Evaluating Drift Detection Methods for Failure Prediction in 5G Network Resource Management

Lucas Matni Bezerra[1], **Adriano Madureira dos Santos**[1], **Glauco Estácio Gonçalves**[1], **Marcos César da Rocha Seruffo**[1]

[1]Postgraduate Program in Electrical Engineering,
Institute of Technology
Federal University of Pará (UFPA) – Belém – PA – Brazil

{lucas.matni, adriano.madureira.santos}@itec.ufpa.br

{glaucogoncalves,seruffo}@ufpa.br

Abstract. *In dynamic cloud environments, ensuring accurate and adaptive resource allocation is essential to maintain 5G network performance. This study compares concept drift detection strategies to improve fault tolerance and resource efficiency using the 5G3E dataset, which encompasses abrupt, gradual, and recurring changes in CPU and network demand. An LSTM model was deployed to predict resource usage, and three drift detection techniques—Page-Hinkley (PHT), ADWIN, and KSWIN—were applied to the prediction residuals. Experimental results show that PHT captured 50% of the CPU drifts and 66% and KSWIN, conversely, detected every drift event (100% recall) at the expense of numerous false positives. These findings highlight the trade-off between sensitivity and precision in dynamic 5G scenarios, underscoring the need to select a drift detection method based on acceptable levels of missed detections versus false alarms.*

1. Introduction

The increasing demand for high-performance, low-latency networks, essential for applications such as IoT, autonomous vehicles, and real-time communications, has driven the adoption of 5G and is shaping the evolution of intelligent, adaptive architectures envisioned for 6G [Saxena and Singh 2022]. Although 5G has introduced new capabilities through virtualization and cloudification, the transition to 6G will emphasize automation, resilience, and real-time decision making, particularly in open and disaggregated infrastructures such as OpenRAN.Our results contribute to this vision by assessing the operational viability of such methods in realistic 5G scenarios that anticipate future 6G demands.

Cloud-based infrastructures, despite offering scalability, bring significant challenges to resource management and Quality of Service (QoS) [Hu and Zhang 2020]. Faults—whether due to sudden overloads or gradual changes—can degrade performance and increase operational costs [Phung et al. 2022]. Compounding this, concept drift, defined as changes in the data distribution over time [Gama et al. 2004], threatens the accuracy of predictive models in such dynamic environments.

Previous work has explored resource failure prediction in virtualized networks. For example, [Saxena and Singh 2022] proposed an elastic resource management model,

and [Mehmood et al. 2023] introduced a drift-aware LSTM architecture for cloud workloads. Other studies, such as [Gudepu et al. 2024], focused on the mitigation of anomalies in open RAN contexts. More recently, advanced drift detection methods such as STUDD [Cerqueira et al. 2023], TRANSCENDENT [Barbero et al. 2022], and OTL-FT-KCD [Zhang and Zhang 2024] have gained attention for their adaptive capabilities.

In contrast, this study investigates the behavior of simpler and well-established drift detectors, often overlooked despite their practical advantages. Although complex models offer flexibility, they can be costly to deploy or underperform in specific edge scenarios. We examine how classic techniques such as Page-Hinkley, ADWIN, and KSWIN perform in detecting model obsolescence and maintaining LSTM-based fault prediction. The objective is to assess the effectiveness of these detectors in identifying performance shifts before failures occur.

This paper is organized as follows: Section 2 details the methodology, Section 3 presents the results, and Section 4 concludes with final remarks and future directions.

2. Evaluation Method

Following the definition of [Saxena and Singh 2022], faults are defined as resource consumption exceeding a set operational limit. The fault alert system uses predicted resource utilization to anticipate failures, meaning that any performance degradation in this component can compromise overall fault detection. Our experiments evaluate whether concept drift detectors can effectively signal model obsolescence and trigger retraining.

Figure 1 shows the evaluation method divided into three phases: (i) Data collection and preprocessing, where relevant metrics from the 5G3E dataset are selected, structured, and normalized (Section 2.1); (ii) ML training, which describes how LSTM models are trained and validated to predict resource usage as a baseline (Section 2.2); and (iii) Concept drift evaluation, where PHT, ADWIN, and KSWIN are applied to detect drift and assess its impact on model reliability (Section 2.3). For reproducibility, all data and code are available on GitHub[1].

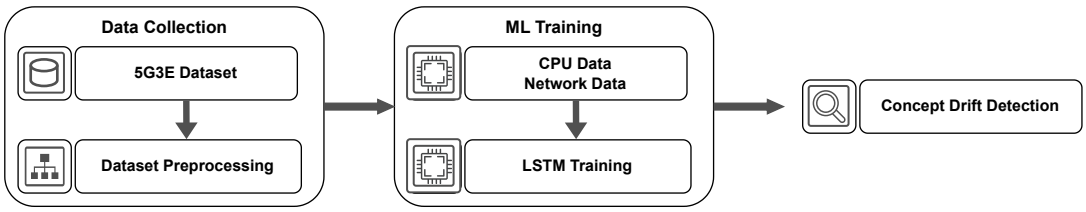

Figure 1. Evaluation method order and phases.

2.1. Data Collection and Preprocessing

For this study, we used the 5G3E dataset [Phung et al. 2022], which combines simulation with real mobile network traffic emulation. Based on anonymized operator data, the dataset integrates NS-3 with virtualized 5G components—using Open5GS for the core network and srsRAN for radio interactions. Over 14 days, 1.1 TB of time-series data was

[1] https://github.com/lucasmatnibezerra/FailedDetectionCloud

collected, with sampling rates of 150 ms for physical/container levels and 80 ms for radio access.

We focused on server-level metrics from Server 1, which hosts virtual Base Stations processing TCP traffic from User Equipment (UEs). This server is representative of cloudified 5G networks that use software-based base stations on commercial off-the-shelf hardware [Gonçalves et al. 2020, Kassi and Hamouda 2024]. The monitored metrics included CPU utilization (*node_load1*) and transmitted network traffic (*node_network_transmit_bytes_total*) via the *eno1* interface.

Due to the 150 ms sampling rate, approximately 576,000 samples were collected daily. To reduce computational complexity while preserving representativeness, the data was resampled to one-minute intervals, resulting in a final dataset of 20,020 samples over 14 days.

For training and testing the machine learning models, the data was structured into 10-minute sequences to capture temporal dependencies, as determined by autocorrelation analysis. The target variable was defined to predict one minute ahead, matching the system's operational interval. Additionally, network traffic data was converted to incremental values by calculating the difference between consecutive samples.

Normalization was performed using the maximum value within three standard deviations of the mean, which preserved data variation while mitigating outliers.

2.2. ML Training

Two LSTM models were developed to predict CPU and Network usage in 5G networks using structured data from the preprocessed dataset. This model was chosen for its ability to capture sequential dependencies in dynamic environments such as cloud computing and 5G [Ahmad et al. 2023].

Both models feature an architecture comprising three bidirectional LSTM layers with 256 neurons each, followed by four fully connected layers. A Normalization Layer precedes the fully connected layers, and a ReLU activation function combined with a 30% Dropout is used to prevent overfitting. The final output layer employs a Sigmoid activation function for stepwise regression.

The first 7 days of data (totaling 10,010 samples) were used for training, with 90% allocated for training and 10% for validation. Training was conducted using the Adam optimizer with a learning rate of 10^{-3} and Mean Squared Error (MSE) as the loss function. Mini-batches of 1,024 samples were processed over 1000 epochs, with early stopping triggered after 10 consecutive epochs without improvement in validation loss, ensuring that the best model was retained.

The remaining 7 days of data were used to test the LSTM models and evaluate the concept drift detectors. All experiments were implemented in Python 3.12 using PyTorch, NumPy, and Pandas on a system equipped with an Intel Core i3-10100F CPU, an NVIDIA RTX 2060 GPU (12 GB VRAM), 16 GB DDR4 RAM, and a 1 TB HDD.

2.3. Concept Drift Detector Evaluation

Three concept drift detectors were evaluated: PHT, ADWIN, and KSWIN. These methods are widely used in data stream environments for their strong theoretical basis and

robustness [Zhang and Zhang 2024, Cerqueira et al. 2023]. PHT targets abrupt changes, while ADWIN and KSWIN dynamically adjust window sizes to detect both gradual and abrupt drifts.

All three detectors operate on the prediction residuals from the LSTM models (the difference between actual and predicted values). Drift points were defined at daily transitions (1,430 samples) to capture natural fluctuations in CPU and network usage.

Detector	**Parameter**	**Values Tested**
ADWIN	δ	0.0001, 0.001, 0.005, 0.05, 0.15, 0.2, 0.5, 0.7
KSWIN	α	0.0001, 0.001, 0.005, 0.01, 0.05, 0.07, 0.2, 0.5, 0.7
	window_size	75, 80, 100, 110, 120, 150, 160
	stat_size	30, 32, 35, 36, 40, 42
Page-Hinkley	δ	0.001, 0.002, 0.005, 0.007, 0.01, 0.1, 0.15
	threshold	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
	α	0.85, 0.95, 0.999, 0.9995, 0.99999

Table 1. Grid search values for tuning drift detection parameters

Our evaluation follows [Gudepu et al. 2024] and [Barros and Santos 2018], defining a 90-sample (1.5-hour) tolerance window before and after each true drift. Within this window, we count True Positives (TP), False Positives (FP), and False Negatives (FN). Precision is TP / (TP + FP), Recall is TP / (TP + FN), and the F1-Score is the harmonic mean of Precision and Recall. Because drift detection is time-dependent, we also measure detection delays: "Delay Before" (early detections) and "Delay After" (late detections).

Parameters for each detector were optimized via grid search, following [Cabello-López et al. 2022, Barbero et al. 2022]. Table 1 shows the tested parameter values, and Table 2 presents the final configurations for CPU and Network usage data.

Detector	**Parameter**	**CPU**	**Network**
ADWIN	δ	0.2	0.9
Page-Hinkley	δ	0.005	0.007
	threshold	0.8	0.1
	α	0.999	0.85
KSWIN	α	0.05	0.0005
	window_size	160	150
	stat_size	42	40

Table 2. Optimized parameters for each drift detector

3. Results and Discussion

Figures 2 and 3 show the predicted versus actual CPU and Network Usage in the test samples. The 5G3E dataset, obtained from a testbed emulating various scenarios, indicates that CPU usage ranges from 50% to 90% and Network Usage from 70% to 100% over a day (1440 samples). Daily patterns and drifts at day transitions are clearly visible, with Network Usage exhibiting more abrupt shifts.

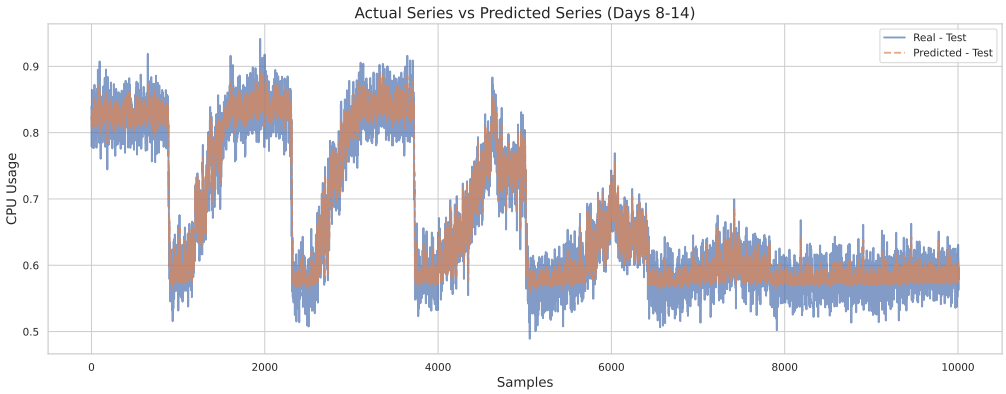

Figure 2. Predicted vs. Actual CPU Usage

The LSTM model captures overall trends but shows deviations during sudden changes. The RMSE is approximately 2.5% for CPU and 0.9% for Network, confirming that the dataset reflects complex 5G usage patterns and the need for drift detection.

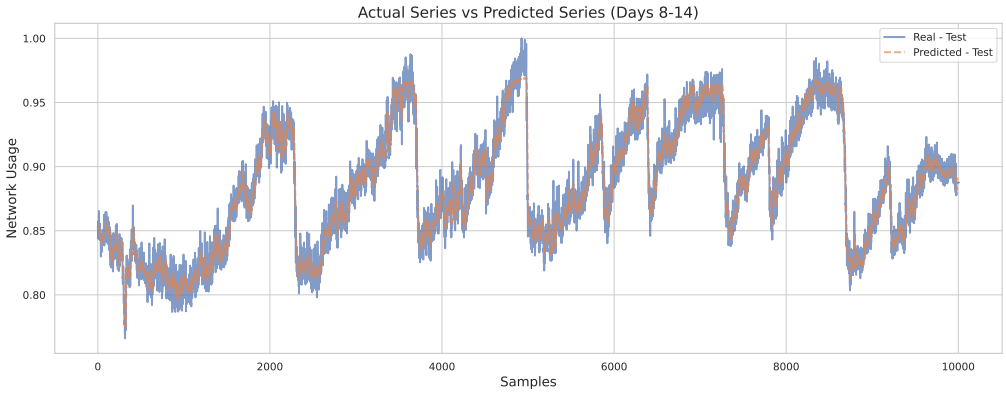

Figure 3. Predicted vs. Actual Network Usage

Figures 4 and 5 illustrate the residual error behavior and drift detection over time. These graphs show that PHT and KSWIN detect more true drifts than ADWIN, though at the cost of more false alarms; ADWIN triggers only once for CPU and four times for Network.

Table 3 summarizes the performance metrics of each detector. For CPU, PHT detected 3 of 6 drifts (TP=3, FN=0) with 6 false positives, yielding a Precision of 0.333 and a Recall of 0.5. ADWIN failed to detect any CPU drifts (TP=0, FN=6), and KSWIN detected all CPU drifts (TP=6) but with 33 false positives, resulting in a Precision of 0.153 and Recall of 1.0. For Network, PHT achieved a Precision of 0.235 and Recall of 0.666; ADWIN again had zero recall; and KSWIN attained a Precision of 0.136 with a Recall of 1.0.

ADWIN showed inconsistent performance, failing to detect any true CPU drifts and yielding high precision (0.500) but low recall (0.333) for Network. KSWIN detected drifts with many false positives (18 for CPU, 15 for Network), resulting in low precision (0.100 for CPU, 0.118 for Network) despite higher recall.

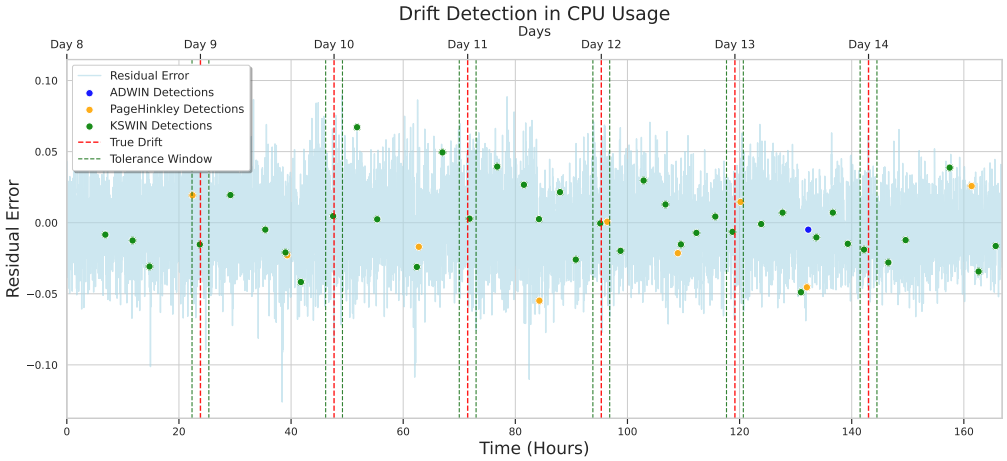

Figure 4. Drift Detection in CPU Usage

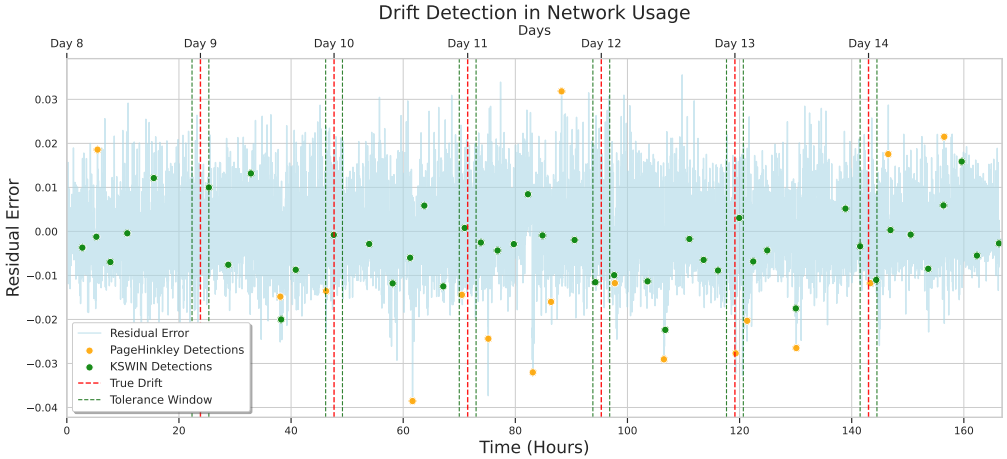

Figure 5. Drift Detection in Network Usage

Method (Resource)	TP	FP	FN	Precision	Recall	F1-Score
Page-Hinkley (CPU)	3	6	0	0.333	0.500	0.400
ADWIN (CPU)	0	1	6	0.000	0.000	0.000
KSWIN (CPU)	6	33	0	0.153	1.000	0.266
Page-Hinkley (Network)	4	13	2	0.235	0.666	0.347
ADWIN (Network)	0	0	6	0.000	0.000	0.000
KSWIN (Network)	6	38	0	0.136	1.000	0.240

Table 3. Detection performance of drift detectors for CPU and network usage

Table 4 presents the detection delays. For CPU, ADWIN did not detect any drifts; KSWIN had a ”Delay Before” of 31.5 minutes with nearly no delay after; and PHT showed delays of 47 minutes before and 47.67 minutes after the drift. For Network, PHT anticipated drifts by about 76 minutes but reacted late, while KSWIN consistently detected changes approximately 47.5 minutes in advance.

Method (Resource)	**Delay Before (min)**	**Delay After (min)**
Page-Hinkley (CPU)	85.00	61.50
ADWIN (CPU)	–	–
KSWIN (CPU)	20.40	20.00
Page-Hinkley (Network)	73.50	14.00
ADWIN (Network)	–	–
KSWIN (Network)	47.00	68.00

Table 4. Detection delays (before and after) for each drift detector

These results highlight the trade-offs between early detection—beneficial for proactive intervention—and the risks of excessive false alarms or delayed responses.

4. Conclusion

This work evaluated the effectiveness of three concept drift detectors—Page-Hinkley, ADWIN, and KSWIN—in supporting fault prediction through LSTM models in dynamic 5G environments. Experiments were conducted using the 5G3E dataset, which includes realistic CPU and network usage patterns with abrupt, gradual, and recurring drifts. The results indicate that each method presents trade-offs. Page-Hinkley achieved moderate recall but generated a high number of false positives. ADWIN exhibited low sensitivity, failing to detect most drift events. KSWIN achieved perfect recall but at the expense of numerous false alarms, especially in noisy conditions.

These findings highlight the importance of selecting drift detectors based on the desired trade-off between early detection and false positives. Even simple detectors, when well-tuned, remain viable for real-time applications and can complement more complex methods. Future work will explore ensemble-based detection strategies and assess their cost-effectiveness in real-time pipelines. We also intend to extend the evaluation to larger and more heterogeneous network environments.

Acknowledgements

This study was funded in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001; and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

References

Ahmad, S., Shakeel, I., Mehfuz, S., and Ahmad, J. (2023). Deep learning models for cloud, edge, fog, and IoT computing paradigms: Survey, recent advances, and future directions. *Computer Science Review*, 49:100568.

Barbero, F., Pendlebury, F., Pierazzi, F., and Cavallaro, L. (2022). Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE.

Barros, R. S. M. and Santos, S. G. T. C. (2018). A large-scale comparison of concept drift detectors. *Information Sciences*, 451-452:348–370.

Cabello-López, T., Cañizares-Juan, M., Carranza-García, M., Garcia-Gutiérrez, J., and Riquelme, J. C. (2022). Concept drift detection to improve time series forecasting of wind energy generation. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 133–140. Springer.

Cerqueira, V., Gomes, H. M., Bifet, A., and Torgo, L. (2023). Studd: A student–teacher method for unsupervised concept drift detection. *Machine Learning*, 112(11):4351–4378.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In *Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA)*, pages 286–295, São Luís, Brazil. Springer.

Gonçalves, G. E., Santos, G. L., Ferreira, L., Rocha, É. d. S., de Souza, L. M., Moreira, A. L., Kelner, J., and Sadok, D. (2020). Flying to the clouds: the evolution of the 5G radio access networks. *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, pages 41–60.

Gudepu, V., Abbas, A., Doe, J., and Smith, J. (2024). The drift handling framework for open radio access networks: An experimental evaluation. *Journal of Network and Systems Management*, 32:45–63.

Hu, P. and Zhang, J. (2020). 5G-enabled fault detection and diagnostics: How do we achieve efficiency? *IEEE Internet of Things Journal*, 7(4):3267–3281.

Kassi, M. and Hamouda, S. (2024). Ran virtualization: How hard is it to fully achieve? *IEEE Access*.

Mehmood, T., Latif, S., Jamail, N. S. M., Malik, A., and Latif, R. (2023). LSTMDD: An optimized LSTM-based drift detector for concept drift in dynamic cloud computing. *PeerJ Computer Science*. Manuscrito submetido, sob revisão.

Phung, C.-D., Ruba, S. B., and Secci, S. (2022). An open dataset for beyond-5G data-driven network automation experiments. In *2022 1st International Conference on 6G Networking (6GNet)*, pages 1–4. IEEE.

Saxena, D. and Singh, A. K. (2022). OFP-TM: An online vm failure prediction and tolerance model towards high availability of cloud computing environments. *The Journal of Supercomputing*, 78:8003–8024.

Zhang, Z. and Zhang, H. (2024). An online transfer learning model for intrusion detection using ft-transformer and kswin-driven concept drift detection mechanism. In *2024 5th International Conference on Computer Engineering and Application (ICCEA)*, pages 128–131. IEEE.