-series anomaly detection model based on dual-channel feature fusion. *Sensors*, 23(8), 2023. ISSN 1424-8220. doi: 10.3390/s23083910.

Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001. doi: https://doi.org/10.1002/qre.392.

Xiaoju Ye. Calflops: A flops and params calculate tool for neural networks in pytorch framework, 2023.

Zahra Zamanzadeh Darban, Geoffrey I. Webb, Shirui Pan, Charu Aggarwal, and Mahsa Salehi. Deep learning for time series anomaly detection: A survey. *ACM Comput. Surv.*, 57(1), October 2024. ISSN 0360-0300. doi: 10.1145/3691338.

Dequan Zheng, Fenghuan Li, and Tiejun Zhao. Self-adaptive statistical process control for anomaly detection in time series. *Expert Systems with Applications*, 57:324–336, 2016. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2016.03.029.

Bin Zhou, Shenghua Liu, Bryan Hooi, Xueqi Cheng, and Jing Ye. Beatgan: anomalous rhythm detection using adversarially generated time series. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pp. 4433–4439. AAAI Press, 2019. ISBN 9780999241141.

Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018.

A Previous Benchmarks

Table 4: Previous benchmark papers considering computational cost.

Paper Type	Paper	Cost Metric	Coverage Model Type			Data Source	Dimension
			Stat.	ML	DL	Real	Multi
Benchmark	Schmidl et al. (2022)	Memory, Time	✓	✓	✓	✓	✓
	Paparrizos et al. (2022)	Time	✓	✓	✓	✓	✗
	Han et al. (2022)	Time	✓	✓	✓	✓	✓
	Dobos et al. (2023)	Time	✓	✓	✓	✗	✓
	Rewicki et al. (2023)	Time	✓	✓	✓	✓	✗
	Liu & Paparrizos (2024)	Time	✓	✓	✓	✓	✓
	Si et al. (2024)	Time	✓	✗	✓	✓	✓
	Qiu et al. (2025)	Memory, Time	✓	✓	✓	✗	✓
Methodology	Xu et al. (2023)	Params, FLOPs, Time	✗	✗	✓	✓	✓
	Liu et al. (2024b)	Params, FLOPs, Time	✗	✗	✓	✓	✓
	Ho & Armanfard (2025)	FLOPs, Time	✗	✗	✓	✓	✓
	Ours	**FLOPs**	✓	✓	✓	✓	✓

As summarized in Table 4, a number of benchmark studies on multivariate time series anomaly detection have undertaken extensive comparisons across models and datasets. Despite this broad of coverage, the evaluation of computational cost in these works remains limited in scope. Most benchmarks have relied primarily on execution time as the cost metric, with a few additionally considering memory usage. However, runtime measurements are inherently dependent on hardware specifications and experimental settings, which constrains their comparability across studies. Likewise, memory consumption does not fully capture the algorithmic complexity of the models and therefore provides only a partial view of computational efficiency.

Beyond benchmark papers, some methodology studies have employed hardware-agnostic measures such as parameter counts or FLOPs. However, these comparisons have typically been restricted to deep learning models, leaving traditional statistical and machine learning approaches unexamined even though they have compatible detection performance with deep learning models. To bridge this gap, we adopt FLOPs, a hardware-agnostic metric, and apply it to both traditional and deep learning models. This unified treatment enables fair and reproducible comparisons of computational cost across paradigms and provides practitioners with a principled basis for model selection in resource-constrained environments.

B Algorithms and Datasets

B.1 Algorithms

Hotelling (H.Hotelling, 1947). A multivariate statistical process control method that scores each observation via its Mahalanobis distance under a Gaussian reference model, thereby capturing correlated variation across variables.

PCA (Shyu et al., 2003). Principal Component Analysis models normal structure in a low-dimensional subspace. Deviations are quantified through reconstruction error. Anomalies arise when observations project poorly onto the principal subspace.

ABOD (Kriegel et al., 2008). Angle-Based Outlier Detection ranks points by the variance of angles formed with all other points, exploiting the geometric insight that outliers yield concentrated angle distributions in high dimensions. Practical variants use subsampling or k-nearest neighborhoods to reduce the quadratic cost while preserving discrimination.

LOF (Breunig et al., 2000). Local Outlier Factor contrasts a point's local reachability density with that of its neighbors to assess how isolated it is within its immediate neighborhood. Large LOF

scores indicate locally sparse regions, enabling detection of context-dependent anomalies that global density models often miss.

CBLOF (He et al., 2003). Cluster-Based LOF assigns each point to a large or small cluster and computes scores from cluster size and distance to representative large clusters. Points in small, distant clusters receive high scores, capturing both rarity and separation.

HBOS (Goldstein & Dengel, 2012). Histogram-Based Outlier Score approximates feature-wise densities with univariate histograms and aggregates inverse densities across dimensions, implicitly assuming weak dependence.

LODA (Pevnỳ, 2016). The Lightweight online Detector of Anomalies ensembles sparse random projections, building one-dimensional histograms in each projected space and combining their anomaly evidences.

Isolation Forest (Liu et al., 2008). Random partitioning via isolation trees isolates anomalies with fewer splits, producing shorter expected path lengths than normal points. Scores are obtained by normalizing path lengths against the average in random trees, enabling fast, distribution-agnostic detection.

HS-Tree (Tan et al., 2011). Half-Space Trees construct randomized, axis-aligned partitions geared for streaming one-class detection. Points that consistently fall into underpopulated half-spaces obtain higher anomaly scores.

DAGMM (Zong et al., 2018). The Deep Autoencoding Gaussian Mixture Model jointly learns a compact representation and a GMM density in an end-to-end fashion, combining reconstruction features with mixture-based energy for scoring. This coupling allows the representation to align with density estimation, improving separability of rare patterns.

DeepSVDD (Ruff et al., 2018). A deep one-class objective trains a network to map normal data into a minimal radius hypersphere in latent space, penalizing distances from a fixed center. Samples that lie far from this center at test time are flagged as anomalies, avoiding reconstruction bias inherent to autoencoders.

LSTM-AE (Malhotra et al., 2016). A sequence-to-sequence LSTM autoencoder learns normal temporal dynamics and emits reconstruction errors over sliding windows. Sustained or abrupt increases in error indicate departures from learned patterns, capturing both gradual drifts and transient spikes.

LSTM-VAE (Park et al., 2018). A variational sequence model with LSTM encoder-decoder estimates a probabilistic generative process, enabling anomaly scoring via low evidence lower bound or high reconstruction error.

USAD (Audibert et al., 2020). A dual-autoencoder architecture trained with an adversarial-inspired objective where two decoders reconstruct each other's outputs to improve robustness. At inference, a calibrated combination of the two reconstruction errors yields stable anomaly scores with strong generalization across regimes.

OmniAnomaly (Su et al., 2019). A stochastic recurrent VAE augmented with normalizing flows models complex temporal dependencies and heteroscedastic noise, producing likelihood-based anomaly scores, negative log-probability, its latent dynamics capture both short and long range dependencies.

LUAD (Fan et al., 2023). A lightweight unsupervised detector that combines efficient temporal encoder, TCN, with a compact probabilistic module and an auxiliary diagnosis head.

Table 5: Dataset overview.

Dataset	Dimensions	Train size	Test size	Anomaly Ratio (%)
PSM	25	129784	87841	27.76
MSL	55	58317	73729	10.53
SMAP	25	135183	427617	12.79
SMD	38	708405	708420	4.16
SWaT	51	496800	449919	12.14
WADI	123	784537	172801	5.77

B.2 DATASETS

Table 5 summarizes the key statistics of the benchmark datasets, including the input dimension, the size of the train set, the size of the test set, and the anomaly ratio. Note that anomalies appear only in test sets, and the reported ratios are calculated with respect to the test instances. In addition to these summary statistics, we provide a description of the key characteristics of each dataset below.

PSM (Abdulaal et al., 2021). The Pooled Server Metrics dataset consists of multivariate time series monitoring server behavior, including signals such as CPU utilization and memory usage. It contains 13 weeks of training data and 8 weeks of testing data. Anomalies are present in both splits, while labels are provided only for the test set and include both planned and unplanned events.

MSL (Hundman et al., 2018). The Mars Science Laboratory dataset was constructed from telemetry of NASA's Curiosity rover. Anomalies were extracted from Incident Surprise, Anomaly reports (ISA) and manually labeled across channels.

SMAP (Hundman et al., 2018). The Soil Moisture Active Passive dataset was derived from telemetry collected during NASA's satellite mission. Anomalies were identified through ISA, which document unexpected spacecraft events during post-launch operations.

SMD (Su et al., 2019). The Server Machine Dataset is a 5 week multivariate time series collection gathered from a large Internet company. It comprises logs with metrics such as CPU load, memory usage, disk activity, and network traffic. The dataset is partitioned into training and testing halves, with anomalies in the testing portion labeled by domain experts based on incident reports.

SWaT (Mathur & Tippenhauer, 2016). The Secure Water Treatment dataset was collected from a fully operational 6 stage water treatment testbed. It comprises readings from sensors and actuators recorded every second over 11 consecutive days, including 7 days of normal operation and 4 days with controlled cyber-physical attacks. All attack instances were labeled by experts.

WADI (Ahmed et al., 2017). The Water Distribution dataset is derived from a scaled-down water distribution network testbed that simulates real industrial control systems. It contains multivariate time series of sensor and actuator signals across different stages of water storage and distribution. The dataset includes normal operations as well as periods with cyber-physical attack scenarios, with labels provided for the anomalous events.

C DERIVATIONS OF FLOPS

This section provides a detailed account of the FLOPs computations for both traditional and deep learning models. For traditional models, FLOPs are derived from the algorithmic procedures, with training and inference costs obtained by applying the formulas to the entire training and test sets, respectively. In contrast, deep learning models operate on sequences generated by a rolling window. To ensure consistency, all calculations used the same input shape, determined by the window length

and feature dimension of each dataset. The FLOPs of deep learning models were computed using the `calflops` package, applied to our predefined model architectures. Training FLOPs were estimated as the sum of forward and backward passes, whereas inference FLOPs were measured from the forward pass alone. Thus, training FLOPs are calculated as the operations required to process one epoch of windowed training data with both forward and backward passes, while inference FLOPs correspond to a forward pass over the windowed test set. The full-training FLOPs are then obtained by multiplying the training FLOPs per epoch by the number of epochs used for each model. For comparability across models, all FLOPs are expressed in GFLOPs.

Global notation and counting rule Given a common vector $x \in \mathbb{R}^d$, we use following notations: n_{tr} denotes the number of training instances, n_{inf} the number of inference instances, and d the input dimensionality. Also, we denote FLOPs per sample by f, and the total FLOPs over the dataset by F. If an algorithm supports per-sample inference, we report both f and F. For models that do not involve a distinct training phase, we set $n = n_{\mathrm{tr}} = n_{\mathrm{inf}}$.

Additions, multiplications, divisions, and comparisons are all counted as 1 FLOP, and the FLOPs required for matrix multiplication between $A \in \mathbb{C}^{M\times N}$ and $B \in \mathbb{C}^{N\times L}$ are calculated as

$$F_{AB} = 2MNL - ML$$

which consists of MNL multiplication and $ML(N-1)$ additions.

C.1 HOTELLING

Training FLOPs Hotelling's statistic is

$$T^2 = (x-\mu)^\top \Sigma^{-1}(x-\mu),$$

where $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d\times d}$ are mean and covariance of the training set. In the training phase, the inverse covariance matrix is calculated.

(i) *Mean*: $\mu = \frac{1}{n_{\mathrm{tr}}} \sum_i x_i$ costs

$$F_\mu = (n_{\mathrm{tr}} - 1)d + d = n_{\mathrm{tr}} d.$$

(ii) *Covariance*: $\Sigma = \frac{1}{n_{\mathrm{tr}}-1} \sum_{i=1}^{n_{\mathrm{tr}}} (x_i - \mu)(x_i - \mu)^\top$ costs

$$\begin{aligned} F_\Sigma &= n_{\mathrm{tr}} d + n_{\mathrm{tr}} d^2 + (n_{\mathrm{tr}} - 1)d^2 + d^2 \\ &= n_{\mathrm{tr}}(2d^2 + d) \end{aligned}$$

with respect to subtraction, multiplication, addition, and division.

(iii) *Inverse*: Inverting Σ costs

$$F_{\Sigma^{-1}} \approx d^3.$$

Summing up,

$$\begin{aligned} F^{\mathrm{train}} &= F_\mu + F_\Sigma + F_{\Sigma^{-1}} \\ &= 2n_{\mathrm{tr}} d^2 + 2n_{\mathrm{tr}} d + d^3. \end{aligned}$$

Inference FLOPs In the inference phase, the monitoring statistics T^2 calculations are performed sample by sample.

(i) *Centering subtraction*: $v = x - \mu$ costs

$$f_{\mathrm{cen}} = d.$$

(ii) *Matrix multiplication*: $T^2 = v^\top \Sigma^{-1} v$ costs

$$f_{\mathrm{mat}} = 2d^2 + d - 1.$$

Therefore,

$$\begin{aligned} f^{\mathrm{infer}} &= f_{\mathrm{cen}} + f_{\mathrm{mat}} = 2d^2 + 2d - 1, \\ F^{\mathrm{infer}} &= n_{\mathrm{inf}}(2d^2 + 2d - 1). \end{aligned}$$

C.2 PCA

Model-specific notation p: number of principal components retained ($p \leq d$).

Training FLOPs We estimate the mean and covariance exactly as in section C.1, and then obtain the top-p eigenspace with QR decomposition.

(i) *Mean*: $\mu = \frac{1}{n_{\text{tr}}} \sum_i x_i$ costs

$$F_\mu = (n_{\text{tr}} - 1)d + d = n_{\text{tr}} d.$$

(ii) *Covariance*: $\Sigma = \frac{1}{n_{\text{tr}}-1} \sum_i^{n_{\text{tr}}} (x_i - \mu)(x_i - \mu)^\top$ costs

$$F_\Sigma \approx n_{\text{tr}}(2d^2 + d).$$

(iii) *QR decomposition of matrix*: Computing QR decomposition costs

$$F_{\text{dec}} \approx 3d^2.$$

Summing up,

$$\begin{aligned} F^{\text{train}} &= F_\mu + F_\Sigma + F_{\text{dec}} \\ &= 2n_{\text{tr}} d^2 + 2n_{\text{tr}} d + 3d^2. \end{aligned}$$

Inference FLOPs In the inference phase, reconstructions can be performed sample by sample. Given $x \in \mathbb{R}^d$, let $U_p \in \mathbb{R}^{d \times p}$ be the loading matrix.

(i) *Projection*: $z = U_p^\top x$ costs

$$f_{\text{proj}} = p(2d - 1).$$

(ii) *Reconstruction*: $\hat{x} = U_p z$ costs

$$f_{\text{rec}} = d(2p - 1).$$

(iii) *Error calculation*: $e = \|x - \hat{x}\|^2 = \sum_{j=1}^d (x_j - \hat{x}_j)^2$ costs

$$f_{\text{err}} = 3d - 1.$$

Therefore, for each instance and total n_{inf} instances,

$$\begin{aligned} f^{\text{infer}} &= f_{\text{proj}} + f_{\text{err}} + f_{\text{rec}} = 4pd - p + 2d - 1, \\ F^{\text{infer}} &= n_{\text{inf}}(4pd - p + 2d - 1). \end{aligned}$$

C.3 ABOD

Model-specific notation k: number of neighbors for Fast-ABOD.

Training/Inference FLOPs We first prepare k-NN neighborhoods, then evaluate the Angle Based Outlier Factor (ABOF) score for a point using its k neighbors. This is Fast-ABOD which approximate original ABOD.

(i) *All-pairs of Euclidean distances*: Computing $\|x_i - x_j\|_2 = \sqrt{\sum_{l=1}^d (x_{il} - x_{jl})^2}$ costs $3d$. If span this to all-pairs, it costs

$$F_{\text{dist}} = \binom{n}{2}(3d) = \frac{3}{2} n(n-1)d.$$

(ii) *Sorting distances*: Sorting algorithms have approximated complexity of $O(n \log_2 n)$. Therefore, sorting all-pair distances costs

$$f_{\text{sort}} \approx (n-1)\log_2(n-1), \quad F_{\text{sort}} \approx n(n-1)\log_2(n-1).$$

(iii) *ABOF calculation*: ABOF is calculated by $\text{VAR}_{B,C \in N_k(A)}\left(\frac{\langle \overline{AB}, \overline{AC} \rangle}{\|\overline{AB}\|^2 \|\overline{AC}\|^2}\right)$ and $\binom{k}{2} = \frac{1}{2}k(k-1)$ is the number of neighbor pair cases.

Reusing calculated distances, each pair needs one dot product and operations for multiplication and normalization. Therefore, each pair needs $2d + 1$ FLOPs and spans $\frac{1}{2}k(k-1)$ times within one sample.

With $\frac{1}{2}k(k-1)$ pairs, VAR $= \frac{1}{N}\sum v^2 - \left(\frac{1}{N}\sum v\right)^2$ costs $\frac{3}{2}k(k-1)+1$.

Then,

$$f_{\text{abof}} = \frac{1}{2}k(k-1)(2d+1) + \frac{3}{2}k(k-1) + 1 = k(k-1)(d+2) + 1$$

and

$$F_{\text{abof}} = nk(k-1)(d+2) + n.$$

Summing up,

$$\begin{aligned} F^{\text{train/infer}} &= F_{\text{dist}} + F_{\text{sort}} + F_{\text{abof}} \\ &= 1.5n(n-1)d + n(n-1)\log_2(n-1) + nk(k-1)(d+2) + n. \end{aligned}$$

C.4 LOF

Model-specific notation k: number of neighbors.

Training/Inference FLOPs We build k-NN neighborhoods and then compute reachability distances, local reachability density, and the LOF score.

(i) *All-pairs of Euclidean distances*: As calculated at Section C.3, it costs

$$F_{\text{dist}} = \binom{n}{2}(3d) = \frac{3}{2}n(n-1)d.$$

(ii) *Sorting distances*: Sorting algorithms have approximated complexity of $O(n\log_2 n)$. Therefore, sorting all-pair distances costs

$$f_{\text{sort}} \approx (n-1)\log_2(n-1), \quad F_{\text{sort}} \approx n(n-1)\log_2(n-1).$$

(iii) *Reachability distances*: For each point x_p and $x_o \in N_k(x_p)$, comparison operation $\text{reach\_dist}(x_p, x_o) = \max\{\text{dist}_k(x_o),\, \text{dist}(x_p, x_o)\}$ is conducted. Total comparison costs

$$f_{\text{reach}} = k, \quad F_{\text{reach}} = nk.$$

(iv) *Local reachability density (LRD)*: Formulation of LRD is

$$\text{LRD}(x_p) = \left(\frac{1}{N_k(x_p)}\sum_{x_o \in N_k(x_p)} \text{reach\_dist}(x_p, x_o)\right)^{-1}.$$

Per point, $k-1$ additions, 1 division, and 1 scaling is conducted with total $k+1$ FLOPs.

Therefore,

$$f_{\text{lrd}} = k+1, \quad F_{\text{lrd}} = n(k+1).$$

(v) *LOF score*: Formulation of LOF score is

$$\text{LOF}(x_p) = \frac{1}{k}\sum_{x_o \in N_k(x_p)} \frac{\text{LRD}(x_o)}{\text{LRD}(x_p)}.$$

Per point, k divisions, $k-1$ additions, 1 division is conducted with total $2k$ FLOPs.

Therefore,

$$f_{\text{lof}} = 2k, \quad F_{\text{lof}} = 2nk.$$

Summing up,

$$\begin{aligned} F^{\text{train/infer}} &= F_{\text{dist}} + F_{\text{sort}} + F_{\text{reach}} + F_{\text{lrd}} + F_{\text{lof}} \\ &= 1.5n(n-1)d + n(n-1)\log_2(n-1) + nk + n(k+1) + 2nk. \end{aligned}$$

C.5 CBLOF

Model-specific notation I: maximum k-means iterations, C: number of clusters, L: number of large clusters, $|LC|$: number instances in large clusters.

Training/Inference FLOPs CBLOF fits C centroids with k-means and then scores samples using the large-small cluster partition.

(i) *K-means cluster assignment*: For each sample $x \in \mathbb{R}^d$ and each centroid c, squared distance $\|x-c\|^2$ costs $3d-1$ FLOPs. Also, finding minimum distance centroid over C centroids contributes $C-1$ comparisons.

Over n points and one iteration, it costs

$$F_{\text{assign/iter}} = n\left[C(3d-1) + (C-1)\right].$$

(ii) *K-means centroid update*: For each centroid, we accumulate assigned points and normalize once. As we have n points and C centroids, accumulation costs $d(n-C)$ and normalization costs Cd making total FLOPs for centroid update is nd.

Over n points and one iteration, it costs

$$F_{\text{update/iter}} = nd.$$

Therefore,

$$\begin{aligned} F_{\text{kmeans}} &= I(F_{\text{assign/iter}} + F_{\text{update/iter}}) \\ &= I(3Cdn - n + nd). \end{aligned}$$

(iii) *Scoring with large/small partition*: For each point x_p, score is computed as

$$Score(x_p) = \begin{cases} |C_i| \times \min_{j \in LC} \text{dist}(x_p, C_j), & \text{if } x_p \in C_i, C_i \in SC \text{ and } C_j \in LC \\ |C_i| \times \text{dist}(x_p, C_i), & \text{if } x_p \in C_i, \text{ and } C_i \in LC \end{cases}$$

where C_i denotes i^{th} cluster and $|C_i|$ denotes the number of points in each cluster. Also, $|LC|$ is the number of instances that belong to large clusters and $|SC|$ is the number of instances that belong to small clusters, formally, $|SC| = n - |LC|$.

If $p \in SC$: As L large clusters exist, calculating distances to L cluster centroid costs $L(3d-1)$ and comparing costs $L-1$. Therefore, each point in small cluster need $3dL$ FLOPs, including multiplication operation of cluster size.

If $p \in LC$: Calculating distances to their own centroid costs $3d-1$ and multiplication costs 1 FLOPs. Therefore, each point in large cluster need $3d$ FLOPs.

Total scoring FLOPs for all n samples is

$$\begin{aligned} F_{\text{score}} &= 3d(|SC| \cdot L + |LC|) \\ &= 3d\big((n - |LC|)\, L + |LC|\big). \end{aligned}$$

Combining k-means and scoring,

$$F^{\text{train/infer}} = nI(3Cd + d - 1) + 3d((n - |LC|)L + |LC|).$$

C.6 HBOS

Model-specific notation b: number of bins per feature.

Training FLOPs The training cost consists of histogram construction. Each sample-feature value is assigned to a bin with one subtraction and one division, giving $2n_{\text{tr}}d$ FLOPs in total. Converting counts to densities, computing bin widths, and performing the normalization check together require $5bd$ FLOPs.

Thus,

$$F^{\text{train}} = 2n_{\text{tr}}d + 5bd.$$

Inference FLOPs At inference time, the score of each sample is computed based on the histograms. Each feature requires the computation of $\log_2(\text{hist} + \alpha)$, which costs 2 FLOPs per bin and yields $2bd$ FLOPs in total. For every sample and feature, assigning the score with boundary checks adds about 3 FLOPs, giving $3n_{\text{inf}}d$ FLOPs.

Therefore,

$$F^{\text{infer}} = 3n_{\text{inf}}d + 2bd.$$

C.7 LODA

Model-specific notation b: number of bins, c: number of random projections.

Training FLOPs The training cost consists of sparse random projections and histogram construction, covering both the computation of projected values and the assignment of sample to bins for density estimation.

(i) *Sparse projection*: Each projection vector has $\sqrt{d}$ nonzero entries. Computing one projection value $z_{ij} = x_j^\top w_i$ requires $2\sqrt{d} - 1$ FLOPs. With n_{tr} training samples and c projections, the cost is

$$F_{\text{proj}} = n_{\text{tr}}c(2\sqrt{d} - 1).$$

(ii) *Histogram construction*: Each projected value must be assigned to a histogram bin. Using binary search over the b bin edges requires $\log_2 b$ comparisons per assignment. The cost is therefore

$$F_{\text{bin}} = n_{\text{tr}}c \log_2 b.$$

Summing up, the training FLOPs are

$$\begin{aligned} F^{\text{train}} &= F_{\text{proj}} + F_{\text{bin}} \\ &= n_{\text{tr}}c(2\sqrt{d} + \log_2 b - 1). \end{aligned}$$

Inference FLOPs During inference, each sample is projected onto the c, its bin is determined, and the corresponding density values are used to compute the anomaly score.

(i) *Sparse projection*: Each projection requires $2\sqrt{d} - 1$ FLOPs, and the total projection cost over all histogram is

$$f_{\text{proj}} = c(2\sqrt{d} - 1).$$

(ii) *Bin lookup and score computation*: For each projection, locating the appropriate bin via binary search and computing log-density with accumulation require $c \log_2 b + 2$ FLOPs. Over all projections this becomes

$$f_{\text{binscore}} = c(\log_2 b + 2).$$

Therefore, including the final division for averaging across projections, the inference cost is

$$\begin{aligned} f^{\text{infer}} &= f_{\text{proj}} + f_{\text{binscore}} \\ &= 2c\sqrt{d} + c \log_2 b + c + 1, \end{aligned}$$

$$F^{\text{infer}} = n_{\text{inf}}(2c\sqrt{d} + c \log_2 b + c + 1).$$

C.8 Isolation Forest

Model-specific notation T: number of trees, s: max samples per tree, γ: Euler-Mascheroni constant ($\gamma \approx 0.5772$).

Training FLOPs Each tree is grown on a random subsample of size s. At each internal node, we pick a random feature, sample a split value within the feature's range, and partition the instances by comparison. Let n_l denote the expected number of samples at a node in level l. At a level l node, computing feature's range costs n_l, and partitioning the instances costs n_l, since the tree is binary.

Also, we approximate the tree height as $h \approx \log_2 s$, the number of nodes in level l as 2^l, and number of samples processed in a level l node as $n_l \approx \frac{s}{2^l}$. Therefore, computation at each node in level l costs $2 \times \frac{s}{2^l}$. The total cost for a single tree is

$$\sum_{l=0}^{h-1}(2 \times \frac{s}{2^l} \times 2^l) = 2sh = 2s \log_2 s.$$

For T trees the training FLOPs are approximated as

$$F^{\text{train}} = T(2s \log_2 s).$$

Inference FLOPs A sample is routed from the root to a leaf in every tree. The expected path length $c(s)$ for subsample size s is presented by authors of Isolation Forest (Liu et al., 2008).

$$c(s) = 2H_{s-1} - \frac{2(s-1)}{s} \approx 2(\ln(s-1) + \gamma) - 2 + \frac{2}{s}$$

Each step down the tree costs one comparison, hence costs per sample across T trees are $T \cdot c(s)$.

Therefore,

$$f_{\text{step\_down}} = T \cdot c(s) = T\left[2\{\ln(s-1) + \gamma\} - 2(1 - \frac{1}{s})\right]$$

Anomaly score of the model is calculated by $2^{-\frac{\mathbb{E}(h(x))}{c(s)}}$.

Score calculation is performed by aggregating path lengths across the T trees. This requires $T-1$ additions, followed by a normalization step that introduces two more scalar operations, division and exponentiation. We thus fold these into an overall $T+2$ overhead.

Therefore, per sample cost is

$$f^{\text{infer}} = T \cdot c(s) + (T+2) = T\left[2\ln(s-1) + 2\gamma - 2 + \frac{2}{s}\right] + (T+2).$$

For n_{inf} instances,

$$\begin{aligned} F^{\text{infer}} &= n_{\text{inf}}(T \cdot c(s) + (T+2)) \\ &= n_{\text{inf}}\left(T\left[2\ln(s-1) + 2\gamma - 2 + \frac{2}{s}\right] + (T+2)\right). \end{aligned}$$

C.9 HS-TREE

Model-specific notation T: number of trees, h: maximum depth of tree, ψ: reference window size.

Training FLOPs Let $|Node| = \sum_{l=0}^{h} 2^l = 2^{h+1} - 1$ be the number of nodes in a full binary tree of height h. Each tree is built by updating simple per-node statistics and routing the ψ reference samples level by level.

(i) *Per-node statistic updates*: For every node we find work range, yielding a constant cost of about 5 FLOPs per node.

Therefore, with a single tree,

$$F_{\text{stat/tree}} = 5(2^{h+1} - 1).$$

(ii) *Routing the ψ reference samples*: At level l there are 2^l nodes and, on average, each processes $\psi/2^l$ samples. With one comparison per routed sample, the cost per level is ψ.

Across levels,

$$F_{\text{routing/tree}} = \sum_{l=0}^{h} \psi \;=\; \psi(h+1).$$

Total FLOPs calculated is

$$\begin{aligned} F^{\text{train}} &= T(F_{\text{stat/tree}} + F_{\text{routing/tree}}) \\ &= T(5(2^{h+1} - 1) + \psi(h+1)). \end{aligned}$$

Inference FLOPs For each sample we route to level h and calculate score, and update leaf mass statistics.

(i) *Path routing and scoring*: Per sample, one comparison per level is conducted with h levels and scoring is conducted with the cumulative sum of $\mathrm{Node.r} \times 2^{\mathrm{Node.k}}$ which costs 3 FLOPs.

Thus,

$$f_{\mathrm{routing/tree}} = h + 3.$$

(ii) *On-path mass updates*: Along the visited path, we update mass with 4 operations including two comparison for lower and upper bound, addition of count, and depth comparison.

As each point pass $h + 1$ nodes,

$$f_{\mathrm{update/tree}} = 4(h + 1).$$

Therefore,

$$\begin{aligned} f^{\mathrm{infer}} &= T(f_{\mathrm{routing/tree}} + f_{\mathrm{update/tree}}) \\ &= T(5h + 7), \\ F^{\mathrm{infer}} &= n_{\mathrm{inf}} T(5h + 7). \end{aligned}$$

D Detailed Experiment Settings

Hyperparameter Tuning For both traditional and deep learning models, we defined hyperparameter search spaces based on ranges commonly adopted in prior benchmark studies. For each model, the parameters of each algorithm (e.g. number of estimators, number of bins, neighborhood size, clustering parameters, latent dimensions, dropout rates, training epochs) were specified as candidate sets. These search spaces were predetermined and systematically explored through grid search across all datasets. For each dataset and model pair, all hyperparameter combinations were evaluated, and the configuration yielding the highest AUROC was selected as the final setting. This procedure ensured that every model was tuned in a consistent and performance-oriented manner while remaining faithful to the parameter ranges established in the literature.

E Additional Experiment Results

E.1 Additional Results of Section 4.1

In our experiments, we provide the complete results for each dataset. The following tables report AUROC and FLOPs across all evaluated models. We observe that GFLOPs for deep learning models are generally much larger than those of traditional models, despite yielding comparable accuracy. For clarity, the best AUROC in each table is highlighted in **bold**, while the second-best score is underlined.

Table 6: All results on the PSM dataset with AUROC and GFLOPs.

Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	0.17	0.11	–	**0.77 ± 0.00**
ABOD	Statistical	926.33	421.87	–	0.75 ± 0.00
LOF	One-class	917.76	416.07	–	0.73 ± 0.00
CBLOF	One-class	6.56	9.91	–	0.70 ± 0.02
PCA	Reconstruction	0.17	0.16	–	0.65 ± 0.00
HBOS	Statistical	0.01	0.01	–	0.73 ± 0.00
LODA	Statistical	0.13	0.10	–	0.65 ± 0.03
Isolation Forest	One-class	< 0.01	0.07	–	0.70 ± 0.02
HS-Tree	One-class	< 0.01	0.10	–	0.54 ± 0.02
DAGMM	Statistical	53.04	11.97	5304.37	0.50 ± 0.03
DeepSVDD	One-class	3.13	0.71	313.22	0.69 ± 0.01
LSTM-AE	Reconstruction	2.45	0.55	244.67	<u>0.76 ± 0.01</u>
LSTM-VAE	Reconstruction	54.73	12.35	13681.67	0.61 ± 0.06
USAD	Reconstruction	0.26	0.06	65.79	0.52 ± 0.01
OmniAnomaly	Reconstruction	2.03	0.46	40.6	0.61 ± 0.00
LUAD	Reconstruction	991.13	223.60	29733.97	0.67 ± 0.01

Table 7: All results on the MSL dataset with AUROC and GFLOPs.

Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	0.36	0.45	–	0.53 ± 0.00
ABOD	Statistical	334.71	536.74	–	0.63 ± 0.00
LOF	One-class	334.41	536.36	–	0.56 ± 0.00
CBLOF	One-class	0.89	2.60	–	$\mathbf{0.65 \pm 0.01}$
PCA	Reconstruction	0.36	0.64	–	0.53 ± 0.00
HBOS	Statistical	0.01	0.01	–	0.62 ± 0.00
LODA	Statistical	0.15	0.21	–	0.47 ± 0.02
IForest	One-class	< 0.01	0.14	–	0.62 ± 0.01
HS-Tree	One-class	< 0.01	0.08	–	$\underline{0.64 \pm 0.03}$
DAGMM	Statistical	10.22	4.31	510.87	0.50 ± 0.02
DeepSVDD	One-class	25.34	10.68	2533.82	0.48 ± 0.03
LSTM-AE	Reconstruction	4.55	1.92	91.03	0.56 ± 0.00
LSTM-VAE	Reconstruction	2.12	0.89	530.67	0.53 ± 0.00
USAD	Reconstruction	0.21	0.09	20.67	0.44 ± 0.00
OmniAnomaly	Reconstruction	0.62	0.26	12.49	0.53 ± 0.00
LUAD	Reconstruction	236.24	99.56	2362.43	0.50 ± 0.00

Table 8: All results on the SMAP dataset with AUROC and GFLOPs.

Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	0.18	0.56	–	0.42 ± 0.00
ABOD	Statistical	998.15	10281.98	–	$\underline{0.64 \pm 0.00}$
LOF	One-class	996.78	10277.64	–	0.62 ± 0.00
CBLOF	One-class	2.48	10.76	–	0.62 ± 0.01
PCA	Reconstruction	0.18	0.36	–	0.41 ± 0.00
HBOS	Statistical	0.01	0.03	–	0.61 ± 0.00
LODA	Statistical	0.08	0.29	–	0.45 ± 0.09
Isolation Forest	One-class	< 0.01	0.48	–	$\mathbf{0.64 \pm 0.01}$
HS-Tree	One-class	0.01	0.70	–	0.45 ± 0.01
DAGMM	Statistical	55.25	58.27	5525.14	0.51 ± 0.01
DeepSVDD	One-class	13.64	14.39	1364.39	0.60 ± 0.00
LSTM-AE	Reconstruction	2.55	2.69	127.40	0.49 ± 0.03
LSTM-VAE	Reconstruction	3.99	4.20	996.66	0.44 ± 0.02
USAD	Reconstruction	1.14	1.20	283.82	0.41 ± 0.01
OmniAnomaly	Reconstruction	3.44	3.63	68.85	0.47 ± 0.00
LUAD	Reconstruction	516.11	544.28	5161.11	0.50 ± 0.01

Table 9: All results on the SMD dataset with AUROC and GFLOPs.

Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	2.10	2.10	–	$\underline{0.73 \pm 0.00}$
ABOD	Statistical	38426.94	38428.58	–	0.71 ± 0.00
LOF	One-class	38357.60	38359.24	–	0.65 ± 0.00
CBLOF	One-class	36.30	34.65	–	0.72 ± 0.01
PCA	Reconstruction	2.10	2.09	–	0.65 ± 0.00
HBOS	Statistical	0.05	0.08	–	0.63 ± 0.00
LODA	Statistical	0.55	0.63	–	0.62 ± 0.02
Isolation Forest	One-class	< 0.01	0.90	–	0.68 ± 0.01
HS-Tree	One-class	< 0.01	0.81	–	0.64 ± 0.02
DAGMM	Statistical	87.14	29.05	8714.30	0.53 ± 0.05
DeepSVDD	One-class	26.72	8.91	6679.65	0.61 ± 0.02
LSTM-AE	Reconstruction	16.54	5.51	1653.89	$\mathbf{0.77 \pm 0.01}$
LSTM-VAE	Reconstruction	158.21	52.74	39551.96	0.63 ± 0.01
USAD	Reconstruction	7.51	2.50	1878.44	0.39 ± 0.00
OmniAnomaly	Reconstruction	39.62	13.21	792.44	0.69 ± 0.02
LUAD	Reconstruction	5553.27	1851.13	166598.10	0.67 ± 0.00

Table 10: All results on the SWaT dataset with AUROC and GFLOPs.

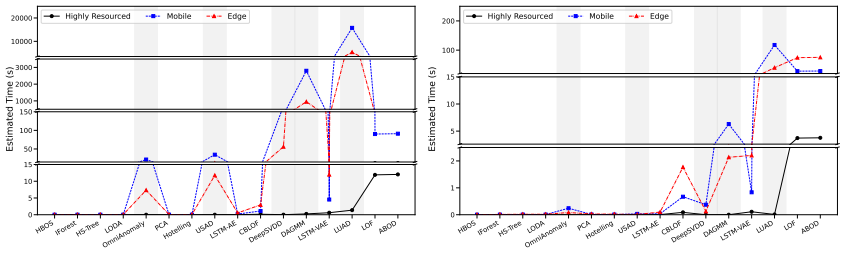
Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	2.64	2.39	–	0.65 ± 0.00
ABOD	Statistical	23615.66	19345.49	–	0.60 ± 0.00
LOF	One-class	23551.21	19287.13	–	0.71 ± 0.00
CBLOF	One-class	3.95	4.01	–	0.81 ± 0.01
PCA	Reconstruction	2.64	2.42	–	0.82 ± 0.00
HBOS	Statistical	0.05	0.07	–	$\mathbf{0.85 \pm 0.00}$
LODA	Statistical	1.39	1.39	–	0.82 ± 0.02
Isolation Forest	One-class	< 0.01	0.36	–	$\underline{0.83 \pm 0.00}$
HS-Tree	One-class	0.01	0.74	–	0.37 ± 0.08
DAGMM	Statistical	40.48	12.22	4047.64	0.51 ± 0.00
DeepSVDD	One-class	51.09	15.42	12773.36	0.82 ± 0.03
LSTM-AE	Reconstruction	37.43	11.30	1871.30	0.76 ± 0.01
LSTM-VAE	Reconstruction	116.89	35.29	29223.50	0.76 ± 0.11
USAD	Reconstruction	6.56	1.98	655.69	0.81 ± 0.00
OmniAnomaly	Reconstruction	5.13	1.55	102.60	0.83 ± 0.00
LUAD	Reconstruction	565.27	170.64	16958.00	0.75 ± 0.01

Table 11: All results on the WADI dataset with AUROC and GFLOPs.

Model	Type	GFLOPs ↓			AUROC ↑
		Train	Inference	Full-training	
Hotelling	Statistical	23.93	5.27	–	0.53 ± 0.00
ABOD	Statistical	125697.00	6047.49	–	0.49 ± 0.00
LOF	One-class	125611.70	6028.72	–	0.54 ± 0.00
CBLOF	One-class	97.02	1.66	–	0.53 ± 0.01
PCA	Reconstruction	23.93	3.18	–	0.50 ± 0.00
HBOS	Statistical	0.19	0.06	–	$\mathbf{0.74 \pm 0.00}$
LODA	Statistical	1.46	0.34	–	0.72 ± 0.04
Isolation Forest	One-class	< 0.01	0.33	–	$\underline{0.74 \pm 0.02}$
HS-Tree	One-class	< 0.01	0.20	–	0.63 ± 0.05
DAGMM	Statistical	750.82	55.12	75081.59	0.45 ± 0.05
DeepSVDD	One-class	97.48	7.16	24369.08	0.43 ± 0.03
LSTM-AE	Reconstruction	49.08	3.60	4907.52	0.54 ± 0.00
LSTM-VAE	Reconstruction	206.30	15.15	51575.30	0.54 ± 0.01
USAD	Reconstruction	29.25	2.15	7312.16	0.45 ± 0.01
OmniAnomaly	Reconstruction	71.70	5.26	1433.97	0.58 ± 0.00
LUAD	Reconstruction	7269.83	533.72	218094.80	0.54 ± 0.00

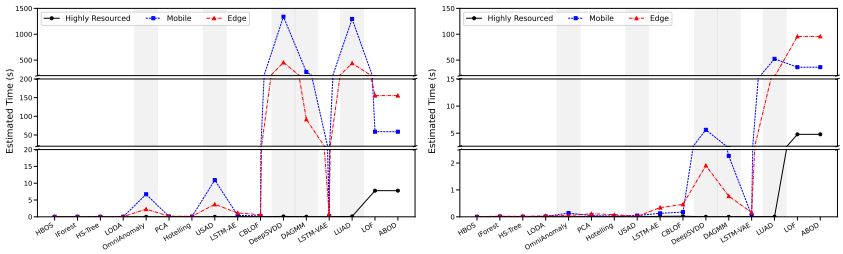
E.2 Additional Results of Section 4.2

For each dataset, we report the estimated execution time under Highly Resourced, Mobile, and Edge scenarios. The following figures show dataset specific visualizations that highlight differences in model feasibility across environments. These results confirm the overall trend that deep learning models demand substantial resources, while traditional models remain efficient, although the degree of variation differs across datasets.



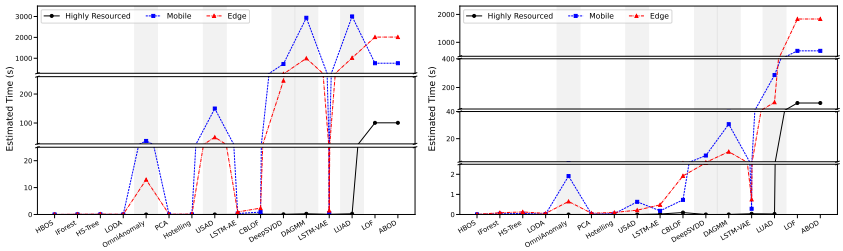
(a) Train (b) Inference

Figure 4: Estimated execution time on PSM.



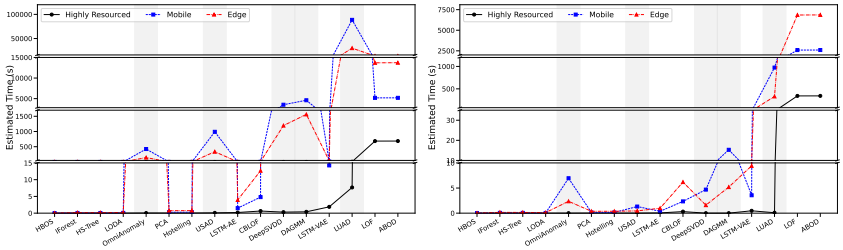
(a) Train (b) Inference

Figure 5: Estimated execution time on MSL.



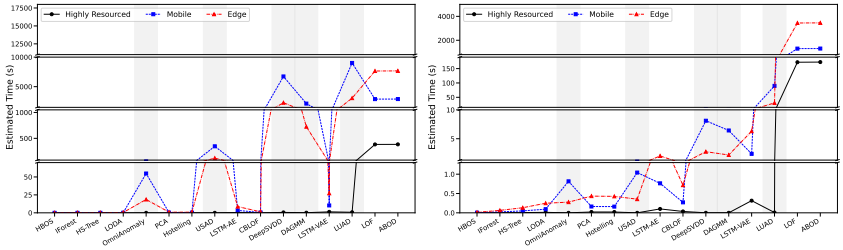
(a) Train (b) Inference

Figure 6: Estimated execution time on SMAP.



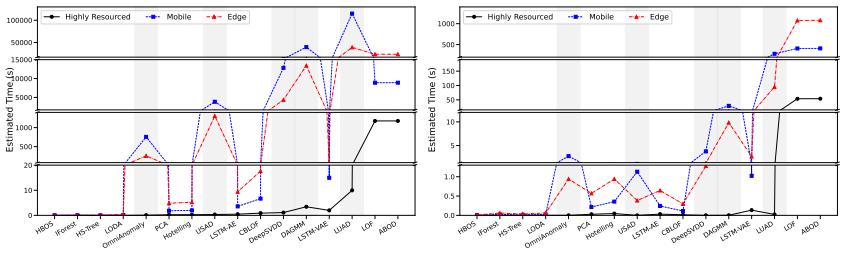
(a) Train (b) Inference

Figure 7: Estimated execution time on SMD.



(a) Train (b) Inference

Figure 8: Estimated execution time on SWaT.



(a) Train (b) Inference

Figure 9: Estimated execution time on WADI.

E.3 Additional Results of Section 4.3

We report scalability tests conducted across datasets of varying dimensionality, where FLOPs were calculated by progressively slicing the size of the dataset. For each dataset, the maximum test range was defined by the smaller length of either the training or inference set, ensuring comparability between the two phases. All models evaluated in this study are included. The left panel of each figure depicts training FLOPs, while the right panel presents inference FLOPs, with both y-axes plotted on a logarithmic scale. Overall, the results reveal consistent scaling patterns across datasets. k-NN based methods exhibit steep growth in computational cost as data size increases, while tree-based and projection methods remain relatively efficient.

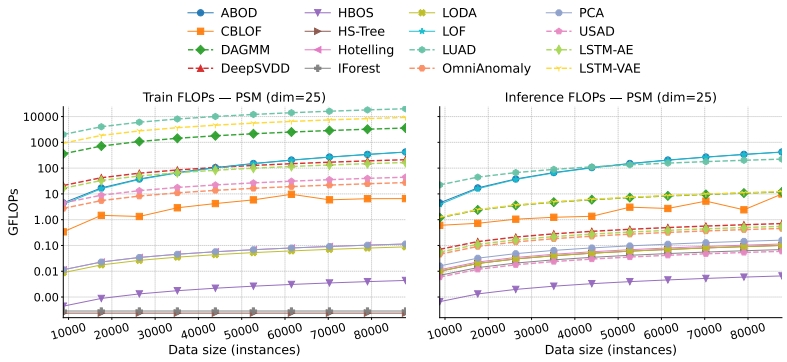


Figure 10: Scalability results on PSM.

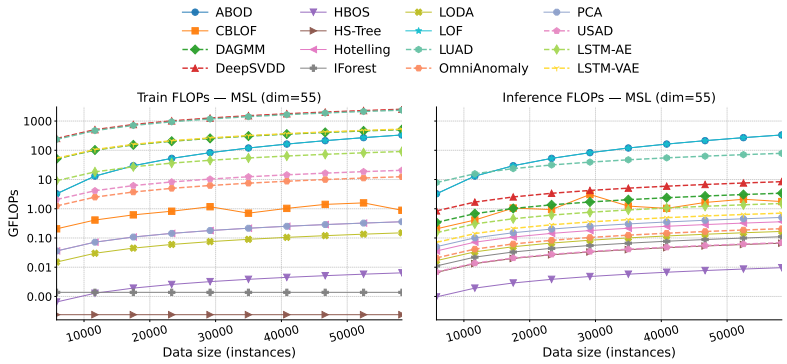


Figure 11: Scalability results on MSL.

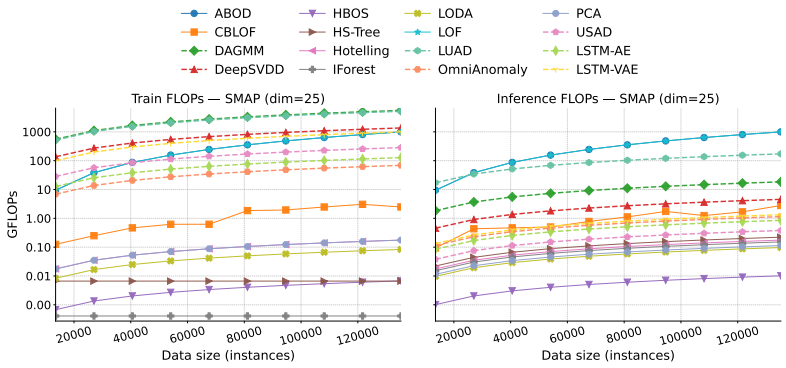


Figure 12: Scalability results on SMAP.

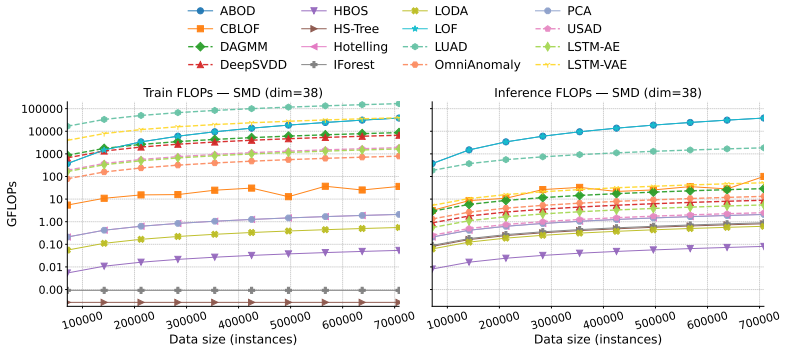


Figure 13: Scalability results on SMD.

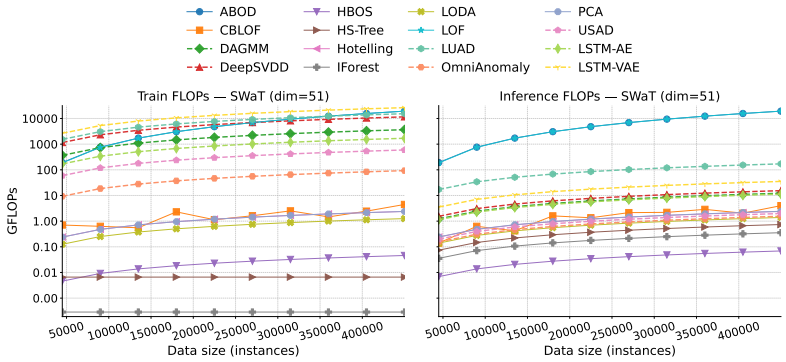


Figure 14: Scalability results on SWaT.

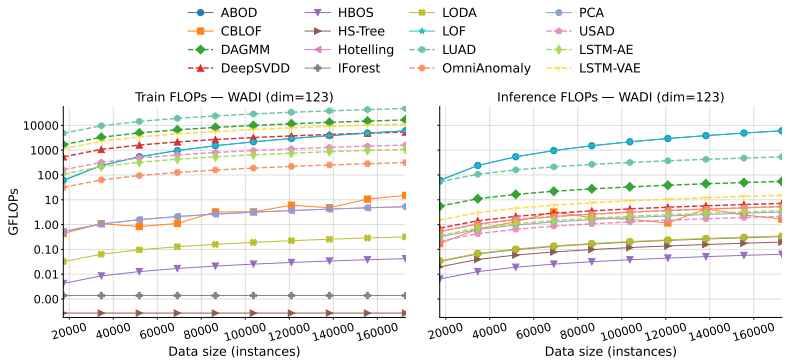


Figure 15: Scalability results on WADI.

E.4 LLM USAGE STATEMENT

The large language model (LLM) was used solely to improve the clarity and readability of the manuscript. Specifically, they helped polish the writing, refine grammar, and improve phrasing. The use of LLM was limited to language editing, and LLM did not contribute to the ideation of the research, experimental design, implementation, analysis, or interpretation of the results.