Improving Transit System Reliability through Log-Based Anomaly Detection in a Smart Autonomous Device

Nadira Anjum Nipa, Nizar Bouguila and Zachary Patterson
Concordia Institute for Information and Systems Engineering, Concordia University, Montreal, QC, Canada
nadiraanjum.nipa@mail.concordia.ca, nizar.bouguila@concordia.ca, zachary.patterson@concordia.ca

***Abstract*—The evolution of intelligent transportation systems (ITS) and smart city infrastructure has led to the integration of smart display devices within public transit networks. These Internet of Things (IoT)-enabled systems deliver a range of transit-related information, thereby improving the efficiency, accessibility, and reliability of public transportation services. However, hardware faults, software bugs, or connectivity disruptions can cause performance issues in these devices, which often remain undetected until they impact service. This paper explores log-based anomaly detection as a scalable, non-intrusive approach for monitoring device health and enabling early fault detection. Using real-world logs, we evaluate a broad range of machine learning and deep learning models across four learning paradigms: supervised, semi-supervised, unsupervised, and weakly supervised. Our study consists of two phases: the first applies models to a small, manually labeled dataset; the second scales the evaluation to a larger dataset labeled heuristically using domain-specific keywords. We analyze trade-offs in detection accuracy, robustness to noisy labels, and computational efficiency. The results show that the performance of the model varies on the basis of the operational conditions of real-world transit environments. This work provides practical insights for implementing predictive maintenance in smart transit systems, reducing downtime, and improving reliability without the need for extensive manual log annotation or constant supervision.**

***Index Terms*—Intelligent transportation systems (ITS), Logs, Anomaly detection**

I. Introduction

The advancement of Intelligent Transportation Systems (ITS) has increased the reliance on Internet of Things (IoT) technologies to improve the reliability and efficiency of public transportation services. "SCiNe" (Smart City Network), developed by BusPas [1], is a solar-powered autonomous transit display designed for deployment in areas with limited connectivity and power. To enable smart mobility, it delivers real-time transit information such as bus arrival times and passenger flow, supporting demand-responsive operations.

Like other IoT devices, SCiNe systems may experience hardware failures, software anomalies, and intermittent connectivity, which can degrade performance and disrupt service. Traditional monitoring approaches are not well suited for autonomous deployments where continuous supervision is unavailable, making reliable fault detection challenging.

Log-based anomaly detection offers a scalable and non-intrusive solution for monitoring device health. System logs record time-stamped internal events, including warnings, errors, and operational messages, providing valuable insight into system behavior. When properly analyzed, logs enable early detection of abnormal patterns, improving system reliability and preventing failures. However, real-world industrial logs are often noisy, unstructured, and lack sufficient labeled anomalies, making model selection and deployment difficult.

Conventional anomaly detection methods based on manual inspection or simple statistical rules are insufficient for large-scale, complex log data. As a result, machine learning (ML) and deep learning (DL) techniques have become increasingly important. Unlike most prior studies that rely on clean public datasets, this work focuses on real industrial logs generated by SCiNe devices, which exhibit noise, varying formats, and limited ground-truth labels [2].

This paper explores log-based anomaly detection in the context of SCiNe devices that are currently under development for future use in transit operations. This study investigates the performance of various ML and DL models on log data from SCiNe devices, using two datasets: one labeled with expert supervision and another labeled heuristically through domain-specific rules. Our research focuses on several important questions:

- How do traditional machine learning and deep learning models perform on industrial log data of varying sizes?
- What are the trade-offs between different anomaly detection models in terms of robustness, scalability, and computational efficiency when applied to industrial log data?
- How can predictive analysis of SCiNe logs improve transit system reliability and enable proactive maintenance?

Through this case study, we provide practical insights into deploying scalable log-based anomaly detection for smart transit infrastructure while minimizing manual labeling effort.

II. Literature Review

This section reviews key research relevant to this study, focusing on two primary areas: (1) Anomaly detection in Intelligent Transportation Systems (ITS) and (2) Log-based anomaly detection techniques across different computing environments.

A. Anomaly Detection in ITS

Anomaly detection has been widely studied in transportation systems using sensor, traffic, and time-series data. Previous work has applied time-series forecasting to detect abnormal traffic behavior [3], real-time sensor fault detection using Kalman filtering and deep learning [4], and unsupervised clustering with graph-based methods for incident detection under limited labels [5]. More recent studies highlight that detection performance strongly depends on data quality and application constraints [6].

While these studies provide valuable insight, most focus on sensor signals or high-level traffic metrics. In contrast, log-based anomaly detection in IoT-enabled transit infrastructure remains relatively underexplored. Building on our earlier work [2], [7], this study examines system logs from two SCiNe transit display devices with different operational characteristics, enabling evaluation of anomaly detection robustness across diverse real-world conditions.

B. Log-Based Anomaly Detection

Log-based anomaly detection has advanced rapidly in distributed computing, cloud systems, and industrial environments [8]–[11]. The growing availability of unstructured event logs from software and IoT systems has driven the use of ML and DL approaches to automate anomaly detection across supervised, semi-supervised, unsupervised, and weakly supervised paradigms.

1) Machine Learning Approaches: Machine Learning models such as Decision Trees [12], Support Vector Machines [13], and regression-based classifiers [10] perform effectively with labeled data but require costly manual annotation, limiting scalability in industrial IoT contexts. Unsupervised methods such as Isolation Forests [14], PCA [15], and Invariant Mining [16] eliminate the requirement of labels by modeling event frequency or structure. Although these methods offer flexibility, they can have difficulty in capturing context and temporal dependencies within complex logs, especially in noisy conditions.

2) Deep Learning Approaches: Deep learning models have shown impressive capabilities in modeling high-dimensional log data. Supervised DL models such as LogRobust [17], LogCNN [18], and LogAttention [19] achieve strong accuracy but rely on extensive labeled datasets. Semi-supervised models, such as PLELog [20] and LogBERT [21], reduce this dependency by learning normal behavior through pseudo-labeling or contextual embeddings. Unsupervised techniques like DeepLog [22] and LogAnomaly [23] use recurrent neural networks to model expected log sequences and identify deviations as anomalies. Variants such as LogAttn [24] and LogTAD [25] use attention mechanisms and domain adaptation to enhance performance in cross-system or label-sparse settings. Weakly supervised methods, including SpikeLog [26] and LogLG [27], demonstrate strong performance through the integration of heuristically labeled anomalies and extensive collections of unlabeled data, making them suitable for complex industrial or transit environments.

Despite these advances, most existing studies rely on clean, well-labeled datasets, limiting applicability to noisy industrial systems. Furthermore, while weakly and semi-supervised methods show promise in balancing label efficiency and accuracy, their effectiveness in ITS environments remains insufficiently explored. Extending our previous work [2], [7], this study evaluates ML and DL models across multiple learning paradigms using real SCiNe logs characterized by noise, incomplete labels, and diverse operational conditions, with the goal of identifying scalable and practical anomaly detection strategies for smart transit systems

III. Methodology

A. Data Collection

This paper uses two real-world system log datasets collected from distinct SCiNe devices, previously introduced in [2], [7]. Each dataset reflects different operational characteristics and labeling strategies, enabling evaluation under varying data quality and scale. SCiNe devices generate multiple log types; this work focuses on system logs that capture low-level device operations such as boot events, kernel messages, and hardware interactions, which are closely related to device health and reliability. Table I summarizes both datasets.

Dataset A comprises 30,730 log entries collected over 14 hours from a single device and manually labeled by domain experts. Only 184 entries (0.6%) were identified as anomalous, resulting in strong class imbalance but high label quality, making it suitable for controlled evaluation.

Dataset B is substantially larger, originally collected over five months from another device, from which a representative 24-hour subset of 598,237 logs was used. Anomalies were identified heuristically using keyword-based rules (e.g., “fail,” “error,” “warn”), producing 46,526 anomalous logs (7.78%). Although this approach reflects practical industrial conditions where large-scale manual labeling is infeasible, it may introduce noise and evaluation bias. Models using semantic representations may implicitly learn these keywords, potentially inflating performance on Dataset B. Therefore, results on Dataset B should be interpreted cautiously.

B. Log-based Anomaly Detection Pipeline

Prior to applying system logs for anomaly detection models, raw logs were processed through a standardized pipeline to convert unstructured data into model-ready representations. The typical pre-processing workflow consists of log parsing, log grouping, log representation, and anomaly detection. Figure 1 shows the workflow.

1) Log Parsing: Log parsing converts unstructured log messages into structured templates by separating constant elements from variable ones, assigning a unique event template ID to each message type. Various automatic log parsers are available to carry out this task.

2) Log Grouping: After parsing, the logs need to be grouped into sequences that can serve as separate input samples for modeling. There are three ways to do this.

TABLE I
SUMMARY OF LOG DATASETS

Attribute	Dataset A	Dataset B
Source	SCiNe Device A	SCiNe Device B
Time Span	14 hours	Over 24 hours
Total Log Entries	30,730	598,237
Anomalous Logs	184 (0.6%)	46526 (7.78%)
Labeling Approach	Manual(Expert-driven)	Heuristic (Keyword-based)

i) Fixed windows: Group logs into a predetermined number of entries or a specific time frame, ensuring that there is no overlap between the windows.

ii) Sliding windows: Employ both window size and step size, facilitating overlap between neighboring windows.

iii) Session windows: Group logs based on unique session identifiers (e.g., block ID or node ID). While effective for some public datasets, this approach was not applicable to SCiNe logs due to the absence of consistent session identifiers. Future SCiNe software versions could support session-based analysis by incorporating application-level identifiers or transaction IDs into system logs.

3) Log Representation: Following the grouping process, every log sequence is converted into a feature vector representation suitable for anomaly detection. These sequences can be represented in three primary ways:

i) Quantitative vectors: Count the occurrence of each event ID within a sequence. These are frequently used alongside conventional ML models such as support vector machines or decision trees.

ii) Sequential vectors: Maintain the chronological order of events in the sequence, which is essential for deep learning models to grasp temporal patterns.

iii) Semantic vectors: Capture the contextual meaning of log messages through the use of pre-trained embeddings or language models, thereby enriching comprehension beyond just structural patterns.

4) Anomaly Detection: The resulting feature vectors were subsequently fed into ML and DL anomaly detection models. ML models primarily detect deviations in event frequency patterns, while DL models learn normal sequential behavior and identify deviations from learned patterns.

IV. EXISTING METHODS

This study evaluates a range of log-based anomaly detection techniques. Machine learning techniques were examined within the supervised and unsupervised categories, whereas deep learning methods were applied across a wider range, including supervised, unsupervised, semi-supervised, and weakly supervised approaches.

A. Supervised ML Methods

Logistic Regression (LR) [28]: A probabilistic binary classifier applied to event-count feature vectors to distinguish normal and anomalous log sequences. It is computationally efficient and suitable for linearly separable data.

Support Vector Machine (SVM) [13]: Constructs an optimal decision boundary in a high-dimensional feature space

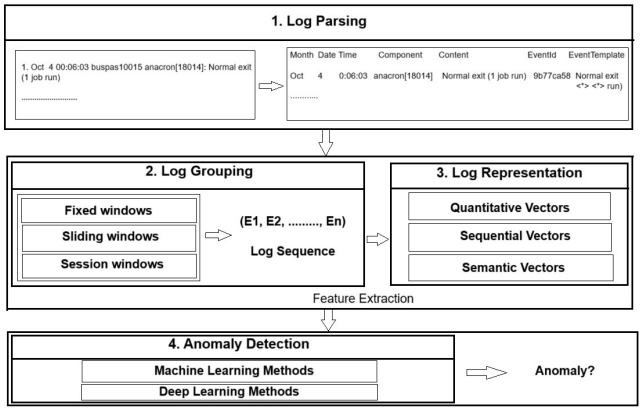

Fig. 1. Log-based anomaly detection workflow [2].

using labeled event-count vectors, providing strong performance on structured log data.

Decision Tree (DT) [12]: A rule-based classifier that recursively partitions feature space using event frequency thresholds, enabling interpretable anomaly detection.

B. Unsupervised ML Methods

PCA [15]: Detects anomalies by identifying deviations from the principal subspace of normal log patterns after dimensionality reduction.

Isolation Forest (IF) [14]: Identifies anomalies based on how easily data points can be isolated through random partitioning, making it efficient for high-dimensional data.

Log Clustering (LC) [29]: Groups log sequences into clusters based on event-count similarity and flags sequences that deviate significantly from known cluster patterns.

C. Deep Learning Methods

SpikeLog [26]: This weakly supervised model combines limited labeled data with large-scale unlabeled logs using a spiking neural network, offering robustness to noisy or incomplete labels.

PLELog [20]: This semi-supervised model learns normal behavior through probabilistic labeling and attention-based sequence modeling, reducing dependence on fully labeled data.

DeepLog [22]: This unsupervised model uses an LSTM-based sequence prediction approach to learn normal event patterns and identify deviations as anomalies.

LogAnomaly [23]: This unsupervised model integrates sequential and quantitative features within an LSTM framework to capture both temporal and frequency-based irregularities.

LogRobust [17]: A supervised Bi-LSTM model enhanced with semantic embeddings and attention to classify log sequences under noisy conditions.

LogCNN [18]: This supervised model applies convolutional neural networks to structured log representations to detect anomalies through local pattern extraction.

```
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: (NvOdmDevice) Error ModuleNotPresent:  (propagating from dvs/git/dirty/git-master_linux/camera-
partner/imager/src/devices/V4L2SensorViCsi.cpp, function initialize(), line 111)
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclDriverInitializeData: Unable to initialize driver v4l2_sensor
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclInitializeDrivers: error: Failed to init camera sub module v4l2_sensor
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclStartPlatformDrivers: Failed to start module drivers
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclDriver_V4L2_Focuser_Stub_Close: Invalid NULL input pPclDriver
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclStateControllerOpen: Failed ImagerGUID 0. (error 0xA000E)
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: NvPclOpen: PCL Open Failed. Error: 0xf
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: SCF: Error BadParameter: Sensor could not be opened. (in src/services/capture/CaptureServiceDeviceSensor.cpp, function
getSourceFromGuid(), line 726)
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: SCF: Error BadParameter:  (propagating from src/services/capture/CaptureService.cpp, function addSourceByGuid(), line 453)
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: SCF: Error BadParameter:  (propagating from src/api/CameraDriver.cpp, function addSourceByIndex(), line 347)
Oct 10 08:00:03 BP3234300020 nvargus-daemon[959]: SCF: Error BadParameter:  (propagating from src/api/CameraDriver.cpp, function getSource(), line 519)
Oct 10 08:00:03 BP3234300020 start_collectors.sh[88476]: Setting pipeline to PAUSED ...
Oct 10 08:00:03 BP3234300020 start_collectors.sh[88476]: Pipeline is live and does not need PREROLL ...
Oct 10 08:00:03 BP3234300020 start_collectors.sh[88476]: Setting pipeline to PLAYING ...
Oct 10 08:00:03 BP3234300020 start_collectors.sh[88476]: New clock: GstSystemClock
Oct 10 08:00:03 BP3234300020 start_collectors.sh[88476]: /GstPipeline:pipeline0/GstNvArgusCameraSrc:nvarguscamerasrc0.GstPad:src: caps = video/x-raw(memory:NVMM), width=(int)1920,
height=(int)1080, format=(string)NV12, framerate=(fraction)30/1
```

Fig. 2. Example of system logs.

TABLE II
LOG SEQUENCE SUMMARY

Dataset	# Log Events	Window size	Training Data		Testing Data	
			# Log seq	# Anomaly	# Log seq	# Anomaly
A	57	20 logs	746	84 (11.2%)	320	36 (11.2%)
		50 logs	298	64 (21.4%)	129	28 (21.7%)
		100 logs	149	53 (35.5%)	65	23 (35.3%)
		200 logs	74	40 (54%)	33	18 (54.5%)
B	1916	20 logs	20938	12208 (58.3%)	8974	5232 (58.3%)
		50 logs	8374	6780 (80.9%)	3591	2907 (80.9%)
		100 logs	4187	3748 (89.5%)	1796	1607 (89.4%)
		200 logs	2094	1982 (94.6%)	898	850 (94.6%)

V. EXPERIMENTAL DESIGN

A. Dataset Overview

Experiments were conducted using system logs from two SCiNe devices operating under different conditions and labeled using two strategies, as described in Section III-A. Each log entry includes a timestamp, device identifier, log source, and descriptive message. The datasets reflect the variability and complexity of real industrial logs. Example log entries containing anomaly-related keywords such as ”error” and ”failed” are shown in Figure 2.

B. Log Parsing

We used the Drain [30] log parser with its default parameter settings to convert the raw, unstructured log messages into a structured format because of its efficiency, scalability, and strong performance on heterogeneous industrial logs reported in prior studies. This parsing process extracts uniform log templates, facilitating later feature engineering and anomaly detection.

C. Log Grouping (Windowing)

We applied a fixed-size windowing method to both datasets, organizing the logs chronologically by their timestamps. Random shuffling was avoided to prevent data leakage. Window sizes of 20, 50, 100, and 200 logs were used across experiments, with additional sizes (10, 60, 120, 220) applied for selected deep learning models in Dataset B. A sequence was labeled anomalous if it contained at least one anomalous log entry. Table II summarizes the resulting sequence distributions.

D. Feature Representation

After log grouping, log sequences were transformed into feature vectors based on the type of model:

Machine Learning Models: Quantitative event-count vectors representing frequency of log templates within each sequence.

Deep Learning Models: Sequential representations, used by models like DeepLog, assign ordered indices to log events. Semantic representations, used by SpikeLog, PLELog, LogRobust, and LogCNN, leverage Word2Vec (often with TF-IDF) to capture contextual meaning. Models like LogAnomaly combine sequential vectors with event counts to capture both temporal and frequency patterns.

E. Anomaly Detection

All models were trained and evaluated using the extracted features under supervised, semi-supervised, unsupervised, and weakly supervised paradigms. This enables comparison under varying levels of label quality and dataset scale.

F. Evaluation Metrics

Anomaly detection is treated as a binary classification problem and evaluated using Precision (P), Recall (R), Specificity (S), and F1-score (F1). Precision $\frac{TP}{TP+FP}$ reflects the proportion of correctly identified anomalous sequences among all sequences predicted as anomalous. Recall $\frac{TP}{TP+FN}$ represents the model's ability to detect true anomalies. Specificity $\frac{TN}{TN+FP}$ measures the model's ability to correctly identify normal sequences. F1-score $\frac{2\times(\text{Precision}\times\text{Recall})}{\text{Precision}+\text{Recall}}$ provides a balanced evaluation of the model's accuracy by combining both precision and recall.

TABLE III
ACCURACY OF MACHINE LEARNING AND DEEP LEARNING MODELS

Metric	ML Method	Dataset A				Dataset B				DL Method	Dataset A				Dataset B			
		20 l	50 l	100 l	200 l	20 l	50 l	100 l	200 l		10 l	60 l	120 l	220 l	20 l	50 l	100 l	200 l
P	LR	0.937	1	1	0.625	0.997	0.989	0.987	0.994	SpikeLog	0.217	0.927	0.948	0.902	0.998	0.999	0.998	1
R		0.75	0.875	1	1	0.993	0.987	0.99	0.991		0.625	0.787	0.705	0.748	0.987	0.991	0.987	0.996
S		0.996	1	1	0.739	0.997	0.95	0.883	0.843		0.912	0.997	0.998	0.996	0.997	0.996	0.981	0.999
F1		0.833	0.933	**1**	0.769	**0.995**	0.988	0.988	**0.993**		0.322	**0.851**	0.808	0.818	0.993	**0.995**	0.992	**0.998**
P	SVM	0.941	1	1	1	1	1	1	1	PLELog	0.2	0.133	0.142	0.095	0.998	0.995	0.999	0.998
R		0.8	0.937	0.769	0.9	0.999	1	1	1		0.125	0.266	0.25	0.166	0.868	0.993	0.988	0.997
S		0.996	1	1	1	1	1	1	1		0.98	0.717	0.57	0.055	0.998	0.979	0.993	0.964
F1		0.864	**0.967**	0.869	0.947	0.999	**1**	**1**	**1**		0.153	0.177	**0.181**	0.121	0.929	**0.996**	0.993	**0.998**
P	DT	1	1	1	1	1	1	1	0.995	DeepLog	0.099	0.245	0.222	0.413	0.77	0.915	0.946	0.968
R		0.8	0.875	0.769	0.9	0.999	1	1	1		0.458	1	1	1	0.7	0.888	0.788	1
S		1	1	1	1	1	1	1	0.875		0.838	0.5	0	0.056	0.697	0.617	0.552	0
F1		0.888	0.933	0.869	**0.947**	**0.999**	**1**	**1**	**0.997**		0.162	0.394	0.363	**0.585**	0.738	0.901	0.86	**0.984**
P	PCA	0.428	0.666	0.714	0.8	0.249	0.22	0.666	0.8	LogAnomaly	0.078	0.141	0.255	0.413	0.654	0.896	0.946	0.984
R		0.45	0.375	0.384	0.4	0.012	0.009	0.032	0.055		0.375	1	1	1	0.841	0.81	0.788	0.978
S		0.96	0.973	0.961	0.956	0.945	0.952	0.847	0.625		0.83	0.011	0.167	0.056	0.364	0.565	0.552	0.536
F1		0.439	0.48	0.5	**0.533**	0.024	0.018	0.061	**0.103**		0.13	0.247	0.406	**0.585**	0.736	0.851	0.86	**0.981**
P	IF	0	1	1	1	0.758	0.517	1	1	LogRobust	0.875	0.357	1	1	0.589	1	0.999	1
R		0	0.25	0.153	0.2	0.026	0.005	0.029	0.256		0.583	0.333	1	1	1	0.997	0.998	1
S		0.993	1	1	1	0.988	0.992	1	1		0.997	0.902	1	1	0	1	0.988	1
F1		0	**0.4**	0.266	0.333	0.05	0.011	0.057	**0.408**		0.7	0.344	**1**	**1**	0.741	**0.998**	**0.998**	**1**
P	LC	0.22	0.22	0.295	0.346	0.561	0.697	0.95	0.974	LogCNN	0.923	1	1	1	1	1	1	1
R		0.65	0.937	1	0.9	0.074	0.988	0.859	0.942		0.5	0.933	1	0.916	0.996	1	1	1
S		0.846	0.53	0.403	0.26	0.916	0.39	0.578	0.343		0.998	1	1	1	1	1	1	1
F1		0.329	0.357	0.456	**0.5**	0.132	0.817	0.902	**0.958**		0.648	**0.965**	**1**	**0.956**	**0.998**	**1**	**1**	**1**

G. Implementation and Environment

The datasets were divided into 70% for training and 30% for testing. Model parameters were tuned to achieve optimal performance while maintaining consistent training conditions across methods. No oversampling or synthetic balancing techniques were applied; instead, F1-score and Recall were emphasized because both datasets exhibit significant class imbalance. Experiments were conducted on a Linux server with an Intel® Core™ i9-13900 CPU, 64 GB RAM, and an NVIDIA RTX A2000 GPU running Ubuntu 24.04.1 LTS.

VI. RESULTS AND ANALYSIS

A. Research Question 1: How do traditional machine learning and deep learning models perform on industrial log data of varying sizes?

We evaluated six ML and six DL models on two SCiNe datasets: Dataset A (small, expert-labeled) and Dataset B (large, heuristically labeled). Logs were grouped using fixed windows (ML: 20–200 logs; DL: 10–220 logs). Performance was assessed using Precision (P), Recall (R), Specificity (S), and F1-score (F1). Table III shows the results. (Note: "20 l," "50 l, etc indicating windows of 20 logs, 50 logs, respectively.)

1) Accuracy of Machine Learning Models: Supervised ML methods (LR, SVM, DT) consistently outperformed unsupervised approaches across both datasets. SVM achieved near-perfect performance on Dataset B across all window sizes $F1 \approx 1$, indicating strong separability when quantitative event-count features are available. Performance generally improved with larger windows due to richer context, though extremely large windows occasionally diluted anomaly-specific signals. Unsupervised ML models showed higher variance. PCA and Isolation Forest struggled on Dataset B, reflecting sensitivity to noise and class imbalance. LogClustering benefited from larger windows and achieved competitive F1-scores (up to 0.958), but remained highly dependent on window configuration.

2) Accuracy of Deep Learning Models: DL models exhibited greater diversity in behavior across learning paradigms. Supervised models (LogCNN, LogRobust) achieved the most consistent and highest accuracy across both datasets, with LogCNN producing near-perfect scores across window sizes. Weakly supervised SpikeLog showed stable and strong performance on Dataset B despite noisy labels, demonstrating robustness at scale. Semi-supervised PLELog underperformed on Dataset A but achieved excellent results on Dataset B, highlighting its dependence on data volume and diversity. Unsupervised models (DeepLog, LogAnomaly) showed limited precision on Dataset A but improved markedly with larger windows and dataset size, achieving high recall on Dataset B. Some unsupervised models produced near-zero precision or F1-scores under certain window configurations, particularly on Dataset B. This behavior is likely due to severe class imbalance, noisy heuristic labels, and difficulty learning stable sequential patterns from highly variable logs. In smaller windows, sequential models such as DeepLog and LogAnomaly may also lack sufficient contextual information to distinguish normal and anomalous behavior.

In summary, supervised models excel when high-quality labels are available, while weakly and semi-supervised models provide practical alternatives in large, noisy industrial settings.

B. Research Question 2: What are the trade-offs between different anomaly detection models in terms of robustness, scalability, and computational efficiency when applied to industrial log data?

To understand the trade-offs among supervised, semi-supervised, unsupervised, and weakly supervised anomaly detection models, we evaluated their performance on real-

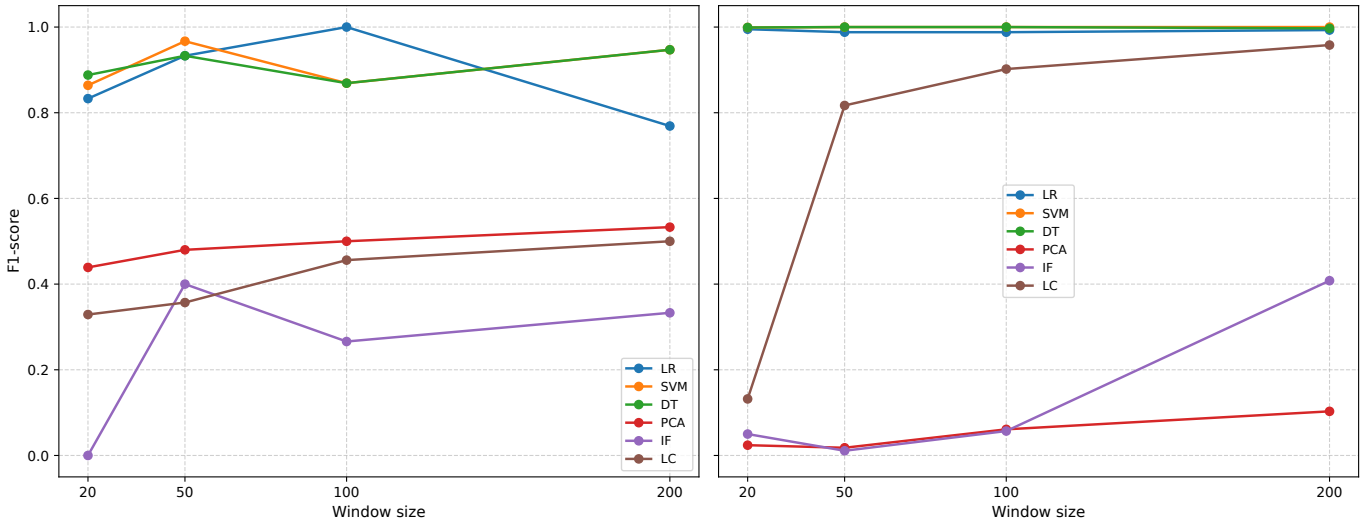

Fig. 3. F1-score of ML models across varying window sizes.

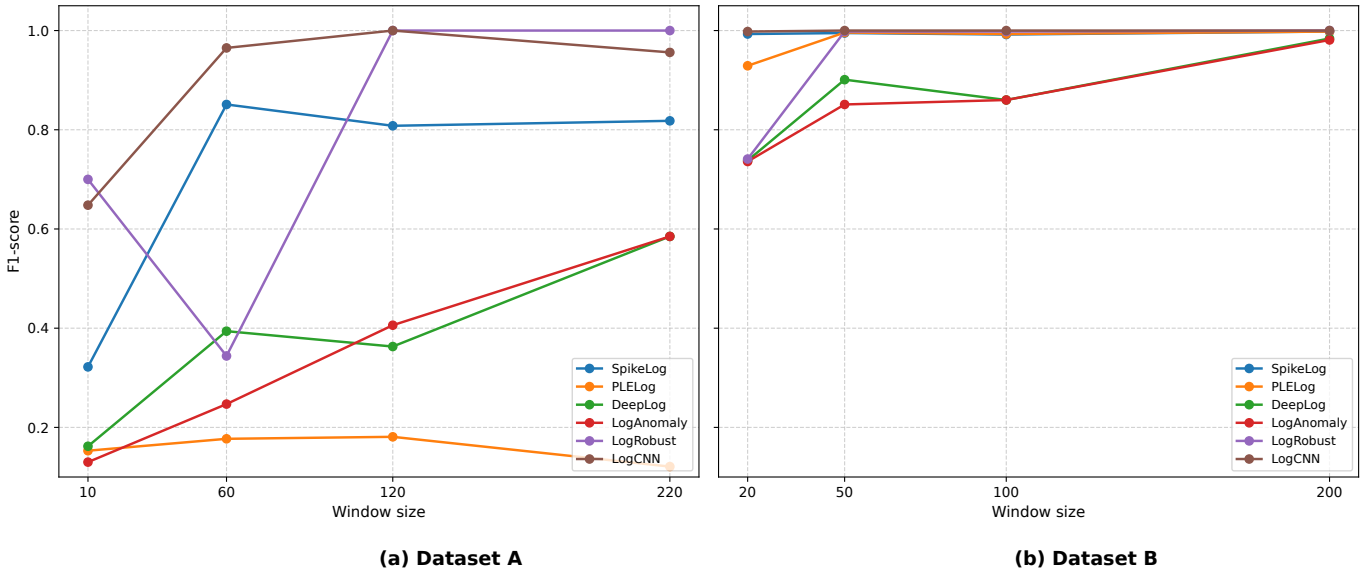

Fig. 4. F1-score of DL models across varying window sizes.

world industrial log data with respect to robustness, scalability, and computational efficiency.

1) Robustness and Scalability: To evaluate robustness and scalability, we analyzed how F1-score of supervised, semi-supervised, unsupervised, and weakly supervised models vary with window sizes across two SCiNe log datasets. The results are shown separately for the ML and DL models in Figure 3 and Figure 4. Supervised ML models, particularly SVM, demonstrated strong robustness with minimal sensitivity to window size. Among DL models, LogCNN showed the highest robustness and scalability, maintaining near-perfect performance across datasets. SpikeLog and LogAnomaly benefited from longer windows, while DeepLog and PLELog exhibited greater variability. In general, increasing window size improved performance by providing richer temporal context, though gains diminished beyond moderate window lengths.

2) Computational Efficiency: To assess the practical usability of the models, we recorded their training and testing times using representative window sizes of 100 logs and 120 logs, chosen because most models achieved optimal or near-optimal accuracy at this size.

ML models were computationally lightweight. SVM and DT required negligible training and testing time, even on Dataset B. PCA and Isolation Forest incurred higher costs, while LogClustering was the most expensive ML method due to clustering overhead Table IV summarizes the results. DL

SequenceId	EventId	Month	Date	Time	Device	Compo	Content
4	08b49568	Oct	4	0:33:21	buspas10015	aziot-edge	2023-10-04T04:33:21Z [INFO] - [mgmt] - - - [2023-10-04 04:33:21.654101577 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)
4	d12b8143	Oct	4	0:33:21	buspas10015	kernel	[3368562.301100] pcieport 0000:00:01.0: can't find device of ID0010
4	913e3e80	Oct	4	0:33:21	buspas10015	kernel	[3368562.654835] pcieport 0000:00:01.0: AER: Corrected error received: id=0010
4	89f32daa	Oct	4	0:33:25	buspas10015	kernel	[3368566.411391] pcieport 0000:00:01.0: [12] Replay Timer Timeout
4	08b49568	Oct	4	0:33:26	buspas10015	aziot-edge	2023-10-04T04:33:26Z [INFO] - [mgmt] - - - [2023-10-04 04:33:26.823354320 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)
4	913e3e80	Oct	4	0:33:27	buspas10015	kernel	[3368568.038574] pcieport 0000:00:01.0: AER: Corrected error received: id=0010
4	08b49568	Oct	4	0:33:31	buspas10015	aziot-edge	2023-10-04T04:33:31Z [INFO] - [mgmt] - - - [2023-10-04 04:33:31.982200135 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)
4	89f32daa	Oct	4	0:33:31	buspas10015	kernel	[3368572.308556] pcieport 0000:00:01.0: [12] Replay Timer Timeout
4	913e3e80	Oct	4	0:33:32	buspas10015	kernel	[3368573.202917] pcieport 0000:00:01.0: AER: Corrected error received: id=0010
4	08b49568	Oct	4	0:33:37	buspas10015	aziot-edge	2023-10-04T04:33:37Z [INFO] - [mgmt] - - - [2023-10-04 04:33:37.116264595 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)
4	89f32daa	Oct	4	0:33:37	buspas10015	kernel	[3368577.831592] pcieport 0000:00:01.0: [12] Replay Timer Timeout
4	913e3e80	Oct	4	0:33:37	buspas10015	kernel	[3368577.912244] pcieport 0000:00:01.0: AER: Corrected error received: id=0010
4	08b49568	Oct	4	0:33:42	buspas10015	aziot-edge	2023-10-04T04:33:42Z [INFO] - [mgmt] - - - [2023-10-04 04:33:42.260566852 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)
4	d12b8143	Oct	4	0:33:42	buspas10015	kernel	[3368583.052149] pcieport 0000:00:01.0: can't find device of ID0010
4	913e3e80	Oct	4	0:33:42	buspas10015	kernel	[3368583.433052] pcieport 0000:00:01.0: AER: Corrected error received: id=0010
4	d12b8143	Oct	4	0:33:45	buspas10015	kernel	[3368586.334770] pcieport 0000:00:01.0: can't find device of ID0010
4	08b49568	Oct	4	0:33:47	buspas10015	aziot-edge	2023-10-04T04:33:47Z [INFO] - [mgmt] - - - [2023-10-04 04:33:47.391539776 UTC] "GET /modules?api-version=2020-07-07 HTTP/1.1" 200 OK 8638 "-" "-" auth_id(-)

Fig. 5. Detected anomalies in a sequence.

TABLE IV
RUN TIME OF ML MODELS

ML Model	Dataset A (100 logs)		Dataset B (100 logs)	
	Training Time (sec)	Testing Time (sec)	Training Time (sec)	Testing Time (sec)
LR	0.0019	0.00114	0.211	0.006
SVM	0.00044	0.00093	**0.017**	**0.0032**
DT	**0.00042**	**0.00091**	0.041	0.0035
PCA	**0.0014**	**7.00E-05**	2.27	0.056
IF	0.049	0.0024	**0.761**	**0.0088**
LC	0.028	0.015	20.6	6.74

TABLE V
RUN TIME OF DL MODELS

DL Model	Dataset A (120 logs)		Dataset B (100 logs)	
	Training Time	Testing Time	Training Time	Testing Time
SpikeLog	10 min 21 sec	56.8 sec	24 min 4 sec	2 min 46.96 sec
PLELog	**3.00 sec**	**0.11 sec**	22.51 sec	**0.7 sec**
DeepLog	10 sec	0.935 sec	**2.5 sec**	2.17 sec
LogAnomaly	11 sec	0.934 sec	13 sec	6.07 sec
LogRobust	1 min 4 sec	0.861 sec	43 sec	2.16 sec
LogCNN	32 sec	0.825 sec	11 min 34 sec	25.17 sec

models were significantly more resource-intensive. PLELog and DeepLog were comparatively efficient, whereas SpikeLog and LogCNN incurred substantial training times. Table V shows the results. These results highlight a clear trade-off: DL models offer superior accuracy and generalization at the cost of increased computation, while ML models are better suited for resource-constrained or edge deployments.

C. Research Question 3: How can predictive analysis of SCiNe logs improve transit system reliability and enable proactive maintenance?

To evaluate practical impact, we performed an offline cross-dataset experiment by training a Decision Tree on Dataset B and testing on Dataset A. With a window size of 160, the model detected a true positive anomaly with zero false positives, demonstrating the feasibility of cross-device generalization when sufficient context is available. Although recall remained limited, this result indicates potential for early warning. Assuming a typical SCiNe failure causes approximately four hours of downtime, detecting anomalies one hour in advance could potentially reduce downtime significantly in practical deployments. Even modest detection rates, when scaled across multiple devices, could yield substantial operational benefits. Figure 5 illustrates a representative anomaly sequence, showing deviation prior to failure. While detection is not exhaustive, the results support the viability of predictive maintenance using lightweight models.

VII. DISCUSSION

A. Key Insights and Practical Implications

Based on the results presented in Section VI, we address several important insights that enhance the contribution of this work.

a. Trade-offs Across Learning Paradigms: Supervised models deliver the highest accuracy but depend on costly expert labels. Weakly and semi-supervised models better tolerate noisy labels and scale to large datasets, making them suitable for real-world transit environments. Purely unsupervised methods offer low labeling cost but require careful windowing and often underperform in noisy settings.

b. Robustness and Efficiency: Models with stable performance across datasets and window sizes (e.g., SVM, LogCNN) are preferable for deployment. Traditional ML methods offer excellent efficiency and are suitable for edge or low-power environments. DL models, while computationally expensive, better capture semantics and long-term dependencies, making them appropriate for centralized monitoring.

c. Deployment Considerations: Although tested offline, the models showed real potential for early fault detection in transit systems. Even a single early warning across many SCiNe devices could support earlier intervention and reduce operational downtime in practical transit deployments if intervention occurs an hour before failure. So, a hybrid approach is recommended: supervised models like SVM and LogCNN work well where labels are available, while weakly and semi-supervised models like SpikeLog and PLELog suit large-scale systems with limited labeling. This balances accuracy, scalability, and practical deployment for smart transit maintenance.

B. Limitations and Future Work

This study is limited to offline evaluation due to the absence of real-time SCiNe log aggregation. Future work will investigate robustness against heuristic labeling bias, including keyword masking and alternative labeling strategies, as well as statistical validation across multiple training runs. Window-size sensitivity and heuristic labeling noise also warrant further

investigation, while explainability and improved annotation techniques remain important directions for future work.

VIII. Conclusion

This work demonstrates that log-based anomaly detection can improve reliability in smart transit systems. Through an empirical comparison of ML and DL models on real-world SCiNe logs, we show that performance depends strongly on data scale, label quality, and computational constraints. Supervised models achieved the highest accuracy, while weakly and semi-supervised methods provided scalable alternatives for real-world deployment. The findings provide actionable guidance for integrating predictive maintenance into future SCiNe deployments, reducing downtime and improving service reliability without extensive manual supervision.

IX. Acknowledgements

This research was conducted in partnership with BusPas Inc. and the Mitacs Accelerate Program, and we sincerely appreciate their assistance in providing access to data and resources.

References

[1] B. Inc., "Real-world applications for remarkable innovations [use cases]," n.d.

[2] N. A. Nipa, N. Bouguila, and Z. Patterson, "A comparative study of log-based anomaly detection methods in real-world system logs," in *Proceedings of the 10th International Conference on Internet of Things, Big Data and Security - IoTBDS*, pp. 141–152, INSTICC, SciTePress, 2025.

[3] M. Bawaneh and V. Simon, "Anomaly detection in smart city traffic based on time series analysis," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, IEEE, 2019.

[4] F. Van Wyk, Y. Wang, A. Khojandi, and N. Masoud, "Real-time sensor anomaly detection and identification in automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1264–1276, 2019.

[5] J. Islam, J. P. Talusan, S. Bhattacharjee, F. Tiausas, S. M. Vazirizade, A. Dubey, K. Yasumoto, and S. K. Das, "Anomaly based incident detection in large scale smart transportation systems," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 215–224, IEEE, 2022.

[6] Q. Cheng, K. Hong, K. Huang, and Z. Liu, "Evaluating effectiveness and identifying appropriate methods for anomaly detection in intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, 2025.

[7] N. A. Nipa, N. Bouguila, and Z. Patterson, "Log-based anomaly detection without ground truth: Evaluating weakly supervised, semi-supervised, and unsupervised deep learning approaches," in *Accepted for presentation at the 34th IEEE International Symposium on Industrial Electronics (ISIE), 2025*, 2025.

[8] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 215–224, 2016.

[9] L. Bao, Q. Li, P. Lu, J. Lu, T. Ruan, and K. Zhang, "Execution anomaly detection in large-scale systems through console log analysis," *Journal of Systems and Software*, vol. 143, pp. 172–186, 2018.

[10] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 60–70, 2018.

[11] J. Wang, Y. Tang, S. He, C. Zhao, P. K. Sharma, O. Alfarraj, and A. Tolba, "Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things," *Sensors*, vol. 20, no. 9, p. 2451, 2020.

[12] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 36–43, IEEE, 2004.

[13] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 583–588, IEEE, 2007.

[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008.

[15] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, 2009.

[16] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[17] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 807–817, 2019.

[18] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 151–158, IEEE, 2018.

[19] Q. Du, L. Zhao, J. Xu, Y. Han, and S. Zhang, "Log-based anomaly detection with multi-head scaled dot-product attention mechanism," in *Database and Expert Systems Applications: 32nd International Conference, DEXA 2021, Virtual Event, September 27–30, 2021, Proceedings, Part I 32*, pp. 335–347, Springer, 2021.

[20] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1448–1460, IEEE, 2021.

[21] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[22] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298, 2017.

[23] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.," in *IJCAI*, vol. 19, pp. 4739–4745, 2019.

[24] L. Zhang, W. Li, Z. Zhang, Q. Lu, C. Hou, P. Hu, T. Gui, and S. Lu, "Logattn: Unsupervised log anomaly detection with an autoencoder based attention mechanism," in *International conference on knowledge science, engineering and management*, pp. 222–235, Springer, 2021.

[25] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 3068–3072, 2021.

[26] J. Qi, Z. Luan, S. Huang, C. Fung, H. Yang, and D. Qian, "Spikelog: log-based anomaly detection via potential-assisted spiking neuron network," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 12, pp. 9322–9335, 2023.

[27] H. Guo, Y. Guo, J. Yang, J. Liu, Z. Li, T. Zheng, L. Zheng, W. Hou, and B. Zhang, "Loglg: Weakly supervised log anomaly detection via log-event graph construction," in *International conference on database systems for advanced applications*, pp. 490–501, Springer, 2023.

[28] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European conference on Computer systems*, pp. 111–124, 2010.

[29] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 102–111, 2016.

[30] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*, pp. 33–40, IEEE, 2017.