A comprehensive study of machine learning techniques for log-based anomaly detection

Shan Ali[1] · Chaima Boufaied[2] · Domenico Bianculli[3] · Paula Branco[1] ·
Lionel Briand[1,4]

Abstract

Growth in system complexity increases the need for automated techniques dedicated to different log analysis tasks such as Log-based Anomaly Detection (LAD). The latter has been widely addressed in the literature, mostly by means of a variety of deep learning techniques. However, despite their many advantages, that focus on deep learning techniques is somewhat arbitrary as traditional Machine Learning (ML) techniques may perform well in many cases, depending on the context and datasets. In the same vein, semi-supervised techniques deserve the same attention as supervised techniques since the former have clear practical advantages. Further, current evaluations mostly rely on the assessment of detection accuracy. However, this is not enough to decide whether or not a specific ML technique is suitable to address the LAD problem in a given context. Other aspects to consider include training and prediction times as well as the sensitivity to hyperparameter tuning, which in practice matters to engineers. In this paper, we present a comprehensive empirical study, in which we evaluate a wide array of supervised and semi-supervised, traditional and deep ML techniques w.r.t. four evaluation criteria: detection accuracy, time performance, sensitivity of detection accuracy and time performance to hyperparameter tuning. Our goal is to provide much stronger and comprehensive evidence regarding the relative advantages and drawbacks of alternative techniques for LAD. The experimental results show that supervised traditional and deep ML techniques fare similarly in terms of their detection accuracy and prediction time on most of the benchmark datasets considered in our study. Moreover, overall, sensitivity analysis to hyperparameter tuning with respect to detection accuracy shows that supervised traditional ML techniques are less sensitive than deep learning techniques. Further, semi-supervised techniques yield significantly worse detection accuracy than supervised techniques.

Keywords Anomaly detection · Log · Machine learning · Deep learning

1 Introduction

Systems typically produce execution logs recording execution information about the state of the system, inputs and outputs, and operations performed. These logs are typically used

Extended author information available on the last page of the article

Springer

during testing campaigns to detect failures, or after deployment and at runtime, to identify abnormal system behaviors; these are referred to as *anomalies*.

The Log-based Anomaly Detection (LAD) problem consists of detecting anomalies from execution logs recording normal and abnormal system behaviors. It has been widely addressed in the literature by means of deep learning techniques (Du et al. 2017; Zhu et al. 2020; Xie et al. 2020; Huang et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Zhang et al. 2019; Lu et al. 2018; Wang et al. 2022; Le and Zhang 2021; Guo et al. 2021; Qi et al. 2023; Chen et al. 2022; Qi et al. 2022; Catillo et al. 2022; Zhang et al. 2023; Almodovar et al. 2023; Xia et al. 2021; Hashemi and Mäntylä 2021; Du et al. 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Han et al. 2021; Lee et al. 2023; Chen et al. 2021; Le and Zhang 2022; Wu et al. 2023; Yu et al. 2024; Li et al. 2024; Xiao et al. 2024; Guo et al. 2024; Zang et al. 2024; Yin et al. 2024; Lin et al. 2024; Gong et al. 2024; Wang et al. 2024). Since logs are typically unstructured, many of the supervised and semi-supervised LAD techniques (except NeuralLog Le and Zhang 2021, LayerLog Zhang et al. 2023, Logfit Almodovar et al. 2023, LogGD Xie et al. 2022, ContexLog Xiao et al. 2024 and SaRLog Adeba et al. 2024) rely on log parsing (e.g., using Drain He et al. 2017) to identify and extract log templates (also called log events Landauer et al. 2024; Yang et al. 2024; Guo et al. 2024; Yu et al. 2024; Yang et al. 2021; Qi et al. 2023; Xie et al. 2022; Gong et al. 2024; Lee et al. 2023; Yin et al. 2024; Li et al. 2024; Huang et al. 2023; Chen et al. 2022 or log keys Chen et al. 2021; Du et al. 2017; Lu et al. 2018; Guo et al. 2021; Zhang et al. 2023; Han et al. 2021). The extracted templates can be grouped into different windows (i.e., fixed, sliding, or session windows) forming different template sequences.

Features first need to be extracted from different template sequences to enable the use of machine learning (ML) techniques. DeepLog (Du et al. 2017), for example, extracts features using sequential vectors where each component is an index-based encoding of a single template within a template sequence. The remaining deep learning techniques rely on semantic vectors to capture the semantic information from the different templates within a sequence. Semantic vectors are obtained by means of different semantic vectorization techniques such as Template2Vec (Meng et al. 2019), word2vec (Han et al. 2021) (augmented by a Post-Processing Algorithm (PPA) Wang et al. 2022), FastText (Zhang et al. 2019) complemented by Term Frequency - Inverse Document Frequency (TF-IDF Salton and Buckley 1988), Recurrent Neural Network (RNN)-based encoders (e.g., the attention Bi-directional Long Short-Term Memory Bi-LSTM encoder LogVec Zhang et al. 2023) and Transformer-based encoders (Huang et al. 2020; Guo et al. 2021; Le and Zhang 2021). Based on the above features, existing deep learning techniques detect log anomalies using different types of neural networks such as Recurrent Neural Network (RNN) (Du et al. 2017; Zhu et al. 2020; Xie et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Zhang et al. 2019; Qi et al. 2023; Li et al. 2022; Zhang et al. 2023; Han et al. 2021; Gong et al. 2024; Nguyen et al. 2024), Convolutional Neural Network (CNN) (Lu et al. 2018; Wang et al. 2022; Chen et al. 2022; Hashemi and Mäntylä 2021; Yin et al. 2024), Transformer-based deep learning models (Huang et al. 2020; Le and Zhang 2021; Guo et al. 2021; Almodovar et al. 2023; Du et al. 2021; Huang et al. 2023; Lee et al. 2023; Guo et al. 2024; Zang et al. 2024; Xiao et al. 2024; Adeba et al. 2024), Auto Encoders (AE) (Catillo et al. 2022), Graph Neural Network (GNN) (Xie et al. 2022; Li et al. 2024; Wang et al. 2024), and Generative Adversarial Network (GAN) (Xia et al. 2021; Qi et al. 2022; Lin et al. 2024).

Some empirical studies (Le and Zhang 2021, 2022; Yin et al. 2024) investigate the impact of log parsing methods on the detection accuracy of deep learning anomaly detection techniques. Others (Zhang et al. 2022) study the impact of different semantic vectorization techniques on the detection accuracy of deep learning techniques.

Detection accuracy has also been further evaluated to assess the impact of several factors (Le and Zhang 2022), such as training data selection strategies, data grouping methods, data imbalance and data noise (e.g., log mislabelling). High detection accuracy often comes with longer training and prediction times, which can be a challenge at run-time for large-scale applications. In such cases, a model with slightly lower detection accuracy but faster time performance may be more practical. The trade-off between detection accuracy and time performance depends on the specific needs of the application, such as the need for real-time detection or available computational resources. Thus, some empirical studies (e.g., Huang et al. 2020; Yang et al. 2021; Wang et al. 2022; Le and Zhang 2021; Guo et al. 2024; Li et al. 2024; Yin et al. 2024; Lin et al. 2024; Xiao et al. 2024) assess the time performance of alternative LAD techniques. Further, a technique with an overall high detection accuracy and practical time performance, may be very sensitive to hyperparameter settings and exhibit widely different results across datasets.

Based on the above discussion, we contend that four evaluation criteria should be systemically considered to assess the overall performance of any ML technique for LAD, regardless of the type of learning they involve. These criteria are i) detection accuracy, ii) time performance, sensitivity of iii) detection accuracy and iv) time performance to different hyperparameter settings.

Most of the existing empirical studies focus on supervised deep learning techniques (Huang et al. 2020; Liu et al. 2021; Zhang et al. 2019; Le and Zhang 2021; Du et al. 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Zhang et al. 2023; Hashemi and Mäntylä 2021; Lee et al. 2023; Chen et al. 2022; Han et al. 2021). Although many studies (Huang et al. 2020; Liu et al. 2021; Zhang et al. 2019; Le and Zhang 2021; Guo et al. 2021; Catillo et al. 2022; Zhang et al. 2023; Du et al. 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Yu et al. 2024; Li et al. 2024; Xiao et al. 2024; Guo et al. 2024; Wang et al. 2024) compare some deep learning techniques to some traditional ones, none of these studies systematically evaluates these techniques w.r.t. the four aforementioned evaluation criteria. Indeed, the strong focus on deep learning is rather arbitrary as traditional ML may indeed fare well in many situations and offer practical advantages. Further, including semi-supervised learning in such studies is also important given the usual scarcity of anomalies in many logs.

In this paper, we report on the first comprehensive, systematic empirical study that includes not only deep learning techniques but also traditional ones, both supervised and semi-supervised, considering the four aforementioned evaluation criteria. More precisely, we systematically evaluate and compare, on seven benchmark datasets, a) supervised traditional (Support Vector Machine SVM Cortes and Vapnik 1995 and Random Forest RF Breiman 2001) and deep learning techniques (Long Short-Term Memory LSTM Hochreiter and Schmidhuber 1997, LogRobust Zhang et al. 2019 and NeuralLog Le and Zhang 2021), as well as b) semi-supervised traditional (One Class SVM OC-SVM Schölkopf et al. 2001) and deep learning techniques (DeepLog Du et al. 2017 and Logs2Graphs Li et al. 2024). We compare them in terms of i) detection accuracy, ii) time performance, sensitivity of iii) detection accuracy and iv) time performance to hyperparameter tuning.

Our experimental results show that supervised traditional and deep ML techniques perform very closely in terms of detection accuracy and prediction time. Further, supervised traditional ML techniques show less sensitivity to hyperparameter tuning than deep learning techniques. Last, semi-supervised techniques, both traditional and deep learning, do not fare well in terms of detection accuracy, when compared to supervised ones.

The results suggest that, despite the strong research focus on deep learning solutions for LAD, traditional ML techniques such as RF can fare much better with respect to our four criteria and therefore be a solution of choice in practice. Semi-supervised techniques,

however, do not seem to be a good option at this point, resulting in practical challenges to collect sufficient anomalous log data.

The rest of the paper is organized as follows. Section 2 explains and formalizes the background concepts used in the rest of the paper and provides a brief overview of the ML techniques considered in the study. Section 3 reports on state-of-the-art empirical studies that are related to the study presented in the paper. Section 4 explains the semantic vector embedding techniques we used to extract features from input log data. Section 5 describes the design of our empirical study. Section 6 reports and discusses the results of the different supervised and semi-supervised ML techniques. Section 7 concludes the paper, providing directions for future work.

2 Background

In this section, we first introduce the different concepts used in the remainder of the paper (§ 2.1). We then briefly describe the common workflow of LAD using DL models(§ 2.2). Finally, we describe three traditional ML techniques (§ 2.3) and five deep learning techniques (§ 2.4) that have been previously used to address the LAD problem and are considered in our study.

2.1 Execution Logs

Information about system executions is usually stored in log files, called execution logs. These logs help with troubleshooting and hence help system engineers understand the behavior of the system under analysis across its different executions. We distinguish between normal and abnormal system executions. The former represents an expected behavior of the system, while the latter represents an anomalous system behavior, possibly leading to a failure. These system executions are therefore stored in labeled execution logs, where the label refers to whether the execution is normal or not.

An execution log can be defined as a sequence of consecutive log entries that capture the behavior of the system over a given time period. A log entry contains: i) an ID; ii) the timestamp at which the logged event was recorded; iii) the log message denoting the occurrence of an event, called *log event occurrence* (Le and Zhang 2021; Zhang et al. 2019) (also called occurrence of log template Wu et al. 2023); and iv) the parameter value(s) recorded for that specific log event occurrence. Figure 1 shows an example of an execution log containing ten log entries (seven of which are displayed in the figure). For instance, the log entry with ID=4 in the figure contains the timestamp "16:05:14", an occurrence of log event *gyroscope_sensor_reading*, and the corresponding parameter values (0.0012, −0.0086 and 0.0020). The different log entries collected in a log are chronologically ordered w.r.t. their recorded timestamps.

An *execution path* (Xie et al. 2020; Le and Zhang 2022; Chen et al. 2021; Catillo et al. 2022; Zhang et al. 2019; Du et al. 2017; Li et al. 2022; Landauer et al. 2023; Wu et al. 2023; Huang et al. 2020; Lu et al. 2018; Yu et al. 2024; Xiao et al. 2024) is the projection with respect to the log event occurrences of the sequence of log entries recorded in the log. For instance, let us consider the first three log entries in Fig. 1. The execution path obtained from these entries is the sequence of the three corresponding log event occurrences (*battery_filtered_voltage_reading*, *gyroscope_sensor_reading*, *ekf2_attitude_pitch_reading*). An execution path is called *anomalous* (i.e., containing execution path log anomalies) when the order of its sequence of log event occurrences is unexpected.

ID	Timestamp	Log Event Occurrence	Parameter Value
1	$16:05:12$	*battery_filtered_voltage_reading*	16.182
2	$16:05:13$	*gyroscope_sensor_reading*	$-0.0013,$ $-1.135,$ -0.002
3	$16:05:13$	*ekf2_attitude_pitch_reading*	-0.026
4	$16:05:14$	*gyroscope_sensor_reading*	$0.0012,$ $-0.0086,$ 0.0020
5	$16:05:14$	*battery_filtered_voltage_reading*	15.687
6	$16:05:19$	*battery_filtered_voltage_reading*	14.921
...	...	...	...
10	$16:05:48$	*ekf2_attitude_pitch_reading*	-0.025

Fig. 1 Example of an Execution Log

Given a log σ, we denote by $\sigma(l)$ the log event occurrence recorded at the entry of log σ having an ID equal to l. For instance, given the log in Fig. 1, we have $\sigma(2) = gyroscope\_sensor\_reading$. We introduce a word-based tokenization function W that, given a log event occurrence as input, returns the sequence of words contained in the log event occurrence. For instance, $W(\sigma(2)) = (gyroscope, sensor, reading)$.

2.2 Common Workflow of LAD Using Deep Learning (DL) models

As shown in Fig. 2, the common workflow of LAD using DL models (Le and Zhang 2022; Wu et al. 2023; Zhang et al. 2019; Wang et al. 2022) includes several steps: log parsing, log
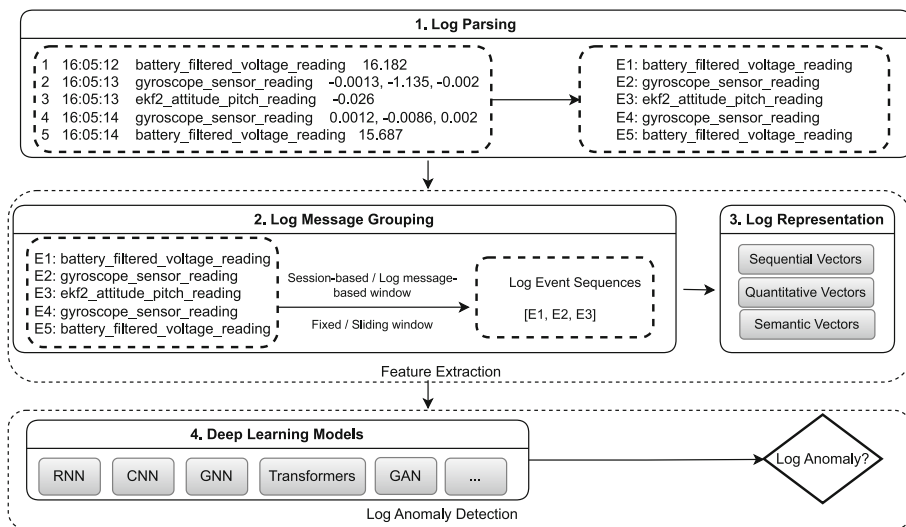

Fig. 2 Common Workflow of LAD using Deep Learning Models

message grouping, log representation, and log anomaly detection through appropriate deep learning models.

Logs are typically unstructured; they contain ID, timestamp, log event occurrences and the parameter values as shown in Fig. 1. To transform these unstructured logs into a structured format, many of the supervised and semi-supervised LAD techniques (except NeuralLog Le and Zhang 2021, LayerLog Zhang et al. 2023, Logfit Almodovar et al. 2023, LogGD Xie et al. 2022, ContexLog Xiao et al. 2024 and SaRLog Adeba et al. 2024) rely on log parsing techniques (e.g., Drain He et al. 2017) to extract log events from raw log messages. For instance, given the first log entry '1 16 : 05 : 12 $battery\_filtered\_voltage\_reading$ 16.182' in Fig. 1, the application of Drain leads to the following log event: '$battery\_filtered\_voltage\_reading$'.

The extracted log events can then be grouped into session-based windows (windows that correspond to log event occurrences recorded within a full system execution) or log message-based windows (Huang et al. 2020; Meng et al. 2019; Yang et al. 2021; Liu et al. 2021; Wang et al. 2022; Le and Zhang 2021, 2022) (windows determined by a specific number of log messages), forming different log event sequences.

Logs are either collected from i) session-based datasets (see Section 5.2.1), where the log event sequences are labeled at the level of the full system execution or ii) log message-based datasets (section 5.2.2), in which the labeling process is done at the level of individual log messages, without providing a clear indication on how to group them into log event sequences. Therefore, a log message grouping step (Landauer et al. 2023) is necessary for such datasets. Log event sequences from log message-based datasets are either created using log message-based windows or timestamp-based windows[1] (Qi et al. 2023; Guo et al. 2021; Le and Zhang 2022). Each of these log message-based grouping strategies can be further split into fixed and sliding windows.

Once the log event sequences are formed, features need to be extracted to enable the use of ML techniques. This can involve i) encoding log event sequences into vector formats such as sequential vectors (where each component is an index-based encoding of a single log event within a log event sequence Du et al. 2017), quantitative vectors (each component is the occurrences of each log event within a log event sequence Le and Zhang 2022), or semantic vectors (where each component captures the semantic information from the different log events within a log event sequence Zhang et al. 2019). These vector representations capture the underlying patterns and semantics of the logs and enable the model to understand differences and similarities between log event sequences, which is crucial for accurately identifying log anomalies (see Section 4). Finally, the numerical representations are fed into the corresponding DL models, such as RNN (Du et al. 2017; Zhu et al. 2020; Xie et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Zhang et al. 2019; Qi et al. 2023; Xia et al. 2021; Li et al. 2022; Zhang et al. 2023; Han et al. 2021; Gong et al. 2024; Nguyen et al. 2024), CNN (Lu et al. 2018; Wang et al. 2022; Chen et al. 2022; Hashemi and Mäntylä 2021; Yin et al. 2024), Transformers (Huang et al. 2020; Le and Zhang 2021; Guo et al. 2021; Almodovar et al. 2023; Du et al. 2021; Huang et al. 2023; Lee et al. 2023; Guo et al. 2024; Zang et al. 2024; Xiao et al. 2024; Adeba et al. 2024), GNN (Xie et al. 2022; Li et al. 2024; Wang et al. 2024)or GAN (Xia et al. 2021; Qi et al. 2022; Lin et al. 2024) to detect log anomalies.

---

[1] Windows delimited by log messages, where the time elapsed between the timestamp of the first and the last log messages within the window is equal to the window size.

2.3 Traditional ML Techniques

We briefly describe three traditional ML techniques further used in this study: SVM (Cortes and Vapnik 1995), RF (Breiman 2001), and OC-SVM (Schölkopf et al. 2001). We selected these techniques since one or several of them have been used as alternatives in the evaluation of previous work on LAD (Huang et al. 2020; Zhang et al. 2019; Le and Zhang 2021; Liu et al. 2021; Guo et al. 2021; Qi et al. 2022; Catillo et al. 2022; Zhang et al. 2023; Li et al. 2022; Huang et al. 2023; Chen et al. 2021; Wu et al. 2023; Guo et al. 2024; Li et al. 2024; Wang et al. 2024; Xiao et al. 2024). Furthermore, an extensive analysis conducted in Fernández-Delgado et al. (2014), which evaluated 179 classifiers (including variants of RF, decision tree, and logistic regression) across 121 datasets, showed that RF and SVM tend to be the most accurate classifiers.

2.3.1 Support Vector Machine (SVM)

SVM is a supervised classification ML technique. The key component of SVM is the kernel function, which significantly affects classification accuracy. The Radial Basis kernel Function (RBF) is typically the default choice when the problem requires a non-linear model (i.e., non-linearly separable data). SVM is based on a hyperparameter γ that controls the distance of influence of a single training data point and a regularization hyperparameter C that is used to add a penalty to misclassified data points. SVM was used as an alternative supervised traditional ML technique in the evaluation of some of the state-of-the-art LAD techniques (Huang et al. 2020; Zhang et al. 2019; Le and Zhang 2021; Guo et al. 2024; Wang et al. 2024; Xiao et al. 2024). Several LAD studies (Zhang et al. 2019; Le and Zhang 2021; Zhang et al. 2023; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Wang et al. 2024) show good detection accuracy for SVM, when evaluated on commonly used benchmark datasets (i.e., HDFS, Hadoop, BGL, Thunderbird and Spirit), which we also consider in this empirical study (see Section 5.2).

2.3.2 Random Forest (RF)

RF is a supervised classification ML technique. Two main hyperparameters can impact its accuracy: the number of decision trees dTr, a hyperparameter driven by data dimensionality, and the number of randomly selected features $sFeat$, a hyperparameter used in an individual tree. RF is used as a supervised traditional ML technique in the evaluation of a few LAD techniques (e.g., LogNads Liu et al. 2021, AdAnomaly Qi et al. 2022) and showed a better detection accuracy than many other alternative techniques, when evaluated on the HDFS, BGL and OpenStack public benchmark datasets.

2.3.3 One-class SVM (OC-SVM)

OC-SVM (a variant of SVM) is a semi-supervised classification ML technique. It has the same hyperparameters as SVM. Anomaly detection using OC-SVM requires building a feature matrix from the normal input. Unlike the unbounded SVM hyperparameter C, the regularization hyperparameter ν of OC-SVM is lower bounded by the fraction of support vectors (i.e., minimum percentage of data points that can act as support vectors). Based on experiments conducted in some recent LAD studies (Catillo et al. 2022; Zhang et al. 2023), OC-SVM showed to be an accurate semi-supervised technique, when evaluated on the HDFS, Hadoop and BGL datasets.

2.4 Log-based Deep Learning Techniques

Over the years, many studies have used deep learning for LAD (Du et al. 2017; Zhu et al. 2020; Xie et al. 2020; Huang et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Zhang et al. 2019; Lu et al. 2018; Wang et al. 2022; Le and Zhang 2021; Guo et al. 2021; Qi et al. 2023; Chen et al. 2022; Qi et al. 2022; Catillo et al. 2022; Zhang et al. 2023; Almodovar et al. 2023; Xia et al. 2021; Hashemi and Mäntylä 2021; Du et al. 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Han et al. 2021; Lee et al. 2023; Chen et al. 2021; Le and Zhang 2022; Wu et al. 2023; Yu et al. 2024; Li et al. 2024; Xiao et al. 2024; Guo et al. 2024; Zang et al. 2024; Yin et al. 2024; Lin et al. 2024; Gong et al. 2024; Wang et al. 2024). Out of the 42 deep learning techniques listed in Table 1, the majority of models addressing the LAD problem are based on RNNs, followed by Transformer-based models with 13 and 11 techniques, respectively. More in detail, many of the semi-supervised and supervised deep learning techniques in the literature rely on RNNs (e.g., Du et al. 2017; Meng et al. 2019; Zhang et al. 2019; Zhu et al. 2020; Liu et al. 2021; Qi et al. 2023; Han et al. 2021; Gong et al. 2024), and more specifically LSTM (Hochreiter and Schmidhuber 1997). Therefore, in our experiments, we considered the vanilla LSTM as a baseline technique, along with two deep RNN-based ML techniques: DeepLog (Du et al. 2017) and LogRobust (Zhang et al. 2019). We selected DeepLog because:

 i) it is the first method to address the LAD problem using deep learning, establishing a foundational benchmark;
 ii) it is the most cited technique in the literature (Zhu et al. 2020; Xie et al. 2020; Huang et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Qi et al. 2023; Wang et al. 2022; Guo et al. 2021; Chen et al. 2022; Le and Zhang 2022; Zhang et al. 2023; Qi et al. 2022; Almodovar et al. 2023; Xia et al. 2021; Hashemi and Mäntylä 2021; Li et al. 2022; Han et al. 2021; Chen et al. 2021; Yang et al. 2024; Zang et al. 2024; Xiao et al. 2024; Guo et al. 2024; Lu et al. 2018; Lee et al. 2023; Li et al. 2024; Yin et al. 2024; Lin et al. 2024; Gong et al. 2024; Wang et al. 2024; Nguyen et al. 2024; Landauer et al. 2024) (referenced in 32 out of the 42 studies listed in Table 1); and
 iii) it achieves an overall high detection accuracy in terms of *F1-score* on the benchmark datasets.

Similarly, LogRobust is the second mostly cited technique in the literature (Huang et al. 2020; Wang et al. 2022; Le and Zhang 2021; Qi et al. 2023; Le and Zhang 2022; Hashemi and Mäntylä 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Chen et al. 2021; Yang et al. 2024; Guo et al. 2024; Xiao et al. 2024; Zang et al. 2024; Yu et al. 2024; Wang et al. 2024; Yang et al. 2021; Lee et al. 2023; Du et al. 2021; Nguyen et al. 2024; Adeba et al. 2024) (referenced in 21 out of the 42 studies listed in Table 1), showing an overall high *F1-score* on the benchmark datasets.

Further, among the 11 transformer-based deep ML techniques (NeuralLog Le and Zhang 2021, ContexLog Xiao et al. 2024, HitAnomaly Huang et al. 2020, LogBERT Guo et al. 2021, LogFit Almodovar et al. 2023, LogAttention Du et al. 2021, HilBERT Huang et al. 2023, Hades Lee et al. 2023, LogFormer Guo et al. 2024, MLAD Zang et al. 2024 and SaRLog Adeba et al. 2024) in Table 1, NeuralLog and LogBERT are the most cited transformer-based ML techniques in the literature, with LogBERT being cited in seven studies (Almodovar et al. 2023; Qi et al. 2023; Huang et al. 2023; Hashemi and Mäntylä 2021; Zang et al. 2024; Lin et al. 2024; Yin et al. 2024) and NeuralLog in five studies (Hashemi and Mäntylä 2021; Xie et al. 2022; Xiao et al. 2024; Yu et al. 2024; Adeba et al. 2024). While LogBERT is cited more frequently than NeuralLog, the latter consistently demonstrates high

detection accuracy in terms of *F1-score* across the five studies where it was used, for the majority of the datasets. In contrast, LogBERT showed low detection accuracy across some of the benchmark datasets. NeuralLog was thus chosen as a baseline technique for our study.

Finally, among the three GNN-based deep ML techniques (LogGD Xie et al. 2022, Logs2Graphs Li et al. 2024 and LogGT Wang et al. 2024), the implementation of Logs2Graphs is the only one made publicly available. We therefore included it as a baseline, reflecting the potential of graph-based models for addressing the LAD problem.

In the following, we briefly describe three RNN-based (LSTM Hochreiter and Schmidhuber 1997, DeepLog Du et al. 2017 and LogRobust Zhang et al. 2019), one transformer-based (NeuralLog Le and Zhang 2021) and one GNN-based (Logs2Graphs Li et al. 2024) deep learning techniques that we evaluate in this study.

2.4.1 LSTM

LSTM (Hochreiter and Schmidhuber 1997) is a supervised deep learning technique. It is known for its capability to learn long-term dependencies between different sequence inputs. LSTM is mainly defined with the following hyperparameters: i) a loss function lF; ii) an optimizer opt; iii) the number of hidden layers hL; iv) the amount of training data utilized in a single iteration during the training process (i.e., the batch size) bS; and v) a number of epochs epN.

2.4.2 DeepLog

Deeplog (Du et al. 2017) is a semi-supervised technique. It relies on a forecasting-based detection model, i.e., anomalies are detected by predicting the next log event given preceding log events. Since it is based on LSTM, the same aforementioned LSTM hyperparameters apply: loss function lF, optimizer opt, hidden layers hL, batch size bS, and epochs epN. DeepLog has been used as an alternative technique in many past studies (Zhu et al. 2020; Xie et al. 2020; Huang et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Qi et al. 2023; Wang et al. 2022; Guo et al. 2021; Chen et al. 2022; Le and Zhang 2022; Zhang et al. 2023; Qi et al. 2022; Almodovar et al. 2023; Xia et al. 2021; Hashemi and Mäntylä 2021; Li et al. 2022; Han et al. 2021; Chen et al. 2021; Yang et al. 2024; Zang et al. 2024; Xiao et al. 2024; Guo et al. 2024; Lu et al. 2018; Lee et al. 2023; Li et al. 2024; Yin et al. 2024; Lin et al. 2024; Gong et al. 2024; Wang et al. 2024; Nguyen et al. 2024).

2.4.3 LogRobust

LogRobust (Zhang et al. 2019) is a supervised technique that relies on a classification-based detection model. LogRobust detects log anomalies by means of an attention-based Bi-LSTM model, allowing it to capture the contextual semantics across log events within a log event sequence. LogRobust is characterized by the same hyperparameters as LSTM and DeepLog, plus an additional hyperparameter, $nEpStop$, which is used to terminate the model training process if it does not improve after having reached a certain number of epochs. Further, the attention-based mechanism of LogRobust comes with an attention layer and Bi-LSTM weights that are incrementally updated by means of the gradient descent method (Kiefer and Wolfowitz 1952). LogRobust uses the FastText (Joulin et al. 2016) semantics-based embedding technique to encode the log messages from the input logs. LogRobust has been frequently used in past studies (Huang et al. 2020; Wang et al. 2022; Le and Zhang 2021; Qi

et al. 2023; Le and Zhang 2022; Hashemi and Mäntylä 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Chen et al. 2021; Yang et al. 2024; Guo et al. 2024; Xiao et al. 2024; Zang et al. 2024; Yu et al. 2024; Wang et al. 2024; Yang et al. 2021; Lee et al. 2023; Du et al. 2021; Nguyen et al. 2024; Adeba et al. 2024) and showed an overall high detection accuracy.

2.4.4 NeuralLog

NeuralLog (Le and Zhang 2021) is a transformer-based supervised classification technique that directly identifies log anomalies from unstructured logs without applying any log parsing technique to extract templates from input logs. In addition to the hyperparameters that characterize the RNN-based techniques, NeuralLog is also defined by the number of attention heads $attH$ (parallel attention mechanisms that allow the model to simultaneously focus on different parts of the input sequence, thereby capturing contextual relationships) and the feed-forward network size $ffnS$ (the number of units in the layers that process attention outputs, impacting the learning ability of the model). NeuralLog uses the Bert encoder (Devlin et al. 2018) semantics-based embedding technique to encode log messages from the input logs. NeuralLog showed an overall high detection accuracy when compared with many deep ML techniques (Hashemi and Mäntylä 2021; Xie et al. 2022; Xiao et al. 2024; Yu et al. 2024; Adeba et al. 2024).

2.4.5 Logs2Graphs

Logs2Graphs (Li et al. 2024) is a recent GNN-based semi-supervised deep ML technique that detects log anomalies by modeling the log data as structured graphs, enabling both anomaly detection and interpretability. This technique first organizes the input logs into graph structures where nodes represent unique log events and directed edges capture the sequential relationships between log events. In addition to the hyperparameters used in RNN-based techniques, Logs2Graphs is further characterized by the number of convolutional layers (cL), which controls the network's depth and sets the number of graph convolutional layers; the proximity parameter (k), specifying the order of neighborhood proximity considered within the graph; and the embedding dimensions ($embD$), which determine the size of each node's embedding vector. Logs2Graphs uses Glove (Pennington et al. 2014) embeddings complemented by TF-IDF (Salton and Buckley 1988) to encode input logs into semantic vectors.

3 State of the Art

Le and Zhang (2022) conducted an in-depth analysis of representative semi-supervised (DeepLog Du et al. 2017, LogAnomaly Meng et al. 2019 and PleLog Yang et al. 2021) and supervised (LogRobust Zhang et al. 2019 and CNN Lu et al. 2018) deep learning techniques, in which several model evaluation criteria (i.e., training data selection strategy, log data grouping, early detection ability, imbalanced class distribution, quality of data and early detection ability) were considered to assess the detection accuracy of these different techniques. The study concludes that the detection accuracy of the five deep learning LAD techniques, when taking into account the aforementioned evaluation criteria, is lower than the one reported in the original papers. For instance, the training data strategies significantly impact the detection accuracy of semi-supervised deep learning techniques. Further, data noise such as mislabelled

logs (e.g., logs with errors resulting from the domain expert labelling process) heavily impacts the detection accuracy of supervised deep learning techniques.

Further, depending on the evaluation criteria considered, Le and Zhang (2022)'s study leads to different conclusions when comparing detection accuracy between supervised LAD techniques and semi-supervised ones. Although the semi-supervised techniques DeepLog (Du et al. 2017), LogAnomaly (Meng et al. 2019) and PleLog (Yang et al. 2021) are sensitive to training data strategies, DeepLog and LogAnomaly, in particular, are less sensitive to mislabeled logs than supervised techniques. However, supervised deep learning techniques show better detection accuracy than semi-supervised ones when evaluated on a large amount of data (e.g., log event sequences), in spite of their sensitivity to mislabeled logs.

Although many deep learning techniques for LAD have shown high detection accuracy (e.g., Du et al. 2017; Meng et al. 2019; Zhang et al. 2019; Huang et al. 2020; Le and Zhang 2021), some of them may not perform well, in terms of training time or prediction time, when compared to traditional ML techniques. For instance, NeuralLog (Le and Zhang 2021) and HitAnomaly (Huang et al. 2020) are slower than traditional ML techniques in terms of training and prediction time, respectively. Moreover, traditional ML techniques can be more suitable to detect log anomalies, depending on the application domain and dataset. Further, traditional techniques such as SVM (Cortes and Vapnik 1995) are defined with significantly less hyperparameters than deep learning techniques (e.g., LSTM Hochreiter and Schmidhuber 1997), thus requiring less computational resources for hyperparameter tuning. Therefore, if traditional and deep ML techniques show similar detection accuracy, and if the time performance of traditional ML techniques is significantly better than the one recorded for deep learning techniques, the former are preferable from a practical standpoint. This statement is aligned with the results of a very recent study (Yu et al. 2024) in which the detection accuracy and the time performance of traditional (K-Nearest Neighbor KNN Fix and Hodges 1989, Decision Tree Chen et al. 2004) and deep ML (supervised) techniques (SLFN, CNN Lu et al. 2018, LogRobust Zhang et al. 2019 and NeuralLog Le and Zhang 2021) is compared on five different log-based datasets.

Nevertheless, an ML technique, regardless of its type (either traditional or deep learning), can show i) a high detection accuracy and acceptable time performance, when evaluated on a particular hyperparameter setting, and ii) entirely different results when evaluated on other hyperparameter settings. From the above discussion, we therefore contend that four evaluation criteria should be systemically considered to assess the overall performance of any ML technique, regardless of the type of learning. These criteria are i) detection accuracy, ii) time performance, sensitivity of iii) detection accuracy and iv) time performance w.r.t. different hyperparameter settings.

In Table 1, we list 42 studies that use LAD techniques, including the five ones considered in the aforementioned work (Le and Zhang 2022), and summarize their evaluation strategies. We selected the studies that use LAD techniques that i) are either semi-supervised or supervised deep learning techniques and ii) are most cited and used as alternative techniques in the literature. Column *C.L* indicates, using the symbols Y and N, whether the proposed deep learning LAD technique was compared to at least one traditional ML technique that shares the same model learning type. We also indicate, for each work, whether the evaluation considered: the detection accuracy (column *Acc.*), the time performance (column *Time*), the sensitivity of the detection accuracy to hyperparameter tuning and different datasets (columns *S.H* and *S.D*, respectively, under the *Sensitivity/Acc.* column), as well as the sensitivity of the time performance to hyperparameter tuning across datasets (columns *S.H* and *S.D* under column *Sensitivity/Time*). For each of these criteria, we use symbol + to indicate if the evaluation

Table 1 Comparison of the Evaluation Strategies of Deep Learning Log-based Anomaly Detection Approaches

Study	C.L	Acc.	Time	Sensitivity				Public Datasets							Impl.	Window
				Acc.		Time		HD	HP	OS	HA	BG	TB	SP		
				S.H	S.D	S.H	S.D									
Du et al. (2017) (DeepLog)	N	+	±	±	-	-	-	✓	×	✓	×	×	×	×	Y↑	×
Zhu et al. (2020) (LogNL)	N	+	-	±	-	-	-	✓	×	✓	×	×	×	×	N	×
Xie et al. (2020) (Att-Gru)	N	+	±	-	-	-	-	✓	×	×	×	×	×	×	N	×
Huang et al. (2020) (HitAnomaly)	Y	+	+	±	-	-	-	◇✓	×	✓	×	✓	×	×	N	×
Liu et al. (2021) (LogNads)	Y	+	±	-	-	-	-	◇✓	×	×	×	✓	×	×	N	✓
Meng et al. (2019) (LogAnomaly)	N	+	-	-	±	-	-	✓	×	×	×	✓	×	×	Y↑	×
Yang et al. (2021) (PleLog)	N	+	+	±	-	-	-	✓	×	×	×	✓	×	×	Y↑	×
Zhang et al. (2019) (LogRobust)	Y	+	-	-	-	-	-	◇✓	×	×	×	×	×	×	Y↑	×
Lu et al. (2018) (CNN)	N	+	-	+	-	-	-	✓	×	×	×	×	×	×	Y↑	×
Wang et al. (2022) (LightLog)	N	+	+	-	-	-	-	✓	×	×	×	✓	×	×	Y	×
Le and Zhang (2021) (NeuralLog)	Y	+	+	±	±	-	-	✓	×	×	×	✓	✓	✓	Y↑	×
Guo et al. (2021) (logBert)	Y	+	-	±	-	-	-	✓	×	×	×	✓	✓	×	Y	×
Qi et al. (2023) (LogEncoder)	N	+	-	±	-	-	-	✓	×	×	×	✓	✓	×	N	×
Chen et al. (2022) (EdgeLog)	N	+	±	-	-	-	-	✓	✓	✓	×	✓	×	×	N	×
Qi et al. (2022) (AdAnomaly)	N	+	+	-	+	-	-	✓	×	✓	×	✓	×	×	N	✓
Catillo et al. (2022) (AutoLog)	Y	+	-	±	-	-	-	×	✓	×	×	✓	×	×	Y	×
Zhang et al. (2023) (LayerLog)	Y	+	-	-	-	-	-	✓	×	×	×	✓	×	×	N	×
Almodovar et al. (2023) (LogFit)	N	+	-	-	-	-	-	✓	×	×	×	✓	✓	×	N	×
Xia et al. (2021) (LogGan)	N	+	-	±	±	-	-	✓	×	×	×	✓	×	×	N	×
Hashemi and Mäntylä (2021) (OneLog)	N	+	-	±	±	-	-	✓	✓	×	×	✓	✓	✓	N	×
Du et al. (2021) (LogAttention)	Y	+	-	-	-	-	-	✓	×	×	×	✓	×	×	N	✓
Li et al. (2022) (SwissLog)⊕	Y	+	+	-	-	-	-	◇✓	✓	✓	×	✓	✓	×	Y	×

Table 1 continued

Study																	
Xie et al. (2022) (LogGD)	Y	+	-	-	-	-	✓	×	×	×	×	×	×	×	✓	N	✓
Huang et al. (2023) (HilBert)	Y	+	-	⊞	-	-	◇✓	×	×	×	×	×	×	×	✓	N	×
Han et al. (2021) (InterpretableSAD)	N	+	-	-	-	-	◇✓	×	×	×	×	×	×	×	◇✓	Y	×
Lee et al. (2023) (Hades)	N	+	-	-	-	-	×	×	×	×	×	×	✓	×	×	Y	×
Chen et al. (2021)	N†	+	+	-	-	-	✓	×	×	×	×	×	×	×	✓	N	×
Le and Zhang (2022)	N	+	-	-	-	-	◇✓	×	×	×	×	×	×	×	✓	Y	✓
Wu et al. (2023)	N	+	-	-	-	-	✓	×	×	×	×	×	×	×	✓	Y	✓
Yu et al. (2024) (LightAD)	Y	+	+	-	-	-	✓	×	×	×	×	×	×	×	✓	Y	✓
Li et al. (2024) (Logs2Graphs)	Y	+	⊞	⊞	-	+	✓	✓	×	×	×	×	×	×	✓	Y	×
Xiao et al. (2024) (ContextLog)	Y	+	-	-	-	-	✓	×	×	×	×	×	×	×	◇✓	N	×
Guo et al. (2024) (LogFormer)	Y	+	+	-	-	-	✓	×	×	×	×	×	×	×	✓	Y	×
Zang et al. (2024) (MLAD)	N	+	-	⊞	⊞	-	✓	×	×	×	×	×	×	×	✓	N	×
Yin et al. (2024) (BTCNLog)	N	+	+	-	-	-	×	×	×	×	×	×	×	×	✓	N	✓
Lin et al. (2024) (FastLogAD)	N	+	+	⊞	⊞	-	✓	×	×	×	×	×	×	×	✓	N	×
Gong et al. (2024) (LogETA)	N	+	-	-	-	-	×	×	×	×	×	×	×	×	✓	N	×
Wang et al. (2024) (LogGT)	Y	+	-	⊞	⊞	-	✓	×	×	×	×	×	×	×	✓	N	✓
Landauer et al. (2024)	Y	+	-	-	-	+	✓	✓	×	×	×	×	×	×	✓	Y	×
Yang et al. (2024) (SemPCA)	N	+	+	-	-	-	✓	×	×	×	×	×	×	×	×	Y	✓
Nguyen et al. (2024) (DistilLog)	N	+	+	⊞	-	-	✓	×	×	×	×	×	×	×	×	Y	×
Adeba et al. (2024) (SaRLog)	N	+	-	⊞	-	-	×	×	×	×	×	×	×	×	✓	N	×
Our study	Y	+	+	+	+	+	✓	◇✓	◇✓	◇✓	✓	✓	✓	✓	✓	Y	✓

HD, HP, OS, HA, BG, TB and SP refer to HDFS, Hadoop, OpenStack, Hades, BGL, Thunderbird and Spirit datasets respectively. Dataset HA (Hades) is named after the technique (Lee et al. 2023) (Heterogeneous Anomaly DEtector via Semi-supervised learning), in which the dataset was first used and released. ◇Authors used the first version of the dataset and/or a synthetic version of it. ⊕Not all datasets are used to evaluate the overall performance of SwissLog. For instance, only hdfs is used to assess its time performance, whereas bgl is used to assess the effectiveness of the proposed log parser and the semantic embedding technique used by SwissLog. †The study compares different supervised and unsupervised, traditional and deep ML techniques. However, it does not compare any semi-supervised traditional ML technique to a semi-supervised deep ML technique

criterion is considered for all the techniques used in the experiments; symbol $\pm$ indicates that the evaluation criterion is only considered for the main technique; symbol $-$ indicates that the evaluation criterion is not measured for any of the techniques considered in the paper.

In addition to datasets obtained from industrial contexts (which are not released for confidentiality reasons), LAD techniques have been mostly evaluated on public benchmark datasets (see Section 5.2). In Column *Public Datasets*, we indicate whether or not a public benchmark dataset is used to evaluate the ML techniques in each LAD study, using symbols ✓ and ×, respectively. Moreover, in Column *Impl.*, we indicate whether the implementation of a specific LAD technique is made available in the original paper (using symbols Y and N, respectively). We use the symbol Y↑ in case the implementation of the LAD technique is provided by third parties. Column *Window* indicates whether or not the study assesses the impact of fixed window sizes[2] (using symbols ✓ and × respectively) on the detection accuracy of ML techniques, considering log message-based datasets. The latter represent datasets that are labeled at the level of individual log messages and do not provide any indication about how to regroup the different log messages into sequences.

Comparison among Techniques As shown in Table 1, all empirical studies report the detection accuracy of all the techniques they consider. Only a subset of these studies — focusing on supervised (Huang et al. 2020; Liu et al. 2021; Zhang et al. 2019; Le and Zhang 2021; Du et al. 2021; Li et al. 2022; Xie et al. 2022; Huang et al. 2023; Zhang et al. 2023; Yu et al. 2024; Xiao et al. 2024; Gong et al. 2024; Wang et al. 2024) and semi-supervised (Guo et al. 2021; Catillo et al. 2022; Zhang et al. 2023; Li et al. 2024; Zang et al. 2024; Yin et al. 2024; Lin et al. 2024) approaches[3] — compare, in terms of detection accuracy, the proposed technique with at least one traditional ML technique.

To the best of our knowledge, the most relevant study to our work is an experience report (Chen et al. 2021), which systematically evaluates traditional and deep ML techniques in terms of their anomaly detection accuracy, time performance (in terms of model training and prediction time) and robustness (the ability of an ML technique to detect log anomalies in the presence of unseen log events). However, the study neither assesses the sensitivity of detection accuracy and time performance to hyperparameter tuning of the different ML techniques across datasets nor investigates the impact of window sizes on detection accuracy. Further, it does not study the impact of data imbalance-a common characteristic of real-world log-based datasets (e.g., HDFS, BGL)-on detection accuracy. Additionally, the evaluation of ML techniques in this study is restricted to a very limited number of datasets (HDFS and BGL only), thus affecting its generalizability. In contrast, our work aims to address these limitations by utilizing a broader set of datasets enabling a more comprehensive evaluation of the different ML techniques while systematically evaluating the impact of data imbalance and window size on detection accuracy, time performance, and sensitivity of both detection accuracy and time performance to hyperparameter tuning.

Datasets Most of the LAD techniques (Du et al. 2017; Zhu et al. 2020; Xie et al. 2020; Huang et al. 2020; Liu et al. 2021; Meng et al. 2019; Yang et al. 2021; Zhang et al. 2019; Lu et al. 2018; Wang et al. 2022; Guo et al. 2021; Qi et al. 2023, 2022; Catillo et al. 2022; Almodovar et al. 2023; Zhang et al. 2023; Xia et al. 2021; Xie et al. 2020; Huang et al. 2023; Han et al. 2021; Chen et al. 2021; Xiao et al. 2024; Guo et al. 2024; Zang et al. 2024; Yin et al. 2024; Lin et al. 2024; Gong et al. 2024; Wang et al. 2024; Yang et al. 2024; Nguyen et al. 2024; Adeba et al. 2024) have been evaluated on a small set (two to three datasets

---

[2] Fixed window sizes are windows that are determined by a specific number of log messages on a dataset.

[3] LayerLog (Zhang et al. 2023) adopts two log anomaly detection models, supervised and semi-supervised.

only) of public benchmark datasets, among which HDFS and BGL are the most commonly used ones. Further, even in the case of studies in which LAD techniques are evaluated on a larger set of datasets (Le and Zhang 2021; Chen et al. 2022; Hashemi and Mäntylä 2021; Li et al. 2022; Xie et al. 2022; Le and Zhang 2022; Wu et al. 2023; Yu et al. 2024; Li et al. 2024; Landauer et al. 2024), they either i) do not report the time performance of the different ML techniques or ii) do not study their sensitivity, in terms of detection accuracy or time performance, to hyperparameter tuning across datasets.

Hyperparameter Tuning Hyperparameter tuning is a time and resource-consuming process that can show a gap i) in the computational time (training time and prediction time) ii) the resource allocation (e.g., memory, CPU) of a single ML technique and, when evaluated on different hyperparameter settings. To the best of our knowledge, none of the LAD empirical studies reports the results of the hyperparameter tuning, when applicable. A common practice across these studies consists of reporting only the exact hyperparameter settings that lead to the best results they report in the corresponding research papers.

Impact of Window Size As depicted in Column *Window* of Table 1, only a few studies (Le and Zhang 2022; Qi et al. 2022; Liu et al. 2021; Xie et al. 2022; Du et al. 2021; Wu et al. 2023; Yu et al. 2024; Yin et al. 2024; Wang et al. 2024) assessed the impact of different fixed window sizes on the detection accuracy of ML techniques. More in detail, Table 2 shows the exact window size values that were used in such studies. We also report (using symbols *Y* and *N*) whether these studies assessed the impact of fixed window sizes on the detection accuracy of all the alternative ML techniques (Column *Alt.*) used in their experiments. Only two (Le and Zhang 2022; Xie et al. 2022) out of the nine aforementioned studies assessed the impact of the fixed window size on the detection accuracy of all the alternative techniques.

Motivations for this Work Overall, restricting the evaluation of existing LAD studies to reporting the best results (in terms of the *F1-score*) and sharing the exact hyperparameter settings that led to these results does not help external users (e.g., practitioners or researchers) assess the suitability of a specific ML technique to detect log anomalies in a specific context and datasets w.r.t. its i) overall computational time (model training time and prediction time) and ii) sensitivity to hyperparameter tuning.

Table 2 Existing studies on the impact of fixed window sizes on the detection accuracy of ML techniques

Technique	Datasets			Alt.
	BGL	Thunderbird	Spirit	
Empirical study (Le and Zhang 2022)	[20, 100, 200]	[20, 100, 200]	[20, 100, 200]	Y
LogNads (Liu et al. 2021)	[10, 20, 30, 40]	–	–	N
AdAnomaly (Qi et al. 2022)	[5, 10, 15, 20, 25, 30]	–	–	N
LogGD (Xie et al. 2022)	[20, 60, 100]	[20, 60, 100]	[20, 60, 100]	Y
LogAttention (Du et al. 2021)	[200, 350, 450, 500]	–	–	N
Embedding techniques evaluation (Wu et al. 2023)∗	–	[20, 100, 200]	–	N
LightAD (Yu et al. 2024)	[1, 10]	[1, 10]	[1, 10]	N
BTCNLog (Yin et al. 2024)	[60, 120, 180, 240]	[60, 120, 180, 240]	–	N
LogGT (Wang et al. 2024)	[5, 10, 15, 20, 25, 40]	[5, 10, 15, 20, 25, 40]	–	N

∗ The paper studies the impact of different log message-based grouping strategies from BGL and Spirit datasets on the detection accuracy of different ML techniques, considering different evaluation criteria (e.g., feature aggregation), which fall outside the scope of our paper

Moreover, most studies do not consistently report the execution time of ML techniques; they include either model training time or prediction time. Further, none of these studies provides a systematic evaluation of all the techniques considered in their experimental campaign w.r.t. the four evaluation criteria discussed above.

We therefore believe that conducting large experiments to evaluate ML techniques would be of a great help for practitioners and researchers to better understand what can be expected from different ML techniques and to thus decide what technique(s) they need to apply to address LAD and get the best possible results with the least resources and effort possible.

Given the aforementioned limitations of existing empirical studies, in this paper, we report on the first comprehensive empirical study, in which we not only evaluate the detection accuracy of existing supervised and semi-supervised, traditional and deep learning techniques applied to LAD, but also assess their time performance as well as the sensitivity of their detection accuracy and their time performance to hyperparameter tuning across datasets.

4 Log Representation

To use ML techniques for the detection of execution path log anomalies, sequences of log event occurrences need to be first converted into numerical representations that are understandable by such techniques, while preserving their original meaning (e.g., the different words forming each log event occurrence, the relationship between the different log event occurrences forming these sequences).

A recent study (Wu et al. 2023) has shown that different semantics-based embedding techniques (Word2Vec Mikolov et al. 2013, FastText Joulin et al. 2016 and Bert Devlin et al. 2018), when evaluated on different supervised traditional (e.g., SVM and RF) and deep learning (e.g., CNN, LSTM) techniques on four public benchmark datasets (HDFS, Thunderbird, BGL and Spirit), yield similar results in terms of detection accuracy. In this study, we apply FastText with the traditional (RF, SVM, OC-SVM) and deep (LSTM, LogRobust Zhang et al. 2019) ML techniques since this embedding technique was already used by LogRobust, along with previous LAD studies (Le and Zhang 2022; Yang et al. 2021; Xie et al. 2020) and showed good results. For NeuralLog (Le and Zhang 2021) and Logs2Graphs (Li et al. 2024) techniques, we conducted experiments with the embedding methods (Bert Devlin et al. 2018 and Glove Pennington et al. 2014, respectively) used in the original papers. Regarding Fast-Text, we use the same log encoding technique adopted by LogRobust (Zhang et al. 2019). We first pre-process sequences of log event occurrences (e.g., removing non-character tokens, splitting composite tokens into individual ones). We then apply a three-step encoding technique (i.e., word-vectorization, log event occurrence vectorization, sequence vectorization), which we describe next.

Word Vectorization FastText (Joulin et al. 2016) maps each word w_i, $1 \leq i \leq E$, in the sequence of words $W(\sigma(l)) = (w_1, w_2, \ldots, w_E)$ extracted from the log event occurrence $\sigma(l)$, to a d-dimensional word vector v_i where $1 \leq i \leq E$ and $d = 300$[4]

For instance, let us consider the log event occurrences *battery_filtered_voltage_reading* and *gyroscope_sensor_reading*, recorded in the first two log entries in Fig. 1. The corresponding lists of words are $W(\sigma(1)) = (battery, filtered, voltage, reading)$ and

---

[4] The choice of $d = 300$ dimensions to encode word vectors is motivated by a few LAD techniques (LogRobust Zhang et al. 2019, PleLog Yang et al. 2021, and LightLog Wang et al. 2022) which use the same value for d when evaluated on HDFS (one of the datasets considered in our study). For consistency, we adopted the same dimensionality.

$W(\sigma(2)) = (gyroscope, sensor, reading)$. By setting the word vector dimension to $d = 2$, the different word vectors resulting from FastText and associated to the words $battery$, $filtered$, $voltage$, $reading$, $gyroscope$, and $sensor$ are $v_1 = [-0.2759, -0.0023]$, $v_2 = [0.2618, 0.1413]$, $v_3 = [-0.4211, 0.4043]$, $v_4 = [0.0834, -0.1302]$, $v_5 = [0.3276, 0.4368]$ and $v_6 = [-0.3419, 0.4418]$, respectively.

Log Event Occurrence Vectorization We transform the word list $W(\sigma(l))$ into a word vector list $WV(\sigma(l))$, such that $WV(\sigma(l)) = [v_1, v_2, \ldots, v_E]$, where $v_j \in \mathbb{R}^d$ and $j \in [1, E]$ denotes the word vector. $WV(\sigma(l))$ is finally transformed to an aggregated word vector by aggregating all its word vectors using the weighted aggregation technique TF-IDF (Salton and Buckley 1988), i.e., a technique that measures the importance of the different words defined in a log event occurrence within a log. For instance, the word vector lists associated with the word lists $W(\sigma(1))$ and $W(\sigma(2))$ are $WV(\sigma(1)) = [[-0.3878, -0.0032], [0.3680, 0.1986], [-0.5918, 0.5682], [0.0834, -0.1302]]$ and $WV(\sigma(2)) = [[0.4604, 0.6139], [-0.4805, 0.6209], [0.0834, -0.1302]]$, respectively. The corresponding aggregated word vectors obtained by means of TF-IDF are $[-0.1321, 0.1583]$ and $[0.0211, 0.3682]$, respectively.

Sequence Vectorization Given the aggregated word vectors from the previous step, the latter are further aggregated to form a sequence vector, i.e., a representation of the sequence of log event occurrences. More in detail, the aggregation is done by means of the average operator for each dimension of the aggregated word vectors. For example, if we consider the sequence of log event occurrences obtained from the first two log entries in Fig. 1, given the corresponding aggregated word vectors from the previous step ($[-0.1321, 0.1583]$ and $[0.0211, 0.3682]$), the final sequence vector is $[-0.0555, 0.2633]$.

5 Empirical Study Design

5.1 Research Questions

The goal of our study is to evaluate alternative ML techniques (described in Section 2) when applied to the detection of execution path log anomalies, considering both supervised and semi-supervised, traditional and deep learning techniques. The evaluation is performed based on the four evaluation criteria described in Section 3. We address the following research questions:

- RQ1: How do supervised traditional ML and deep learning techniques compare at detecting execution path log anomalies?
- RQ2: How do supervised traditional ML and deep learning techniques compare in terms of time performance?
- RQ3: How do semi-supervised traditional ML and deep learning techniques compare at detecting execution path log anomalies?
- RQ4: How do semi-supervised traditional ML and deep learning techniques compare in terms of time performance?

These research questions are motivated by the fact that traditional ML techniques are less data hungry and typically less time consuming than deep learning ones when it comes to training the corresponding ML models, and are therefore more practical in many contexts. Therefore, if the loss in detection accuracy is acceptable, assuming there is any, and if the time performance is significantly better than the one recorded for deep ML techniques, traditional

ML techniques are preferable. Similarly, given the scarcity of anomalies in many logs, semi-supervised techniques should be considered in certain contexts. Further, a ML technique, regardless of its type (traditional or deep), when evaluated on the same dataset, can show wide variation in detection accuracy or time performance from one hyperparameter setting to another. This motivates us to study the sensitivity of such accuracy and performance to hyperparameter tuning.

5.2 Benchmark Datasets

All of the LAD studies illustrated in Table 1 have been evaluated on at least one of the seven public labeled benchmark datasets (HDFS, Hadoop, BGL, Thunderbird, Spirit, OpenStack and Hades) listed in Column *Public Datasets*. These benchmark datasets, except for Spirit and Hades, are published in the LogHub dataset collection (He et al. 2020). Most of these datasets are collected from real system executions (HDFS Xu et al. 2009, Hadoop Lin et al. 2016, BGL Oliner and Stearley 2007, Thunderbird Oliner and Stearley 2007, Spirit Oliner and Stearley 2007 and OpenStack Du et al. 2017), whereas one dataset (Hades Lee et al. 2023) is generated from a simulated system. Further, different synthetic versions of the first versions of HDFS, BGL and Thunderbird datasets have been proposed in the context of the empirical evaluation of some of the LAD techniques considered in this study. These versions have been obtained by removing, inserting, or shuffling log events within log event sequences to study the impact of log instability on LAD accuracy. These synthetic datasets are marked with ◇ symbol in Column *Public Datasets* in the table. As seen in Table 1, HDFS and BGL are the most commonly used benchmark datasets across LAD studies. Hades has been only used in one LAD study (Lee et al. 2023) as it has only been released recently.

In this empirical study, we evaluate ML techniques on datasets that are i) suitable for detecting execution path log anomalies (i.e., datasets containing sequences of log messages), ii) labeled, and iii) publicly available. Public benchmark datasets are either labeled at the level of a single log message (BGL, Thunderbird, Spirit, and Hades) or at the level of a session (HDFS, Hadoop, and OpenStack), representing a full system execution. We therefore regroup these datasets into two categories, based on the nature of their original labeling: log message-based or session-based datasets.

Among the seven public benchmark datasets we identified satisfying our requirements, OpenStack is too imbalanced (i.e., anomalies are only injected in four out of 2069 sequences of log event occurrences) and contains a high overlap of 98.5% between normal and anomalous log event sequences (identical sequences) according to findings reported in a recent study (Landauer et al. 2024), and is thus not suitable for our experiments. As an alternative dataset, we used F-dataset (Cotroneo et al. 2019), which was recently reported in the experiments of the Semparser (Huo et al. 2023) technique.

A recent empirical study (Landauer et al. 2024) recommended the ADFA-LD (Australian Defence Force Academy Log Dataset) dataset (Creech and Hu 2013) for evaluating LAD techniques, as its log anomalies are more complex to detect than those in commonly used benchmark datasets (HDFS, Hadoop, BGL and Thunderbird). However, we could not include ADFA-LD in our experiments since only a preprocessed version with numeric identifiers is available, making it unsuitable for our study, where ML techniques (except DeepLog) are fed with semantics encoding of the original log messages (see Section 4). Overall, we evaluated the ML-based LAD techniques on the seven aforementioned datasets.

Since all but one of the datasets are unstructured, we used the Drain (He et al. 2017) log parsing tool to parse them. We chose Drain since it was already used to parse the logs

in the Hades dataset (whose log templates are included in the replication package of the corresponding paper Lee et al. 2023); moreover, Drain has shown to fare much better than other log parsing tools (Khan et al. 2022). We configured Drain with i) the default settings (similarity threshold = 0.5 and tree depth = 4), that are commonly adopted in LAD studies (Le and Zhang 2022; Guo et al. 2021; Li et al. 2022), and ii) the default regular expressions[5].

In the following, we describe in more detail the datasets we used in our empirical study.

5.2.1 Session-based Datasets

The Hadoop Distributed File System (HDFS) dataset was produced from more than 200 nodes of the Amazon EC2 web service. HDFS contains 11175629 log messages collected from 575061 different labeled blocks representing 558223 normal and 16838 anomalous program executions.

The Hadoop dataset contains logs collected from a computing cluster running two MapReduce jobs (WordCount and PageRank). Different types of failures (e.g., machine shut-down, network disconnection, full hard disk) were injected in the logs. The dataset contains 978 executions; 167 logs are normal and the remaining ones (811 logs) are abnormal.

The F-dataset is a synthesized version of the OpenStack dataset that integrates additional failure tests across three subsystems-Cinder, Nova, and Neutron-by injecting 16 distinct types of API error failures. The dataset contains 1640 executions; 1189 are normal and the remaining ones (451 logs) are abnormal.

Table 3 shows the main characteristics of the three session-based datasets used in our experiments. Column *#Temp.* indicates the number of unique templates extracted from the original log messages using the Drain tool. Columns *#N* and *#A* under *#Seq* indicate the total number of normal and anomalous log event sequences, respectively. Column *IR* represents the percentage of log event sequences from the minority class[6]. Columns *Min* and *Max* under *#Len* denote the minimum and the maximum sequence length, respectively. We therefore observe that the three session-based datasets HDFS, Hadoop, and F-dataset are imbalanced, where normal sequences represent the majority class on HDFS and F-dataset and anomalous sequences represent the majority class on Hadoop. Further, HDFS is more imbalanced than both Hadoopand F-dataset. The percentage of log event sequences from the minority class in the former represents 2.93% of the dataset (16838 anomalous sequences out of a total of 575061 sequences), while the percentages in the other datasets are 17.08% for Hadoop (167 normal sequences out of 978 sequences) and 27.5% for F-dataset (1189 normal sequences out of a total of 1640 sequences).

5.2.2 Log message-based Datasets

The BGL dataset contains logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL), California. The dataset contains 4747963 labeled log messages among which 348460 log messages are anomalous (the remaining 4399503 log messages are labeled as normal).

---

[5] We adopted the regular expressions from Logpai (LOGPAI n.d.) for HDFS and Hadoop datasets. The regular expressions used for the BGL and Thunderbird datasets do not cover as many cases as the ones used in LogBert (Guo et al. 2021) (e.g., IP address, hexadecimals, and warnings). We therefore adopted the regular expressions shared by the latter. As the regular expressions for the Spirit dataset are not shared, we adopted the ones used for Thunderbird as both datasets share the same data structure.

[6] Anomalous log event sequences represent the minority class across log message-based datasets on all window sizes, except for Spirit on window size 300 and session-based datasets except Hadoop.

Table 3 Characteristics of session-based Benchmark Datasets

Dataset	# Temp.	#Seq			#Len	
		#N	#A	IR	Min	Max
HDFS	48	558223	16838	2.93%	1	297
Hadoop	340	167	811	17.08%	5	11846
F-dataset	97	1189	451	27.5%	35	1616

The Thunderbird dataset contains logs collected from a supercomputer system at Sandia National Labs (SNL). The dataset contains more than 200000000 log messages labeled by system engineers. In this study, we selected the first ten million[7] log messages from the first version of the Thunderbird dataset. It contains 353794 anomalous log messages while the remaining 9646206 are normal.

The Spirit dataset contains aggregated system logs collected from a super computing system at Sandia National Labs. The dataset contains more than 172000000 labeled log messages. In this study, we selected the first five million[8] log messages from the first version of the dataset. The selected subset contains 4235110 normal log messages while the remaining 764890 log messages are labeled as anomalous.

The Hades dataset contains logs that were obtained by injecting faults on Apache Spark. It is shared by a recent work (Lee et al. 2023) in which a novel semi-supervised ML technique is proposed for large-scale software systems. The dataset consists of 37.64MB of log files collected over a duration of 95.87 hours. The authors share a structured version of the dataset obtained from Drain. Hades contains 1048575 labeled log messages, among which only 575 log messages are anomalous.

Recall that unlike session-based datasets in which sequences are labeled and determined by full executions of a system, log message-based datasets are labeled at the level of individual log messages and do not provide any indication about how to regroup the different log messages into sequences (see Section 2.2). Therefore, a log message grouping (Landauer et al. 2023) step first needs to be applied to such datasets. More in detail, in some studies log messages are grouped using log message-based windows (Huang et al. 2020; Meng et al. 2019; Yang et al. 2021; Liu et al. 2021; Wang et al. 2022; Le and Zhang 2021, 2022) or timestamp-based windows (Qi et al. 2023; Guo et al. 2021; Le and Zhang 2022). Each of these log message-based grouping strategies can be further split into fixed and sliding windows.

5.3 Evaluation Metrics

In the context of (log-based) anomaly detection, we define the standard concepts of *True Positive*, *False Positive*, *True Negative*, and *False Negative* as follows:

- *TP* (True Positive)[9] is the number of the abnormal sequences of log event occurrences that are correctly detected by the model.
- *FP* (False Positive) is the number of normal sequences of log event occurrences that are wrongly identified as anomalies by the model.

---

[7] Most of the studies in the literature (Le and Zhang 2022, 2021; Almodovar et al. 2023; Xie et al. 2022; Yu et al. 2024) used the same subset of Thunderbird for their experiments.

[8] For computation time purpose, we used the same subset from the first version of the Spirit dataset used in the experiments of a recent empirical study (Le and Zhang 2022).

[9] Note that the positive class in our experiments is always associated with the anomalous log event sequences in every dataset, even when this class is not the minority class in a dataset.

- *TN* (True Negative) are normal sequences of log event occurrences that are classified correctly.
- *FN* (False Negative) is the number of abnormal sequences of log event occurrences that are not detected by the model.

In Table 4, we list the evaluation metrics adopted in the existing studies (already introduced in Section 3) to evaluate the corresponding LAD techniques. *Precision* (column *Prec*) indicates the percentage of the *correctly* detected anomalous sequences of log event occurrences over all the anomalous sequences detected by the model; the corresponding formula is $Prec = \frac{TP}{TP+FP}$. *Recall* (column *Rec*) is the percentage of sequences of log event occurrences that are *correctly* identified as anomalous over all real anomalous sequences in the dataset; it is defined as: $Rec = \frac{TP}{TP+FN}$. The *F1-score* (column *F1*) represents the harmonic mean of precision and recall: $F1 = \frac{2*Prec*Rec}{Prec+Rec}$. *Specificity* (column *Spec*) is the percentage of sequences of log event occurrences that are *correctly* identified as normal over all real normal sequences in the dataset; it is defined as: $Spec = \frac{TN}{TN+FP}$. *Accuracy* (column *Acc*) is defined as: $Acc = \frac{TP+TN}{TP+TN+FN+FP}$. *False Positive Rate* (column *FPR*) is defined as: $FPR = \frac{FP}{FP+TN}$. The corresponding formula for the *Area Under Curve* (column *AUC*) is : $AUC = \frac{Rec+(1-FPR)}{2}$.

We indicate whether or not an evaluation metric is used to evaluate the ML techniques in each LAD study, using symbols ✓ and ×, respectively. As shown in Table 4, most of the studies (41 out of 42) evaluated the different LAD techniques by means of *Prec*, *Rec* and *F1*. This is because most of the log-based datasets (see Section 5.2) are highly imbalanced, with normal log event sequences representing the majority class. This imbalance makes evaluation metrics such as the *F1-score*, which prioritize the accurate detection of the minority (anomalous log event sequences) class, particularly valuable for assessing the detection accuracy of log anomalies.

In contrast, evaluation metrics such as accuracy (*Acc*) can be misleading in such contexts, as they are skewed by the majority class and, therefore, unreliable for evaluating LAD techniques (Yao and Shepperd 2021). Similarly, while AUC measures the ability of a model to distinguish between normal and anomalous log event sequences across various thresholds, it does not provide detailed insights into precision or false positive rates-key factors in imbalanced scenarios where the majority class heavily influences the detection accuracy (Hancock et al. 2023). Further, *FPR*, which quantifies the proportion of normal log event sequences incorrectly classified as anomalous can be problematic in the context of imbalanced log-based datasets. This is because the false positives become obfuscated by the large number of normal log event sequences (the negative class). Since the denominator in the definition of FPR is the size of the negative class (the total number of FP and TN), which is considerably larger in such datasets, even notable changes in the number of false positives may appear negligible. This limitation makes FPR an unsuitable evaluation metric for effectively evaluating LAD techniques in scenarios where minimizing false alarms is critical (Hancock et al. 2023). For these reasons, we adopt *Prec*, *Rec* and *F1* to assess the detection accuracy of the different ML techniques considered in our study.

Further, although specificity is not commonly reported in the literature (it was used in only four studies), we select this evaluation metric because i) it is relevant for assessing the ability of ML models to recognize normal log event sequences (the majority class in most benchmark datasets) and ii) its usage was strongly recommended in a recent empirical study (Le and Zhang 2022), in which deep ML techniques show a low specificity (below 0.5), revealing that the corresponding models perform poorly by classifying many normal log event sequences as anomalies, causing many false alarms.

Table 4 Evaluation metrics considered in existing studies

Study	Prec	Rec	F1	Acc	Spec	FPR	AUC
Du et al. (2017) (DeepLog)	✓	✓	✓	✗	✗	✗	✗
Zhu et al. (2020) (LogNL)	✓	✓	✓	✗	✗	✗	✗
Xie et al. (2020) (Att-Gru)	✓	✓	✓	✓	✗	✗	✗
Huang et al. (2020) (HitAnomaly)	✓	✓	✓	✗	✗	✗	✗
Liu et al. (2021) (LogNads)	✓	✓	✓	✓	✗	✓	✓
Meng et al. (2019) (LogAnomaly)	✓	✓	✓	✗	✗	✗	✗
Yang et al. (2021) (PleLog)	✓	✓	✓	✗	✗	✗	✗
Zhang et al. (2019) (LogRobust)	✓	✓	✓	✗	✗	✗	✗
Lu et al. (2018)	✓	✓	✓	✗	✗	✗	✗
Wang et al. (2022) (LightLog)	✓	✓	✓	✗	✗	✗	✗
Le and Zhang (2021) (NeuralLog)	✓	✓	✓	✗	✗	✗	✗
Guo et al. (2021) (logBert)	✓	✓	✓	✗	✗	✗	✗
Qi et al. (2023) (LogEncoder)	✓	✓	✓	✗	✗	✗	✗
Chen et al. (2022) (EdgeLog)	✓	✓	✓	✗	✗	✗	✗
Qi et al. (2022) (AdAnomaly)	✓	✓	✓	✗	✗	✗	✗
Catillo et al. (2022) (AutoLog)	✓	✓	✓	✗	✗	✗	✗
Zhang et al. (2023) (LayerLog)	✓	✓	✓	✗	✗	✗	✗
Almodovar et al. (2023) (LogFit)	✓	✓	✓	✗	✓	✗	✗
Xia et al. (2021) (LogGan)	✓	✓	✓	✗	✗	✗	✗
Hashemi and Mäntylä (2021) (OneLog)	✓	✓	✓	✗	✗	✗	✗
Du et al. (2021) (LogAttention)	✓	✓	✓	✗	✗	✗	✗
Li et al. (2022) (SwissLog)	✓	✓	✓	✗	✗	✗	✗
Xie et al. (2022) (LogGD)	✓	✓	✓	✗	✗	✗	✗
Huang et al. (2023) (HilBert)	✓	✓	✓	✗	✗	✗	✗
Han et al. (2021) (InterpretableSAD)	✓	✓	✓	✗	✗	✗	✗
Lee et al. (2023) (Hades)	✓	✓	✓	✗	✗	✗	✗
Chen et al. (2021)	✓	✓	✓	✗	✗	✗	✗
Le and Zhang (2022)	✓	✓	✓	✗	✓	✗	✗
Wu et al. (2023)	✓	✓	✓	✗	✗	✗	✗
Yu et al. (2024)	✓	✓	✓	✗	✗	✗	✗
Li et al. (2024) (Logs2Graphs)	✓	✗	✗	✗	✗	✗	✓
Xiao et al. (2024)(ContexLog)	✓	✓	✓	✗	✗	✗	✗
Guo et al. (2024)(LogFormer)	✓	✓	✓	✗	✗	✗	✗
Zang et al. (2024) (MLAD)	✓	✓	✓	✗	✗	✗	✗
Yin et al. (2024) (BTCNLog)	✓	✓	✓	✗	✓	✗	✗
Lin et al. (2024) (FastLogAD)	✓	✓	✓	✗	✗	✗	✗
Gong et al. (2024) (LogETA)	✓	✓	✓	✗	✗	✗	✓
Wang et al. (2024) (LogGT)	✓	✓	✓	✗	✗	✗	✓
Landauer et al. (2024)	✓	✓	✓	✗	✓	✗	✗
Yang et al. (2024) (SemPCA)	✓	✓	✓	✗	✗	✗	✗
Nguyen et al. (2024) (DistilLog)	✓	✓	✓	✗	✗	✗	✗
Adeba et al. (2024) (SaRLog)	✓	✓	✓	✗	✗	✗	✗

5.4 Experimental Setup

In this empirical study, as discussed in Section 2, we consider nine alternative ML techniques. Three of them are traditional: SVM, RF (supervised) and OC-SVM (semi-supervised); see Section 2.3. The others are deep learning-based: LogRobust (Zhang et al. 2019), LSTM (Hochreiter and Schmidhuber 1997), NeuralLog (Le and Zhang 2021)[10] (supervised), DeepLog (Du et al. 2017,) and Logs2Graphs (Li et al. 2024) (semi-supervised); see Section 2.4.

5.4.1 Hyperparameter Settings

Each of the nine alternative ML techniques considered in our study requires hyperparameter tuning before models can be trained. In the following, we provide the hyperparameter settings associated with each of the techniques considered in this study.

- **SVM.** We used the RBF kernel function, set the values of C to {1, 10, 100, 1000} and γ to {0.0001, 0.001, 0.01, 0.1}. These values of γ and C are within the range of values that were recommended in a study (Probst et al. 2019) in which hyperparameter tuning was conducted to assess the impact of different hyperparameter settings on the detection accuracy of SVM on 38 datasets. Setting the hyperparameters of SVM to the above values leads to 16 different hyperparameter settings (i.e., combinations of hyperparameter values).

- **RF.** We set the number of decision trees dTr to values ranging from 10 to 100 in steps of 10 based on the findings reported in the past studies (Oshiro et al. 2012; Probst and Boulesteix 2018) which thoroughly investigated the impact of the number of decision trees on the detection accuracy of RF using a large number of datasets. The findings suggest that RF can achieve the highest detection accuracy using 100 trees. Additionally, considering that computational time (training and prediction time) increases linearly with the number of trees (Probst et al. 2019), we aimed to strike a balance between the detection accuracy and the computational time. Consequently, we opted for dTr values ranging from 10 to 100 in steps of 10. We set the number of features $sFeat$ in a single node of each decision tree to the square root[11] of the total number of features (i.e., the total features represent the $d = 300$ dimensions of the encoded sequence of log event occurrences as defined in Section 4), leading to 10 hyperparameter settings.

- **OC-SVM.** We used the RBF kernel function and set the values of ν from 0.1 to 0.9 in steps of 0.1. The selection of ν values aligns with the recommendations from a previous study (Yu and Kang 2019) in which they studied the impact of ν hyperparameter on the performance of OC-SVM, considering different values of ν ranging within the interval bounded by 0.02 and 1 on ten benchmark datasets. For γ hyperparameter, — similarly to the SVM settings — we selected values in {0.0001, 0.001, 0.01, 0.1}, leading to 36 different hyperparameter settings.

---

[10] NeuralLog was intentionally designed without incorporating any template extraction technique (see Section 2.4). We therefore consider two versions of this ML technique in our experiments: one version (called "NeuralLog1") that we trained using raw log messages from all the seven datasets (see Section 5.2) and another version (called "NeuralLog2"), where the ML model is fed with the templates extracted by means of Drain; such templates are also fed to the remaining ML techniques for consistency (see Section 5.5).

[11] The square root of the total number of features is a common practice when applying RF (Bernard et al. 2009; Genuer et al. 2008).

- **LSTM, LogRobust, DeepLog, NeuralLog, and Logs2Graphs.** To train these deep learning-based techniques[12], we set the loss function lF to the binary cross entropy, the optimizer opt to the three commonly used optimizers (`adam`, `rmsprop`, and `adadelta`). According to Perin and Picek (2021), `adam` and `rmsprop` are more suitable on small neural networks (e.g., a small number of hidden layers and a small number of neurons), whereas `adadelta` is more suitable for larger neural networks. Further, another study (Okewu et al. 2019) suggests that the three selected optimizers (`adadelta`, `adam` and `rmsprop`) lead to a high detection accuracy of deep learning ML techniques based on CNN. We therefore selected these three optimizers to conduct our experiments. We set the batch size bS to three different values (32, 64 and 128) specifically in multiples of 32. We remark that a batch size of 32 was recommended as a default value by Bengio (2012). We also set the number of hidden layers hL to 2 and the number of epochs[13] epN to {10, 50, 100, 150}, leading to 36 different hyperparameter settings for each of these techniques. As LogRobust is defined with an additional hyperparameter $nEpStop$, we set the latter to 10, as adopted by a previous empirical study (Le and Zhang 2022). For the hyperparameters that are restricted to the definition of transformer-based ($attH$ and $ffnS$) and GNN-based models (cL, k and $embD$) and do not apply to RNN-based models, we set the corresponding values to the ones used in the original papers (see Section 2.4).

5.5 Experimental Methodology

In this section, we discuss the experimental methodology we adopted to answer the four research questions. More in detail, we first present the grouping strategy we follow to group log messages in log message-based datasets. We then explain how we perform the hyperparameter tuning and evaluate the different ML techniques across session-based and log message-based datasets.

5.5.1 Log message-based Grouping Strategy

Due to the inconsistent use of fixed window sizes across studies and the lack of coverage of all alternative techniques and common benchmark datasets (BGL, Thunderbird and Spirit) in existing studies, we assess the impact of the size of fixed log message-based windows on the detection accuracy of the traditional and deep, supervised and semi-supervised ML techniques, considering nine window sizes (ws) ranging from 10 to 300.

Table 5 describes the characteristics of the four log message-based datasets based on the nine window sizes we considered in our study (Column *Window size*). For each dataset, we indicate: the number of unique templates extracted from the original log messages (Column *#Temp.*); the total number of normal and anomalous sequences (Column *#N* and Column *#A* under *Seq.*, respectively); the percentage of log event sequences from the minority class (Column *IR* under *Seq.*) computed for each window size in the different log message-based datasets. As shown in Table 5, log message-based datasets become less imbalanced with the increase of window size. In other words, the percentage of log event sequences from the minority class (Column *IR*) increases from small to large window sizes, across datasets. For

---

[12] For DeepLog, we set the top log event candidates, i.e., log events that are likely to occur given a history of previously seen log events, to 9.

[13] Due to the high computational cost of the experiments conducted in the paper, we set the maximum number of epochs for all the deep learning techniques to 150.

Table 5 Characteristics of log message-based Benchmark Datasets

Data.	#Temp.	Seq.	Window Size								
			10	15	20	50	100	150	200	250	300
Hades	117	#N	104718	69776	52314	20887	10410	6921	5184	4132	3439
		#A	139	128	114	84	75	69	58	62	56
		IR	0.13%	0.18%	0.22%	0.40%	0.72%	0.99%	1.11%	1.48%	1.60%
BGL	1425	#N	432326	287671	215418	85465	42310	28004	20885	16621	13775
		#A	39023	26561	20256	8804	4824	3419	2682	2232	1936
		IR	8.28%	8.45%	8.59%	9.34%	10.23%	10.88%	11.38%	11.84%	12.32%
Thunderbird	4265	#N	832313	525125	377278	129342	61737	40501	30096	23925	19829
		#A	167686	141541	122721	70657	38262	26165	19903	16074	13504
		IR	16.77%	21.23%	24.54%	35.33%	38.26%	39.25%	39.81%	40.19%	40.51%
Spirit	15487	#N	353642	230075	169976	64611	30270	18987	13399	10153	8131
		#A	146356	103257	80023	35388	19729	14346	11600	9846	8535
		IR	29.27%	30.98%	32.01%	35.39%	39.46%	43.04%	46.40%	49.23%	48.79%

Table 6 Set up of benchmark datasets

Learning	Training	Validation	Test
Semi-supervised	70% C_1	10% C_1	20% C_1
		10% C_2	90% C_2
Supervised	70% C_1	10% C_1	20% C_1
	70% C_2	10% C_2	20% C_2

C_1 (C_2) is the majority (minority) class in each dataset

instance, we observe that Hades is the most imbalanced dataset, in which the percentage of log event sequences from the minority class varies between 0.13% on $ws = 10$ and 1.60% on $ws = 300$. Spirit is one of the two less imbalanced log message-based datasets. The percentage of log event sequences from the minority class ranges between 29.27% and 49.23% on $ws = 10$ and $ws = 250$, respectively.

5.5.2 Hyperparameter Tuning Phase

Table 6 summarizes the strategy we followed to divide the benchmark datasets so as to enable training. Symbol C_1 denotes the majority class in each dataset, whereas C_2 denotes the minority class[14]. Column *Learning* indicates the learning type, semi-supervised or supervised. We divided each dataset used in our experiments into training, validation[15], and testing sets and assigned different proportions for these sets depending on the learning type of each technique as follows:

- *Semi-supervised.* Models are trained on 70% of the majority class, validated on 10% of each class and tested on the remaining set (20% C_1 and 90% C_2).
- *Supervised.* Models are trained on 70% of each class, validated on 10% of each class, and tested on the remaining set (20% C_1 and 20% C_2).

It is typically challenging to specify what hyperparameter values to use for a specific ML technique, on a particular dataset. Therefore, for each learning algorithm, we carried out hyperparameter tuning, using a grid search (Bergstra and Bengio 2012), which is one of the commonly used strategies.

To perform our experiments and answer all the research questions (see Section 5.1), we first trained the different ML techniques with features extracted from the seven benchmark datasets used in this study (see Section 5.2). We then test the different ML models on these datasets, considering different combinations of hyperparameter settings per technique (see Section 5.4.1). At the end of this step, we collected i) the different *F1-score* values and ii) the different *training* time values from both supervised and semi-supervised techniques to study their sensitivity to hyperparameter tuning.

For each hyperparameter setting, we trained the ML technique on the training set and validated it on the validation set. To avoid biased results and assess the stability of the detection accuracy of each technique, we repeated this process (training and validation) five times, computed *precision*, *recall*, *F1-score*, and *Specificity*, and recorded the computational

---

[14] Recall that the majority class is "normal" for all the datasets, except Spirit on the largest window size ($ws = 300$) and Hadoop.

[15] The validation set is used to determine the hyperparameter settings that lead to the highest detection accuracy for semi-supervised and supervised ML techniques by testing the model on unseen data.

time needed for the training phase (training time and validation time) for each iteration; we reported the average values from the five iterations.

Given that there are 16, 10 and 36 hyperparameter settings for the traditional techniques considered in this study (respectively, SVM, RF and OC-SVM) and 36 hyperparameter settings for each of the five deep learning techniques (LSTM, DeepLog, LogRobust, both versions of NeuralLog, and Logs2Graphs), the total number of hyperparameter settings considered in this study during hyperparameter tuning is 278. Concurrently executing[16] each algorithm five times for all the 278 hyperparameter settings i) on three session-based datasets leads to $5 \times 278 \times 3 = 4170$ executions and ii) on four log message-based datasets with nine different window sizes leads to $5 \times 278 \times 4 \times 9 = 50040$ executions. The total number of executions is therefore set to 4170 + 50040 = 54210, leading to 1933 days (≈ 5.30 years) of computation time.

We collected the average *F1-score* for each hyperparameter setting of a ML technique, across datasets, to analyze its sensitivity in terms of detection accuracy. Similarly, we collected the average computational time needed for the training phase for each hyperparameter setting to assess the time performance sensitivity of each technique, considering each dataset separately.

Best Hyperparameter Settings Table 7 shows the hyperparameter settings that led to the highest detection accuracy on the validation set for each ML technique, on each benchmark dataset. Recall that unlike session-based datasets (HDFS, Hadoop, and F-dataset), log message-based datasets (Hades, BGL, Thunderbird and Spirit) are labeled at the level of individual log messages. After extracting log events from the raw log messages, we generated sequences of log event occurrences from such datasets using nine fixed window sizes (see Section 5.5.1). We therefore evaluated each ML technique on all window sizes and reported the results associated with the one which yields the highest detection accuracy in terms of *F1-score*.

5.5.3 Testing Phase

We selected the best hyperparameter setting for each ML technique on each dataset obtained from the previous step to i) re-train the different ML models on the training and validation sets and ii) evaluate them on the test set. We repeated the process five times for each ML technique, on each dataset, and then computed *precision*, *recall*, and *F1-score*, as well as *re-train time* and *test time* per iteration[17]. We finally computed and reported the average *F1-score* and the average *re-train time* from the five iterations associated with the best hyperparameter setting for each ML technique, evaluated on each dataset, to reflect the best possible detection accuracy and time performance of that technique. More in details, each of the nine ML techniques was concurrently executed five times for the best hyperparameter setting on each of i) the three session-based datasets (HDFS, Hadoop and F-dataset), leading to $9 \times 5 \times 3 = 135$ executions and ii) the four log message-based datasets (BGL, Thunderbird, Spirit and Hades) with nine window sizes leading to $9 \times 5 \times 4 \times 9 = 1620$ executions. The total number of

---

[16] All the experiments were conducted on cloud computing platforms provided by the Digital Research Alliance of Canada (DRAC n.d.): a) the Narval cluster with a total of 636 NVIDIA A100 GPUs with 8 to 40 GB of memory, b) the Cedar cluster with a total of 1352 NVIDIA P100 Pascal GPUs with 8 to 64 GB of memory, c) the Beluga cluster with a total of 688 NVIDIA NVidia V100SXM2 GPUs with 4 to 16 GB of memory, and d) the Graham cluster with a total of 520 NVIDIA (P100 Pascal, V100 Volta and T4 Turing) GPUs with 8 to 32 GB of memory.

[17] We computed the time performance metrics (model training, re-training and prediction time) by means of Python `time` function (Python Software Foundation n.d.).

Table 7 Best Hyperparameter Settings

Technique	Hyper.	Dataset						
		HDFS	Hadoop	F-dataset	Hades	BGL	Thunderbird	Spirit
SVM	C	1	1000	1000	1000	1000	10	1000
	γ	0.1	0.0001	0.1	0.001	0.001	0.1	0.01
RF	dTr	60	50	100	80	80	80	100
LSTM	opt	adam	adam	adam	adam	rmsprop	adam	adam
	epN	150	10	150	100	10	150	100
	bS	64	32	32	64	128	32	32
LogRobust	opt	rmsprop	rmsprop	rmsprop	rmsprop	adam	rmsprop	adam
	epN	100	150	150	100	100	10	100
	bS	64	128	128	32	128	64	64
NeuralLog1	opt	adadelta	rmsprop	rmsprop	adam	adam	rmsprop	adam
	epN	150	150	150	150	150	150	100
	bS	32	128	128	128	128	32	128
NeuralLog2	opt	rmsprop	rmsprop	rmsprop	adam	rmsprop	rmsprop	adam
	epN	50	150	150	100	100	100	50
	bS	128	128	128	64	128	64	128
OC-SVM	ν	0.2	0.1	0.3	0.1	0.1	0.9	0.4
	γ	0.0001	0.01	0.0001	0.1	0.1	0.0001	0.1
DeepLog	opt	rmsprop	rmsprop	rmsprop	adam	adam	rmsprop	adam
	epN	10	150	100	50	100	150	150
	bS	64	32	32	32	64	32	64
Logs2Graphs	opt	rmsprop	rmsprop	adam	adadelta	adadelta	rmsprop	adadelta
	epN	100	150	150	50	50	10	150
	bS	32	128	128	64	32	32	128

executions during the testing phase is therefore set to $135 + 1620 = 1755$ leading to 96 days ($\approx$ three months) of computation time.

We remark that research questions RQ1 and RQ2 are both dedicated to supervised ML techniques, whereas RQ3 and RQ4 concern semi-supervised ones. We therefore used the same hyperparameter settings for RQ1 and RQ2. Similarly, RQ3 and RQ4 share the same settings.

5.5.4 Statistical Analysis of the Results

To assess the significance of the difference among the semi-supervised and supervised, traditional and deep ML techniques used in this study, we applied the non-parametric statistical Kruskal-Wallis test (Kruskal 1952) on the results obtained from answering our research questions. We selected the Kruskal-Wallis test because it is i) suitable for non-normally distributed data and ii) commonly used to evaluate the performance of ML techniques on multiple datasets. This test was chosen as it does not require assumptions about the underlying data distribution, making it particularly well-suited for comparing multiple independent groups, especially when dealing with datasets of varying sizes and distributions.

More in detail, we conducted five statistical tests, each associated with one of the evaluation criteria: a) detection accuracy, b) sensitivity of the detection accuracy to hyperparameter tuning, c1) time performance - re-training time; c2) time performance - prediction time, and d) the sensitivity of the time performance (training time) to hyperparameter tuning. We provided as input (score) to these tests i) the highest F1-score for each ML technique on each dataset; ii) the range of F1-score (i.e., the difference between the minimum and maximum F1-score) across hyperparameter settings from the sensitivity analysis; iii) the model re-training time and iv) the prediction time (both associated with the best F1-score reported for each ML technique on each dataset); v) the range of the training time (i.e., the difference between the minimum and maximum training time) across hyperparameter settings from the sensitivity analysis. More in detail, we performed the five statistical tests on the detection accuracy (in terms of *F1-score*) and the time performance of nine alternative ML techniques (see Sections 2.3 and 2.4) across seven datasets (see Section 5.2), leading to a sample size of $9 \times 7 = 63$. For each of the five statistical tests, we set the null hypothesis to: "There is no significant difference among ML techniques across datasets". We considered a confidence level of 95%, setting the significance level value to 0.05. We then calculated the test statistic and the corresponding *p-value*. We rejected the null hypothesis when the *p-value* was below that selected significance level (p-value < 0.05).

Further, we conducted a post-hoc analysis on the results associated with each of the evaluation criteria in which the null hypothesis was rejected. To do so, we applied the non-parametric pairwise post-hoc statistical Dunn's test (Dunn 1964) to compare all the different pairs of ML techniques in terms of the sensitivity of the *F1-score* to hyperparameter tuning.

6 Results

6.1 RQ1 - Detection accuracy of supervised traditional and deep ML techniques

6.1.1 Detection Accuracy

As shown in Table 9, both supervised traditional (SVM, RF) and deep (LSTM, LogRobust, NeuralLog1 and NeuralLog2) ML techniques show a high detection accuracy (*F1-score*)

Table 8 Window sizes associated with the highest detection accuracy for supervised and semi-supervised, traditional and deep ML techniques

Learning Type	Technique	Log message-based dataset			
		Hades	BGL	Thunderbird	Spirit
Supervised	SVM	15	15	50	20
	RF	10	15	10	15
	LSTM	15	15	10	20
	LogRobust	10	10	20	10
	NeuralLog1	10	15	15	10
	NeuralLog2	10	15	15	50
Semi-supervised	OC-SVM	300	50	250	10
	DeepLog	10	10	200	15
	Logs2Graphs	10	200	15	10

Table 9 Comparison of the detection accuracy of supervised traditional and deep ML techniques on all datasets

Dataset		Metric	Technique					
			SVM	RF	LSTM	LogRobust	NeuralLog1	NeuralLog2
Session	HDFS	*Prec*	99.20	99.64	98.41	100.00	99.32	99.64
		Rec	99.91	99.91	98.69	99.48	97.34	92.56
		F1	99.56	99.78	98.53	99.74	98.31	95.60
		Spec	99.98	99.99	99.95	100.00	100.00	100.00
	Hadoop	*Prec*	82.74	82.87	82.74	82.74	83.33	83.33
		Rec	100.00	97.91	100.00	100.00	100.00	100.00
		F1	90.56	89.76	90.56	90.56	90.91	90.91
		Spec	0.00	2.94	0.00	0.00	0.00	0.00
	F-dataset	*Prec*	98.68	93.97	100.00	0.00	0.00	0.00
		Rec	82.42	99.34	61.76	0.00	0.00	0.00
		F1	89.82	96.58	76.25	0.00	0.00	0.00
		Spec	99.58	97.56	100.00	99.50	100.00	100.00
Log message	Hades	*Prec*	100.00	100.00	95.05	95.00	100.00	60.00
		Rec	88.46	57.14	73.85	53.57	73.33	42.86
		F1	93.88	72.73	83.11	68.50	84.00	49.26
		Spec	100.00	100.00	99.99	100.00	100.00	100.00
	BGL	*Prec*	97.52	93.29	97.49	99.79	99.97	99.66
		Rec	92.38	78.96	84.27	95.50	99.91	98.76
		F1	94.88	85.46	90.39	97.59	99.94	99.21
		Spec	99.78	99.46	99.80	99.98	100.00	99.97
	Thunderbird	*Prec*	99.99	99.49	99.38	99.98	99.98	99.99
		Rec	98.22	97.01	98.55	99.84	99.97	99.96
		F1	99.10	98.24	98.96	99.91	99.98	99.97
		Spec	99.99	99.90	99.88	99.99	100.00	100.00
	Spirit	*Prec*	97.87	98.60	97.98	100.00	99.98	99.96
		Rec	97.53	85.33	93.31	95.76	99.97	99.77
		F1	97.70	91.49	95.57	97.83	99.98	99.86
		Spec	99.00	99.46	99.08	100.00	100.00	99.98

when evaluated on the session-based datasets HDFS and Hadoop, with better results on HDFS than Hadoop. On F-dataset, traditional ML techniques by far outperform deep ML techniques with the highest *F1-score* of 96.58 achieved by RF and the lowest *F1-score* (0.00) recorded for LogRobust and both versions of NeuralLog (NeuralLog1 and NeuralLog2). The low detection accuracy of the latter techniques is likely due to the small number of anomalous log event sequences in F-dataset relative to the other datasets, which makes it difficult for complex models like the attention-based RNN model (LogRobust) and the Transformer-based model (NeuralLog) to effectively learn the minority class features. We remark that the specificity of all supervised ML techniques is high and similar across session-based datasets except for Hadoop, due to the fact that the majority class of this dataset corresponds to anomalous log event sequences, making it challenging for the different supervised ML techniques to recognize the normal log event sequences.

When evaluated on three log message-based datasets (BGL, Thunderbird, and Spirit), supervised ML techniques yield a high detection accuracy (in terms of *F1-score*), with slightly better results on Thunderbird and Spirit than BGL. Except for SVM, the remaining supervised ML techniques show a large decrease in *F1-score* when evaluated on Hades. More specifically, while SVM achieves an *F1-score* of 93.88, the *F1-score* of the remaining techniques ranges from 49.26 for NeuralLog2 to 84.00 for NeuralLog1, with 72.73 for RF. The higher *F1-score* of SVM on the most imbalanced dataset Hades, compared to that of the remaining supervised ML techniques, can be attributed to hyperparameters C, which penalizes the misclassification of the minority class (anomalous log event sequences), and γ, which makes the decision boundary more flexible to effectively differentiate between normal and anomalous log event sequences.

We remark that all supervised, traditional and deep ML techniques show very high and similar detection accuracy on HDFS, Hadoop, BGL and Thunderbird. This is due to the nature of the datasets, where a recent study (Landauer et al. 2024) shows that simple, non-ML detection techniques, like counting sequence lengths, can also effectively detect log anomalies and achieve high accuracy. This is because log anomalies typically manifest themselves through new log event types, variations in log event frequencies and, to a lesser extent, changes in sequence lengths. Overall, the study suggests that a majority of the anomalies are straightforward to identify and the relation between log anomalies and sequential patterns is less pronounced than expected within these commonly used benchmark datasets.

The specificity (*Spec*) of all supervised traditional and deep ML techniques is high on all log message-based datasets, as all these datasets contain a high number of normal log event sequences. We also observe that NeuralLog1 outperforms NeuralLog2 in terms of *F1-score* and *Spec* on all datasets. This is expected given that NeuralLog (Le and Zhang 2021) is designed to detect log anomalies directly from raw logs rather than from log templates extracted by means of a log parsing technique.

Figure 3 shows the impact of different window sizes on the detection accuracy of supervised traditional and deep ML techniques on log message-based datasets.

- *Small window sizes* {10, 15, 20}. As depicted in Table 8, supervised ML techniques yielded their highest detection accuracy (in terms of *F1-score*) on smaller window sizes across the log message-based datasets. For instance, on Hades, RF and LogRobust obtained their highest detection accuracy with $ws = 10$. This may be expected given that small window sizes lead to more sequences to train supervised ML models.
- *Large window sizes* {50, 100, 150, 200, 250, 300}. All the supervised ML techniques showed a decrease in detection accuracy when evaluated on large window sizes across all the log message-based datasets. The overall decrease in detection accuracy is higher on
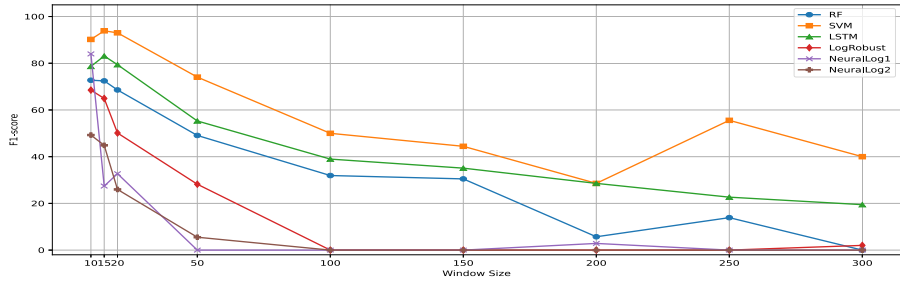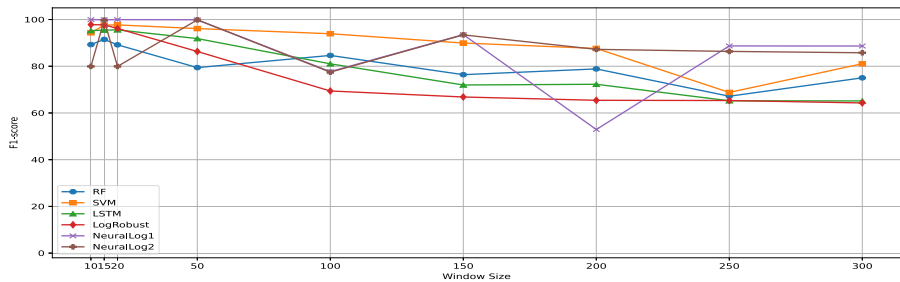
(a) Impact of window size when using the Hades dataset
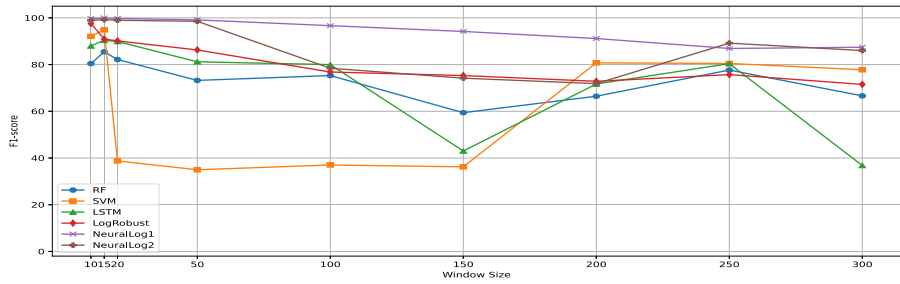
(b) Impact of window size when using the BGL dataset
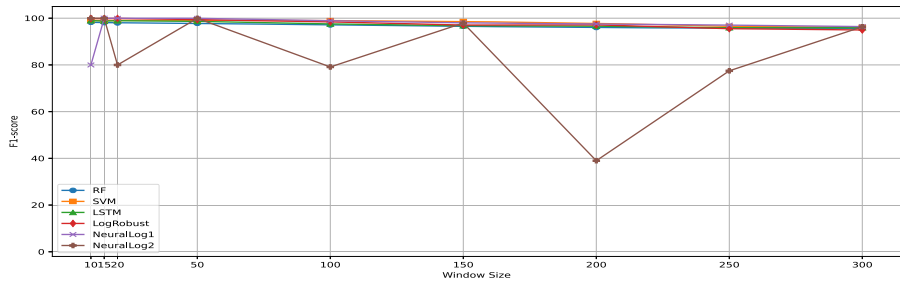
(c) Impact of window size when using the Thunderbird dataset

(d) Impact of window size when using the Spirit dataset

Fig. 3 Impact of window size on the detection accuracy of supervised traditional and deep ML techniques on log message-based datasets

more imbalanced datasets (Hades, BGL) than on less imbalanced datasets (Thunderbird, Spirit). For instance, on Hades, RF yielded an *F1-score* that decreased from 72.73 with $ws = 10$ to 0.0 with $ws = 300$; on Thunderbird, RF shows a detection accuracy ranging from 98.24 with $ws = 10$ to 95.67 with $ws = 300$. This confirms that larger window sizes often lead to lower detection accuracy (especially on highly imbalanced datasets), indicating potential challenges for the supervised ML techniques in capturing log patterns effectively.

Statistical analysis (see § 5.5.4) yields a *p-value* of 0.88, suggesting the detection accuracy of the different supervised traditional and deep ML techniques is not significantly different. Therefore accuracy is not a distinguishing factor among techniques on these datasets.

6.1.2 Sensitivity of Detection Accuracy

As depicted in Fig. 4, the overall sensitivity of RF on the three session-based datasets HDFS, Hadoop and F-dataset (plots in Fig. 4a, b and c, respectively) is far lower than that of the remaining traditional (SVM) and deep (LSTM, LogRobust, NeuralLog1 and NeuralLog2) ML techniques. More in detail, RF shows a detection accuracy ranging from i) 99.27 to 99.33

(a) HDFS dataset　　　　　　　　　　　　　　　　　(b) Hadoop dataset
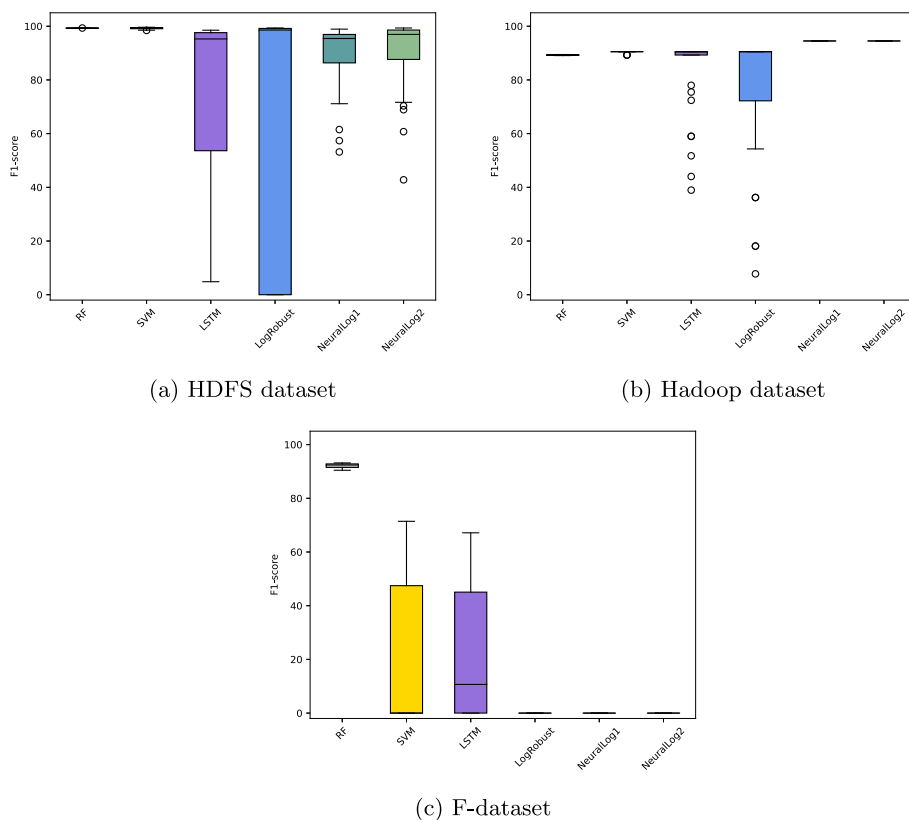
(c) F-dataset

Fig. 4 Sensitivity of the detection accuracy of supervised traditional and deep ML techniques on session-based datasets

(avg $\approx$ 99.29, stdDev $\approx$ 0.02) on HDFS, ii) 89.07 to 89.47 (avg $\approx$ 89.28, stdDev $\approx$ 0.13) on Hadoop and iii) 90.44 to 93.18 (avg $\approx$ 92.15, stdDev $\approx$ 0.83) on F-dataset.

In contrast, on HDFS, the detection accuracy of LogRobust ranges from 0.00 to 99.34 (avg $\approx$ 65.87, stdDev $\approx$ 46.58), while for NeuralLog1, it ranges from 53.17 to 98.92 (avg $\approx$ 89.16, stdDev $\approx$ 12.26). On Hadoop, the detection accuracy of LogRobust ranges from 7.77 to 90.50 (avg $\approx$ 76.62, stdDev $\approx$ 24.28), and on the F-dataset, the detection accuracy of SVM ranges from 0.00 to 71.43 (avg $\approx$ 22.29, stdDev $\approx$ 29.70). These results show high sensitivity in terms of *F1-score* across the session-based datasets. LSTM is the only ML technique that shows a high sensitivity to hyperparameter tuning across all the session-based datasets. Its detection accuracy ranges from 4.84 to 98.51 (avg $\approx$ 80.17, stdDev $\approx$ 25.40) on HDFS, from 38.96 to 90.50 (avg $\approx$ 83.45, stdDev $\approx$ 14.18) on Hadoop and from 0.00 to 67.15 (avg $\approx$ 21.50, stdDev $\approx$ 24.36) on F-dataset. Although NeuralLog1 and NeuralLog2 show a very small sensitivity to hyperparameter tuning on Hadoop, their detection accuracy on the F-dataset remains consistently 0 across all hyperparameter settings, indicating that the model is not learning. The same observation applies to LogRobust on the same dataset.

Figure 5 shows the sensitivity of the detection accuracy of supervised traditional and deep ML techniques across different window sizes on log message-based datasets.

- *Small window sizes* {10, 15, 20}. On small window sizes, the supervised ML techniques (except NeuralLog1 and NeuralLog2 on Hades) showed limited sensitivity in terms of detection accuracy to hyperparameter tuning on most of the log message-based datasets, in particular Thunderbird and Spirit. For instance, on Spirit, with $ws = 10$, the detection accuracy of RF is far less sensitive (*F1-score* avg $\approx$ 88.93, stdDev $\approx$ 0.35) than that of all the remaining supervised ML techniques. As for deep ML techniques, the *F1-score* observed for LSTM ranges from 80.71 to 96.52 (avg $\approx$ 93.05, stdDev $\approx$ 3.51); NeuralLog2 is the most sensitive deep ML technique showing an *F1-score* ranging from 39.96 to 99.97 (avg $\approx$ 92.24, stdDev $\approx$ 14.08).
- *Large window sizes* {50, 100, 150, 200, 250, 300}. The overall detection accuracy of all supervised ML techniques is more sensitive to hyperparameter tuning across most of the log message-based datasets and large window sizes. For instance, the detection accuracy of LogRobust on Spirit (see Fig. 5d) with $ws = 100$ ranges from 0.00 to 97.95 (avg $\approx$ 70.93, stdDev $\approx$ 37.50); RF is the least sensitive ML technique to hyperparameter tuning (*F1-score* avg $\approx$ 96.19, stdDev $\approx$ 0.49).

Overall, RF is the least sensitive supervised traditional ML technique to hyperparameter tuning in terms of detection accuracy across datasets. One possible reason of the stability of its detection accuracy across different datasets is its decision tree ensemble, which effectively averages out individual tree errors and mitigates overfitting, allowing it to maintain consistent detection accuracy. Except for Hades (see Fig. 5a), on which SVM is the most sensitive supervised ML technique[18] and both NeuralLog1 and NeuralLog2 are not learning on larger window sizes, showing a near 0 *F1-score* across hyperparameter settings[19], supervised deep ML techniques are more sensitive to hyperparameter tuning than supervised traditional ML techniques on the remaining datasets (BGL, Thunderbird, and Spirit) across window sizes.

---

[18] The choice of hyperparameter C highly impacts the detection accuracy of SVM, especially when evaluated on a highly imbalanced dataset like Hades: the larger the hyperparameter value, the bigger the misclassification penalty, leading to reduced bias towards the majority class in SVM.

[19] NeuralLog1 and NeuralLog2 rely on the transformer architecture, which requires large amounts of data. They therefore struggle when the training data is limited, and class imbalance further hinders their ability to differentiate between normal and anomalous log event sequences.
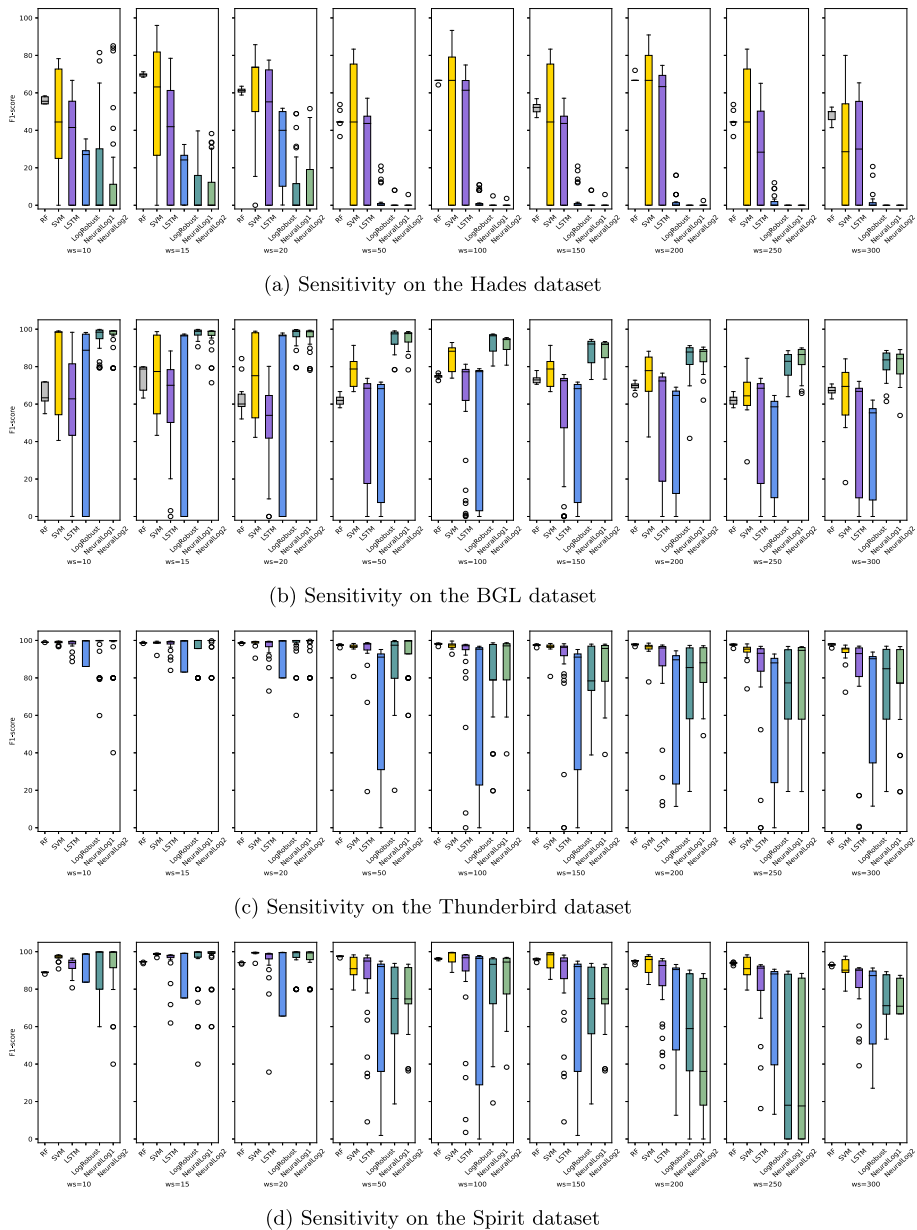
(a) Sensitivity on the Hades dataset

(b) Sensitivity on the BGL dataset

(c) Sensitivity on the Thunderbird dataset

(d) Sensitivity on the Spirit dataset

Fig. 5 Sensitivity of the detection accuracy of supervised traditional and deep ML techniques on log message-based datasets

LogRobust is particularly sensitive on BGL, Thunderbird, and Spirit, and NeuralLog1 and NeuralLog2 show increased sensitivity on Thunderbird and Spirit.

Statistical analysis (see § 5.5.4) yields a *p-value* of 0.052, suggesting the sensitivity of detection accuracy across supervised traditional and deep ML techniques is not a distinguishing factor on the seven datasets.

The answer to RQ1 is that the overall detection accuracy (*F1-score*) of supervised traditional (RF and SVM) and deep (LSTM, LogRobust, NeuralLog1 and NeuralLog2) ML techniques yields similar results on all the benchmark datasets except F-dataset, on which all deep ML models struggle to learn, resulting in poor predictions of log anomalies, whereas traditional ML techniques continue to perform well. In terms of specificity (*Spec*), all supervised traditional and deep ML techniques show high and similar values across most of the datasets (except for Hadoop), showing that the corresponding models accurately identify normal log event sequences. The low specificity of all supervised ML techniques on Hadoop is explained by the majority class consisting of anomalous log event sequences, making it difficult for these models to recognize the normal log event sequences.

Further, traditional ML techniques (especially RF) show much less sensitivity, in terms of detection accuracy, to hyperparameter tuning, compared to deep learning techniques on most of the datasets. Specifically, RF is the least sensitive on all datasets, followed by SVM which, in spite of being the most sensitive technique on Hadesand F-dataset, is less sensitive than deep ML techniques on the remaining datasets, across window sizes. Overall, deep ML techniques are the most sensitive techniques to hyperparameter tuning, with LSTM (followed by NeuralLog1 and NeuralLog2), showing more outliers across datasets.

All the studied traditional and deep ML techniques show their best detection accuracy (*F1-score*) on window sizes ranging from 10 to 50, across log message-based datasets (Table 8). As expected, we also observed that data imbalance has a negative impact on the detection accuracy of all supervised ML techniques across log message-based datasets: detection accuracy improves from more imbalanced datasets (Hades and BGL) to less imbalanced ones (Thunderbird and Spirit).

6.2 RQ2 - Time performance of supervised traditional and deep ML techniques

6.2.1 Time performance

In Table 10, we report the model re-training time[20] and the prediction time (rows *Re-train.* and *Pred.*, respectively) of the supervised traditional and deep ML techniques on session-based (HDFS, Hadoop, F-dataset) and log message-based (Hades, BGL, Thunderbird and Spirit) datasets. One important result is that the overall model re-training time of traditional ML techniques is about *one order of magnitude shorter* than that of deep learning techniques on all session-based datasets (HDFS, Hadoopand F-dataset).

We further study the impact of different window sizes on the time performance of supervised traditional and deep ML techniques across log message-based datasets.

- *Small window sizes* {10, 15, 20}. Supervised traditional ML techniques are faster (in terms of model re-training time) than deep ML techniques on Hades and Thunderbird datasets across small window sizes (see Table 8 for the window sizes associated with the highest detection accuracy and Table 10 for the model re-training time of the six supervised ML techniques considered in this study). For instance, on Hades, with $ws = 10$, RF takes 66.14 s, whereas NeuralLog1 shows the highest model re-training time (2479.34 s) among all the supervised ML techniques on the same dataset and window size. On Thunderbird, with $ws = 10$, RF takes 847.18 s ($\approx$ 14 min), whereas LSTM shows a much higher model re-training time of 16 135.18 s ($\approx$ 269 min). On the other hand, although LSTM shows

---

[20] Re-training refers to training the model again using the best hyperparameter settings. The data used to re-train the model consists of both the training set and the validation set from the training phase.

Table 10 Time performance (in seconds) of supervised traditional and deep ML techniques on all datasets

Dataset		Metric	Technique					
			SVM	RF	LSTM	LogRobust	NeuralLog1	NeuralLog2
Session	HDFS	*Re-train.*	397.64	96.01	2222.09	1135.53	57497.17	13718.53
		Pred.	23.88	0.89	3.01	3.53	70.10	180.64
	Hadoop	*Re-train.*	0.04	0.16	2.62	2.26	15.95	53.96
		Pred.	0.01	0.01	0.33	0.01	0.62	1.54
	F-dataset	*Re-train.*	0.07	2.11	29.49	2.12	13.87	13.87
		Pred.	0.03	0.01	0.49	0.01	0.48	0.48
Log message	Hades	*Re-train.*	4.45	66.14	178.03	327.60	2479.34	1974.00
		Pred.	0.65	0.28	0.61	0.69	13.37	12.22
	BGL	*Re-train.*	246.29	101.38	60.57	813.34	6377.34	8387.62
		Pred.	14.55	0.72	1.96	2.89	95.29	87.42
	Thunderbird	*Re-train.*	475.41	847.18	16135.18	856.60	25067.90	3329.26
		Pred.	32.60	3.61	11.71	3.38	127.37	42.21
	Spirit	*Re-train.*	2317.00	407.28	1416.07	1184.84	18582.84	2138.44
		Pred.	45.74	1.12	1.61	3.05	190.62	14.99

the lowest model re-training time on BGL with $ws = 15$ (60.57 s), the time taken by RF to re-train the model is relatively close (101.38 s). On Spirit, the model re-training time taken by RF (407.28 s) is by far lower than the time taken by all the remaining supervised techniques to re-train the corresponding models. Recall that small window sizes lead to more sequences to train the supervised ML models (see Section 6.1.1). This explains the longer model re-training time taken by all the supervised deep ML techniques, when trained on small window sizes.

- *Large window sizes* {50, 100, 150, 200, 250, 300}. All the supervised ML techniques show a short model re-training time across the log message-based datasets on large window sizes. For instance, on Thunderbird, with $ws = 50$, SVM takes 475.41 s, whereas re-training the same model takes longer (4928.68 s) with $ws = 10$. This shows that larger window sizes with fewer log event sequences fed into the ML models lead to shorter re-training time.

The prediction time computed for most of the supervised traditional and deep learning techniques (except for NeuralLog1 on Thunderbird and Spirit, and both NeuralLog1 and NeuralLog2 on HDFS and BGL) is similar on Hadoop, F-dataset and Hades, with no practically significant differences. More in detail, the prediction time is less than one minute for all the supervised techniques, ranging from 0.01 s for SVM, RF and LogRobust on Hadoop dataset to 45.74 s for SVM on Spirit dataset. However, the prediction time of NeuralLog2 on HDFS takes 180.64 s and that of NeuralLog1 on Spirit takes 190.62 s. Therefore, prediction time is generally not a distinguishing factor among most of the techniques, except for the transformer-based models (NeuralLog1 and NeuralLog2), which tend to have significantly longer prediction times due to their complex architecture. Indeed, the number of transformer layers $ffnS$ and the number of attention heads $attH$ control the learning ability of the transformer-based model to capture complex patterns and dependencies in log messages (see Section 2).

Statistical analysis (see § 5.5.4) indicate that the time performance of the supervised traditional and deep ML techniques, in terms of model re-training time is not significantly different, showing a *p-value* of 0.0826. While the prediction time of all the supervised ML techniques is significantly different (*p-value* = 0.047), no significant pairwise difference using the post-hoc statistical Dunn's test is observed (no pair of ML techniques shows a *p-value* smaller than 0.05).

6.2.2 Sensitivity of Training Time

As depicted in Fig. 6, the overall model training time of traditional ML techniques (RF and SVM) is much less sensitive to hyperparameter tuning than that of deep ML techniques on the three session-based datasets. For instance, RF takes from 13.24 s to 130.64 s to train the model (avg $\approx$ 71.98 s, stdDev $\approx$ 37.43 s) on HDFS (box plot in Fig. 6a) and SVM takes from 0.029 s to 0.092 s to train the model (avg $\approx$ 0.042 s, stdDev $\approx$ 0.019 s) on Hadoop (box plot in Fig. 6b).

Regarding deep ML techniques, the model training time of LogRobust is less sensitive to hyperparameter tuning than that of the remaining techniques on all the session-based datasets. For instance, on HDFS, LogRobust takes from 632.88 s to 2334.76 s (avg $\approx$ 1285.13 s,

(a) HDFS dataset   (b) Hadoop dataset
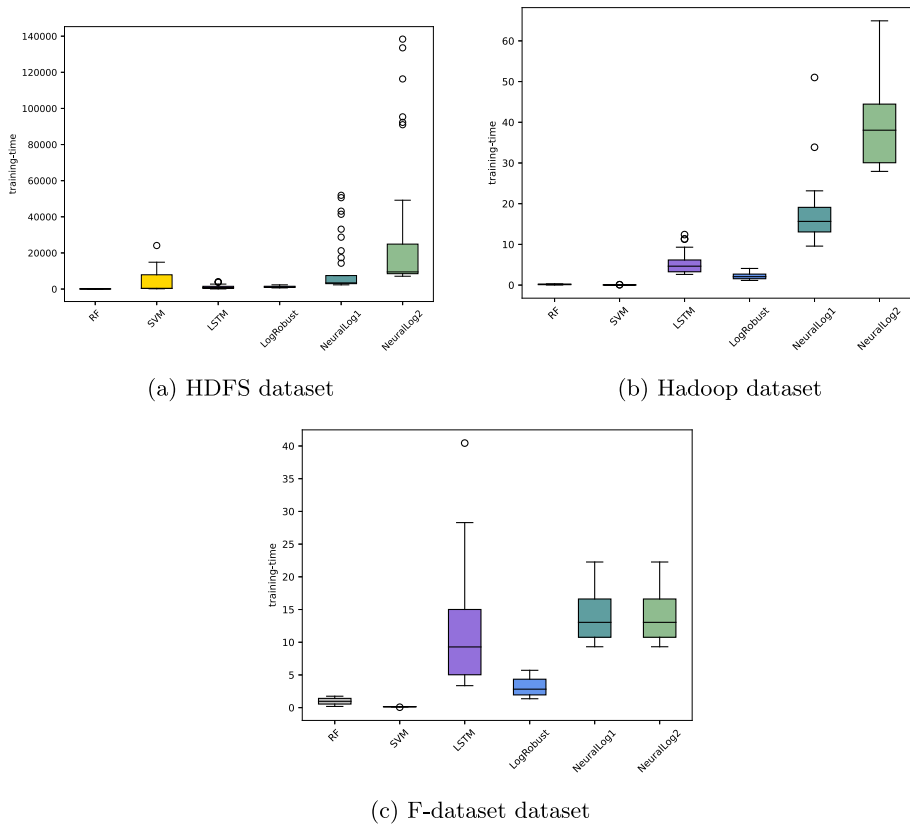
(c) F-dataset dataset

Fig. 6 Sensitivity of the time performance (in seconds) of supervised traditional and deep ML techniques on session-based datasets (The difference in the y-axis scale of the three plots is due to the difference in the training size of the datasets)

stdDev $\approx$ 490.36 s), whereas the model training time of NeuralLog2 ranges from 7090.97 s to 138 370.41 s (avg $\approx$ 29 868.56 s, stdDev $\approx$ 38 664.96 s) to train the corresponding models. This is expected, given the complex architecture of transformer-based models.

Figure 7 shows the sensitivity of training time of supervised traditional and deep ML techniques across different window sizes on log message-based datasets.

(a) Sensitivity on Hades dataset

(b) Sensitivity on BGL dataset

(c) Sensitivity on Thunderbird dataset

(d) Sensitivity on Spirit dataset
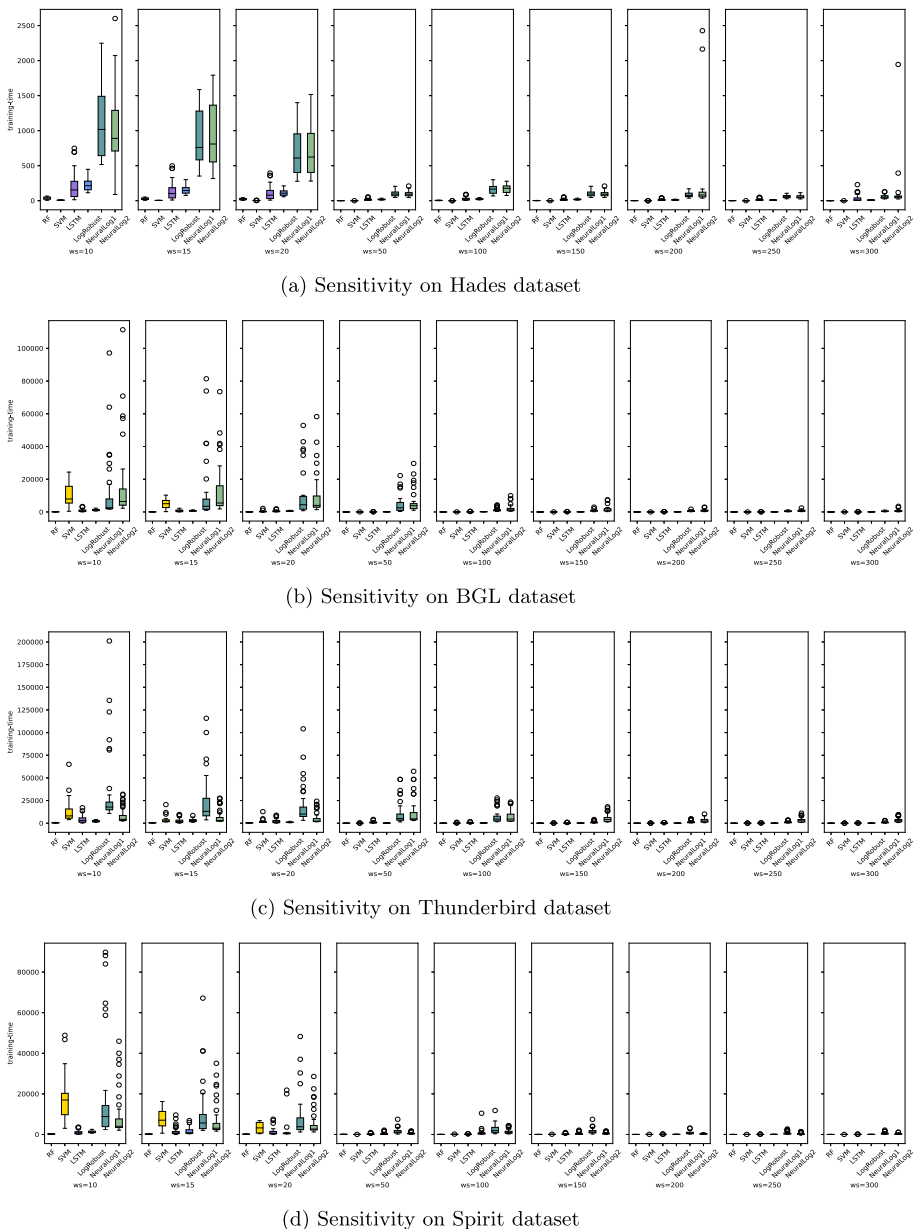
Fig. 7 Sensitivity of the time performance (in seconds) of supervised traditional and deep ML techniques on four log message-based datasets

- *Small window sizes* {10, 15, 20}. The overall model training of RF is less sensitive to hyperparameter tuning (with no outliers) than that of the remaining supervised deep ML techniques across most of the datasets and window sizes. The only exception we observe is on window sizes ranging from 10 to 50 on Hades, in which RF is slightly more sensitive to hyperparameter tuning than SVM. Nevertheless, the difference in model training between RF and SVM on these window sizes is negligible. For instance, on Hades, with $ws = 10$, RF takes from 6.43 s to 65.45 s (avg $\approx$ 36.41 s, stdDev $\approx$ 19.03 s), whereas SVM takes from 4.71 s to 12.31 s (avg $\approx$ 7.33 s, stdDev $\approx$ 2.90 s) to train the corresponding model. The two versions of NeuralLog (NeuralLog1 and NeuralLog2) show the highest sensitivity to hyperparameter tuning on small window sizes on all log message-based datasets with many outliers. For instance, on Hades, with $ws = 10$, NeuralLog1 takes from 516.72 s to 2247.86 s (avg $\approx$ 1127.87 s, stdDev $\approx$ 516.89 s) to train the corresponding model. On Spirit, on the same window size, the model training time of the latter technique takes from 2430.81 s to 89 768.24 s (avg $\approx$ 18 852.15 s, stdDev $\approx$ 25 835.39 s)
- *Large window sizes* {50, 100, 150, 200, 250, 300}. Overall, the model training time of all supervised ML techniques is much less sensitive to hyperparameter tuning across large window sizes than that observed on small window sizes. For instance, on BGL, SVM takes from 7.04 s to 38.01 s (avg $\approx$ 14.18 s, stdDev $\approx$ 7.02 s) with $ws = 100$. This implies that larger window sizes, with fewer log event sequences fed to the ML models, result in a lower sensitivity to hyperparameter tuning of the model training time for all the supervised ML techniques.

Statistical analysis (see § 5.5.4) indicates that the sensitivity of the training time of the different supervised traditional and deep ML techniques across hyperparameter settings is not significantly different (*p-value*=0.11).

The answer to RQ2 is that, except for the transformer-based ML techniques NeuralLog1 and NeuralLog2, the remaining supervised traditional (RF and SVM) and deep (LSTM and LogRobust) ML techniques show similar model prediction time with no practically significant differences across the different session-based (HDFS, Hadoopand F-dataset) and log message-based (Hades, BGL, Thunderbird and Spirit) datasets.

The model re-training time taken by traditional ML techniques (especially RF) is, however, significantly lower than the time taken by deep ML techniques on most of the datasets, notably HDFS, Hades and Thunderbird.

Overall, supervised traditional and deep ML techniques generally i) take less time for model training and ii) are less sensitive to hyperparameter tuning when using large window sizes compared to small ones. This trend holds across both more imbalanced datasets like Hades and BGL and less imbalanced ones such as Thunderbird and Spirit. Notably, the model training time of RF shows less sensitivity to hyperparameter tuning on all session-based and log message-based datasets compared to deep ML techniques across window sizes.

6.3 RQ3 - Detection Accuracy of Semi-supervised Traditional and Deep ML Techniques

6.3.1 Detection Accuracy

As shown in Table 11, DeepLog far outperforms OC-SVM and Logs2Graphs on HDFS in terms of detection accuracy with a notable difference of 26.07 pp and 20.14 pp (pp = percentage points), respectively. A recent study in log-based datasets (Landauer et al. 2024)

Table 11 Comparison of the detection accuracy of semi-supervised traditional and deep ML techniques on all datasets

Dataset		Metric	Technique		
			OC-SVM	DeepLog	Logs2Graphs
Session	HDFS	*Prec*	46.32	93.86	95.27
		Rec	67.94	72.19	44.88
		F1	55.09	81.16	61.02
		Spec	89.31	96.82	99.70
	Hadoop	*Prec*	50.36	51.01	50.68
		Rec	86.50	90.80	92.02
		F1	63.66	64.54	65.36
		Spec	7.95	9.17	3.31
	F-dataset	*Prec*	63.04	61.90	0.00
		Rec	100.00	95.37	0.00
		F1	77.33	75.06	0.00
		Spec	0.00	0.00	100.00
Log message	Hades	*Prec*	89.19	66.92	99.02
		Rec	64.71	48.85	80.16
		F1	75.00	56.46	80.60
		Spec	99.42	99.38	100.0
	BGL	*Prec*	43.65	70.86	96.86
		Rec	96.80	64.70	84.35
		F1	60.16	67.61	90.17
		Spec	42.06	90.48	92.86
	Thunderbird	*Prec*	74.09	76.73	87.44
		Rec	81.95	63.11	17.22
		F1	77.82	67.98	90.55
		Spec	13.33	42.11	96.82
	Spirit	*Prec*	74.15	90.89	99.87
		Rec	86.65	77.87	91.18
		F1	79.91	83.87	95.33
		Spec	43.73	78.57	99.71

shows a high redundancy in log event sequences within HDFS (a total of 575061 sequences can be reduced to 26814 sequences only). So many nearly identical event sequences make the index-based encoding technique DeepLog more effective than semantics-based encoding techniques (OC-SVM and Logs2Graphs) on that dataset. More in detail, index-based encoding preserves the order of log event occurrences within sequences and handles the high redundancy of the dataset, whereas semantics-based encoding struggles to capture the differences in order, leading to reduced detection accuracy.

However, the detection accuracy of the semi-supervised ML techniques is very similar on Hadoop and F-dataset (with a difference in detection accuracy of only 0.88 pp and 2.27 pp, respectively), with the exception of Logs2Graphs on the latter dataset (*F1-score*=0.00), suggesting that the corresponding ML model is not able to detect anomalous log event sequences on that dataset. In terms of specificity, all the semi-supervised ML techniques show a high value on HDFS, indicating their ability to detect normal log event sequences. They, how-

ever, do not perform well on Hadoop and F-dataset. The only exception on F-dataset is for Logs2Graphs (*Spec*=100.00), which perfectly detects normal log event sequences. According to a recent study (Landauer et al. 2024), there is a high overlap in the Hadoopdataset, in the sense that 83.2% of normal log event sequences contain at least one log event sequence that also appears in anomalous log event sequences. Additionally, 75.5 % of anomalous log event sequences are identical to normal ones. Therefore, this overlap makes it difficult for the semi-supervised ML models to effectively distinguish between normal and anomalous log event sequences, resulting in poor detection accuracy (the *F1-score* ranges from 63.66 for OC-SVM to 65.36 for Logs2Graphs) and very low specificity values, ranging from 3.31 for Logs2Graphs to 9.17 for DeepLog.

Logs2Graphs far outperforms OC-SVM and DeepLog in terms of *F1-score* and *Spec* on all log message-based datasets. This suggests that the GNN-based semi-supervised approach is more effective at detecting log anomalies compared to the traditional OC-SVM and the RNN-based DeepLog, demonstrating a superior ability to differentiate between normal and anomalous log event sequences. For instance, on Thunderbird, Logs2Graphs achieves an *F-score* of 90.55 and a *Spec* of 96.82, by far outperforming DeepLog with an *F-score* of 67.98 and a *Spec* of 42.11, and OC-SVM with an *F-score* of 77.82 and a *Spec* of 13.33. We also observe that the specificity of Logs2Graphs is not impacted by the imbalance ratio (IR, see Table 5). For instance, the highest specificity (*Spec*=100.00%) of the latter is recorded on Hades, with an IR of 0.13% at $ws = 10$, while its lowest specificity (*Spec*=92.86%) is recorded on BGL, with an IR of 11.38% at $ws = 200$. Although DeepLog outperforms OC-SVM on the two log message-based datasets BGL and Spirit, the difference in detection accuracy (*F1-score*) is relatively small when compared to the one observed on HDFS: the difference in *F1-score* value is 7.45 pp and 3.96 pp on BGL and Spirit respectively. OC-SVM, however, outperforms DeepLog on the remaining two log message-based datasets (Hades and Thunderbird).

Specificity, however, is a distinguishing factor for OC-SVM and DeepLog on log message-based datasets, except for Hades. For instance, the difference in specificity is 28.78 pp on Thunderbird and 48.42 pp on BGL.

Overall, both OC-SVM and DeepLog show a decreasing specificity from more imbalanced (Hades and BGL) to less imbalanced (Thunderbird and Spirit) datasets, given that the imbalance ratio (IR) on the former datasets is much less than that on the latter datasets (see Table 5).

For instance, DeepLog achieves its highest specificity (*Spec*=99.38%) on Hades (the most imbalanced dataset), with an IR of 0.13% at $ws = 10$, while its lowest specificity (*Spec*=42.11%) is recorded on Thunderbird on $ws = 200$, with an IR=39.81%. Similarly, the highest specificity of OC-SVM is recorded on Hades, with an IR of 1.6% at $ws = 300$, while its lowest specificity is recorded on Thunderbird, with an IR of 40.19% at $ws = 250$. This trend in terms of specificity reflects the ability of all traditional (OC-SVM) and deep (DeepLog and Logs2Graphs) semi-supervised ML techniques to better distinguish normal from anomalous log event sequences in datasets with lower *IR* (Hades and BGL), reflecting that the identification of normal log event sequences decreases with the increase of the imbalance ratio.

The difference in detection accuracy (F1-score) between OC-SVM and DeepLog is higher on the Hades and Thunderbird datasets than on the BGL and Spirit ones, showing a higher ability of DeepLog at detecting anomalous log event sequences on these datasets. More in detail, the difference between the detection accuracy of both semi-supervised ML techniques on Hades is 18.54 pp, whereas it is 9.84 pp on Thunderbird. In terms of specificity, DeepLog
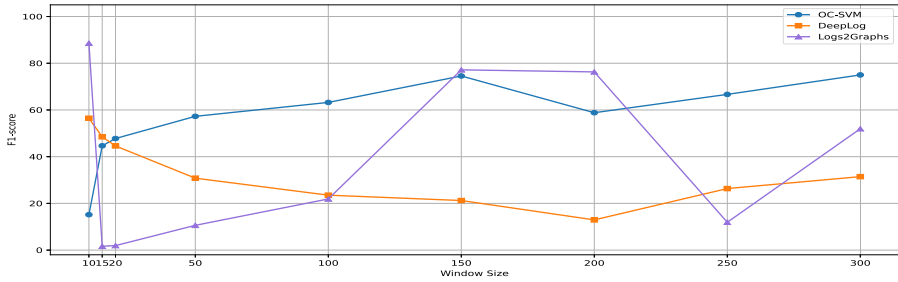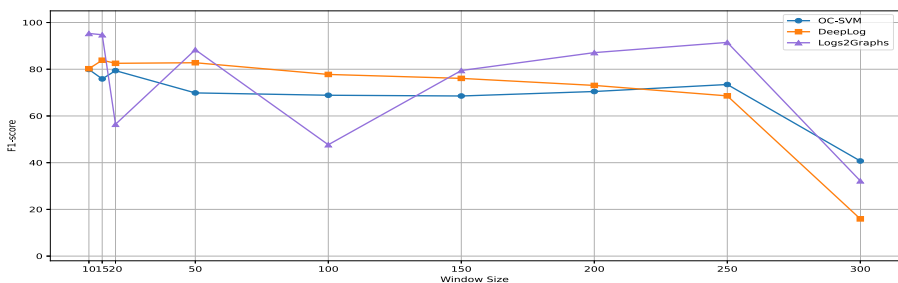
(a) Impact of window size when using the Hades dataset
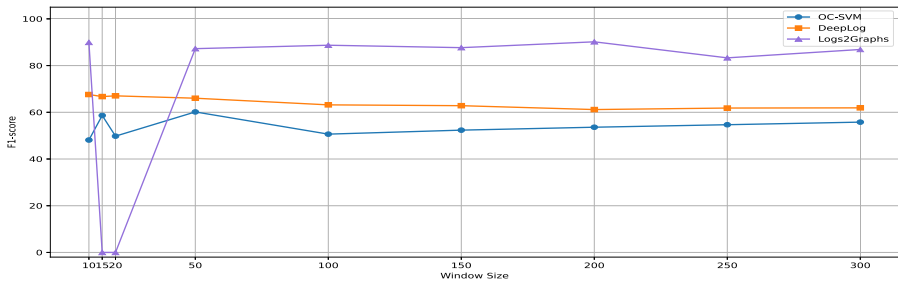
(b) Impact of window size when using the BGL dataset
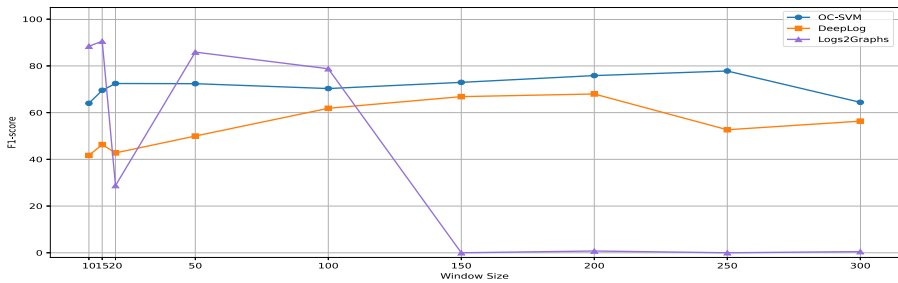
(c) Impact of window size when using the Thunderbird dataset

(d) Impact of window size when using the Spirit dataset

Fig. 8 Impact of window size on the detection accuracy of semi-supervised traditional and deep ML techniques on log message-based datasets

shows a better ability at avoiding false positives than OC-SVM on BGL, Thunderbird and Spirit, with a difference in specificity values of 48.42 pp, 28.78 pp and 34.84 pp, respectively.

Figure 8 shows the impact of different window sizes on the detection accuracy of semi-supervised traditional and deep ML techniques on log message-based datasets.

- *Small window sizes* {10, 15, 20}. As depicted in Table 8, both deep ML techniques achieve their highest detection accuracy with smaller window sizes on three out of four datasets, with DeepLog showing its highest detection accuracy in terms of *F1-score* on Hades, BGL and Spirit, and Logs2Graphs on Hades, Thunderbird and Spirit.
- *Large window sizes* {50, 100, 150, 200, 250, 300}. As shown in Table 8, OC-SVM yields its highest detection accuracy on larger window sizes on three (Hades, BGL and Thunderbird) out of the four log message-based datasets. Overall, large window sizes are deemed more suitable for OC-SVM in detecting execution path log anomalies on log message-based datasets.

To conclude, the detection accuracy of semi-supervised traditional (OC-SVM) and deep ML (DeepLog and Logs2Graphs) techniques varies across different window sizes. Our findings related to DeepLog are consistent with a recent empirical study (Le and Zhang 2022), which also reported similar variations in detection accuracy across different window sizes for semi-supervised deep ML techniques, including DeepLog.

Statistical analysis (see § 5.5.4) yields a *p-value* of 0.56, suggesting the detection accuracy of semi-supervised traditional and deep ML techniques is not significantly different.

6.3.2 Sensitivity of Detection Accuracy

As depicted in Fig. 9, the detection accuracy of Logs2Graphs is more sensitive to hyperparameter tuning than the remaining semi-supervised ML techniques across all the session-based datasets. For instance, on HDFS (Fig. 9a), the detection accuracy of Logs2Graphs ranges from 0.00 to 72.61 (avg $\approx$ 11.43, stdDev $\approx$ 18.21), whereas that of DeepLog ranges from 55.38 to 81.12 (avg $\approx$ 68.46, stdDev $\approx$ 6.65) and that of OC-SVM ranges from 5.67 to 31.64 (avg $\approx$ 11.92, stdDev $\approx$ 5.66). Although the difference in sensitivity between OC-SVM and DeepLog on HDFS is small (with outliers recorded for OC-SVM), OC-SVM is generally much more sensitive to hyperparameter tuning than DeepLog on both Hadoop and F-dataset. For instance, on Hadoop, the detection accuracy of DeepLog ranges from 36.78 to 79.47 (avg $\approx$ 43.86, stdDev $\approx$ 8.84), whereas the detection accuracy computed for OC-SVM ranges from 13.64 to 88.89 (avg $\approx$ 66.88, stdDev $\approx$ 21.15).

Figure 10 shows the sensitivity of the detection accuracy of semi-supervised traditional and deep ML techniques across different window sizes on log message-based datasets.

- *Small window sizes* {10, 15, 20}. On small window sizes, the detection accuracy of semi-supervised deep ML techniques (notably Logs2Graphs) is more sensitive to hyperparameter tuning on more imbalanced datasets (Hades and BGL) than the less imbalanced ones (Thunderbird and Spirit). For instance, the detection accuracy (in terms of *F1-score*) of Logs2Graphs on Hades (the most imbalanced dataset) ranges from 0.00 to 96.00 (avg $\approx$ 27.72, stdDev $\approx$ 35.13), whereas the detection accuracy of the latter technique ranges from 32.74 to 95.12 (avg $\approx$ 87.58, stdDev $\approx$ 12.13) on Spirit (the least imbalanced dataset). In contrast, OC-SVM is less sensitive to hyperparameter tuning on more imbalanced datasets than the less imbalanced ones. For instance, on Hades, the detection accuracy of OC-SVM ranges from 0.27 to 6.50 (avg $\approx$ 1.23, stdDev $\approx$ 1.53) with $ws = 10$, whereas the detection accuracy of DeepLog ranges from 18.05 to 46.94 (avg
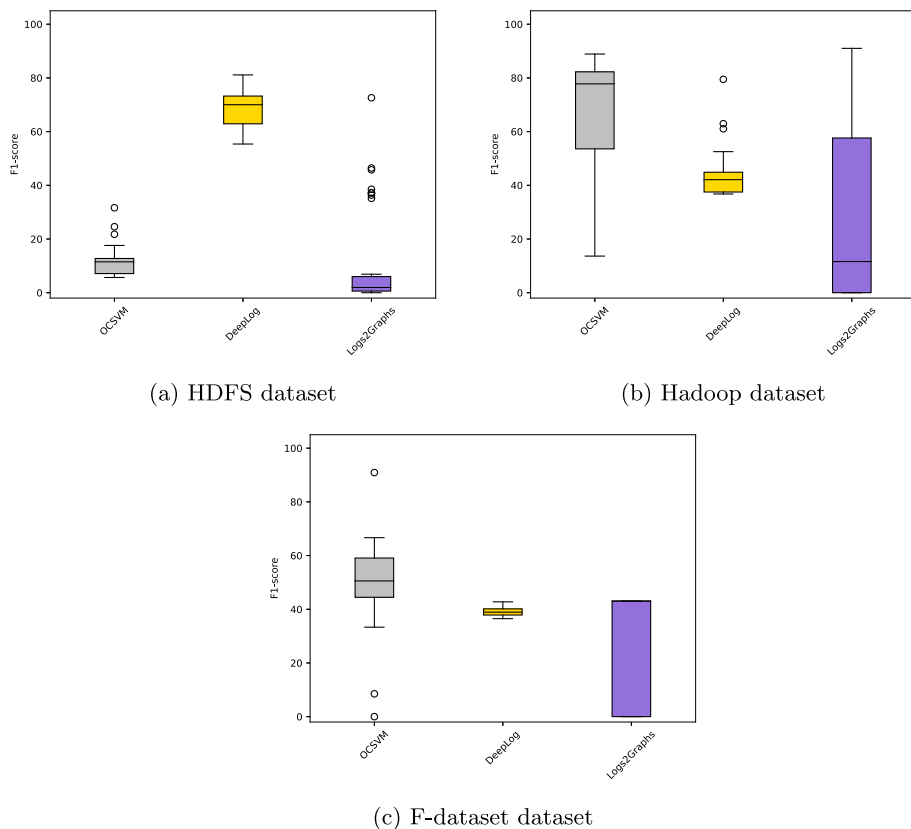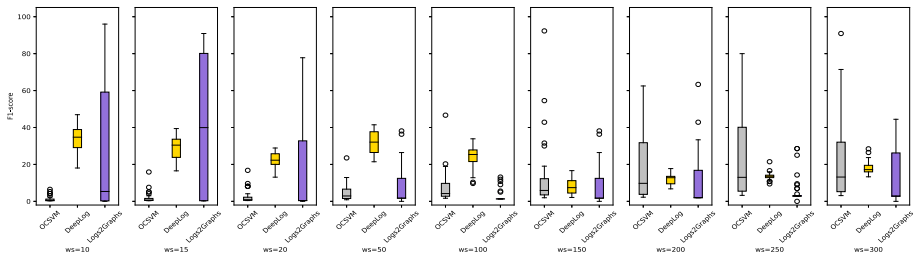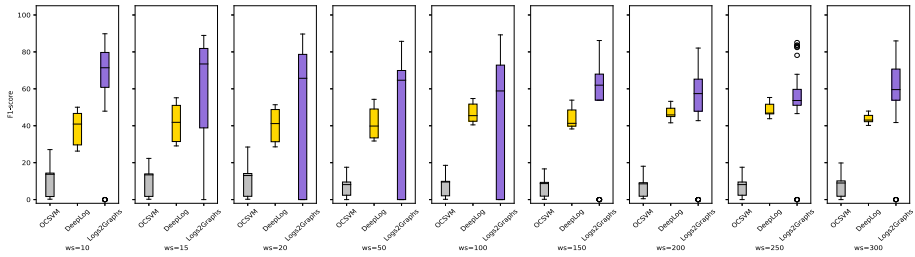
(a) HDFS dataset

(b) Hadoop dataset

(c) F-dataset dataset

Fig. 9 Sensitivity of the detection accuracy of semi-supervised traditional and deep ML techniques on session-based datasets

≈ 34.21, stdDev ≈ 7.29) on the same window size. Overall, the results show that data imbalance has an impact on sensitivity in terms of detection accuracy of semi-supervised, traditional and deep ML techniques.

- *Large window sizes* {50, 100, 150, 200, 250, 300}. The overall detection accuracy of OC-SVM (in terms of *F1-score*) is more sensitive to hyperparameter tuning than that of the remaining semi-supervised deep ML techniques on large window sizes on Spirit and Hades datasets. For instance, on Spirit, with $ws = 300$, the detection accuracy of OC-SVM ranges from 0.00 to 93.76 (avg ≈ 61.39, stdDev ≈ 32.31), whereas the detection accuracy of Logs2Graphs ranges from 0.23 to 68.13 (avg ≈ 22.69, stdDev ≈ 25.10). However, on BGL and Thunderbird, Logs2Graphs shows more sensitivity of detection accuracy to hyperparameter tuning than the remaining semi-supervised techniques. For instance, on BGL, with $ws = 300$, the detection accuracy of Logs2Graphs ranges from 0.00 to 85.95 (avg ≈ 53.83, stdDev ≈ 26.34) whereas that of the OC-SVM ranges from 0.00 to 19.84 (avg ≈ 8.01, stdDev ≈ 5.19) and that of DeepLog ranges from 40.18 to 47.94 (avg ≈ 43.75, stdDev ≈ 2.14), indicating that the latter technique is the most suitable semi-supervised ML technique to detect log anomalies on log message-based datasets on larger contexts.

(a) Sensitivity on Hades dataset

(b) Sensitivity on BGL dataset
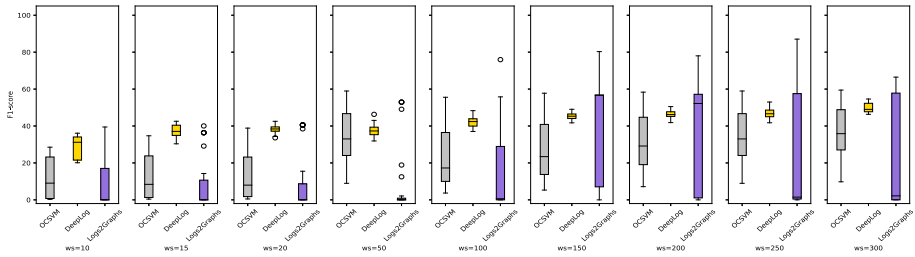
(c) Sensitivity on Thunderbird dataset
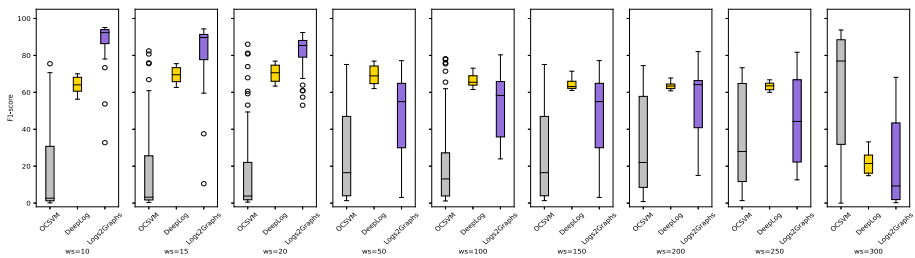
(d) Sensitivity on Spirit dataset

Fig. 10 Sensitivity of the detection accuracy of semi-supervised traditional and deep ML techniques on log message-based datasets

Table 12 Pairs of ML techniques with significant differences in the sensitivity of the *F1-score* to hyperparameter tuning	ML.1	ML.2	p-value
	DeepLog	Logs2Graphs	0.0077
	DeepLog	OC-SVM	0.0476

The statistical test (see § 5.5.4) indicates that the sensitivity of the detection accuracy of the different semi-supervised ML techniques across hyperparameter settings is significantly different, showing a *p-value* of 0.006. The results of the post-hoc analysis based on Dunn's test are shown in Table 12, which includes the pairs of ML techniques (Columns *ML.1* and *ML.2*) that show statistically significant differences in terms of the sensitivity of their detection accuracy to hyperparameter tuning.

The answer to RQ3 is that all semi-supervised traditional (OC-SVM) and deep (DeepLog and Logs2Graphs) ML techniques do not fare well in terms of detection accuracy. Moreover, the overall detection accuracy of semi-supervised ML techniques and their sensitivity to hyperparameter tuning vary greatly across datasets. We also observe that the detection accuracy of the semi-supervised techniques varies across log message-based datasets with different window sizes: OC-SVM performs better than DeepLog on small window sizes, when evaluated on Thunderbird. Its detection accuracy, however, reaches its maximum on large window sizes on the remaining log message-based datasets. Further, Logs2Graphs outperforms OC-SVM and DeepLog on large window sizes, when evaluated on BGL. Its detection accuracy, however, reaches its maximum on smaller window size on the remaining datasets.

6.4 RQ4 - Time Performance of Semi-supervised Traditional and Deep ML Techniques

6.4.1 Time Performance

Table 13 shows that DeepLog performs much better, in terms of model re-training time and prediction time, than OC-SVM and Logs2Graphs on HDFS: it takes $\approx$ 40 min to retrain the corresponding model and 100.23 s to detect log anomalies, whereas OC-SVM and Logs2Graphs take 53 896.93 s ($\approx$ 15 h) and 65 495.75 s ($\approx$ 18 h) for the model re-training, and $\approx$ 1.7 h and $\approx$ 2 min for log anomaly prediction, respectively. The faster model re-training time of DeepLog is due to its index-based embedding of log event sequences, in contrast to the 300-dimensional vectors fed to other semi-supervised techniques (see Section 4), resulting in lower-dimensional input. However, DeepLog takes much longer than OC-SVM and Logs2Graphs in terms of model re-training time on Hadoop and F-dataset, respectively. The significantly higher model re-training time observed for DeepLog compared to OC-SVM and Logs2Graphs can be explained by the difference in the number of log event sequences processed during their respective model re-training processes: DeepLog is trained with 130172 and 704474 log event sequences[21], whereas OC-SVM and Logs2Graphs are trained with 648 and 951 log event sequences only representing 80% of the majority class (see Table 3) on Hadoop and F-dataset, respectively. However, the prediction time of all semi-supervised techniques on Hadoop and F-dataset is very close (< 0.01 s for OC-SVM,

---

[21] The values 130172 and 704474 correspond to the number of sequences obtained by applying a sliding window of size 10 with a step size of 1 on Hadoop and F-dataset, respectively in DeepLog's implementation (wuyifan18 2020).

Table 13 Time performance (in seconds) of semi-supervised traditional and deep ML techniques on all datasets

Dataset		Metric	Technique		
			OC-SVM	DeepLog	Logs2Graphs
Session	HDFS	*Re-train.*	53896.93	2398.04	65495.75
		Pred.	6180.44	100.23	122.19
	Hadoop	*Re-train.*	0.01	944.49	132.99
		Pred.	0.00	0.24	0.74
	F-dataset	*Re-train.*	0.05	5110.91	90.03
		Pred.	0.07	13.13	0.21
Log message	Hades	*Re-train.*	0.17	229.15	1170.75
		Pred.	0.04	1.85	4.66
	BGL	*Re-train.*	295.87	1014.52	54.91
		Pred.	45.20	4.56	1.05
	Thunderbird	*Re-train.*	100.25	742.90	1310.45
		Pred.	108.81	60.43	65.37
	Spirit	*Re-train.*	24110.32	866.07	12326.49
		Pred.	11060.41	51.38	39.07

0.24 s for DeepLog and 0.74 s for Logs2Graphs on Hadoop, whereas < 0.08 s for OC-SVM, 13.13 s for DeepLog and 0.21 s for Logs2Graphs on F-dataset).

Recall that small window sizes lead to more sequences to train the supervised ML models (see Section 6.1.1). We observe that, on the Hades, BGL and Thunderbird datasets, OC-SVM takes less model re-training time than DeepLog and Logs2Graphs due to the fewer sequences it uses for training, as compared to DeepLog and Logs2Graphs. More in detail, the highest detection accuracy of OC-SVM on Hades, BGL and Thunderbird is associated with larger window sizes (300, 50 and 250, respectively) than DeepLog (10, 10 and 200, respectively) and Logs2Graphs (10, 200 and 15, respectively), leading to fewer sequences fed to OC-SVM than those fed to the remaining semi-supervised ML techniques (see Table 5). Logs2Graphs, however, takes less model re-training time than OC-SVM on BGL due to the fewer sequences used for training with $ws = 200$.

We further study the impact of different window sizes on the time performance of semi-supervised traditional and deep ML techniques across log message-based datasets.

- *Small window sizes* {10, 15, 20}. As shown in Table 13, DeepLog and Logs2Graphs show a much higher model re-training time than OC-SVM on Hades. For instance, on Hades, the model re-training time of DeepLog and Logs2Graphs is 229.15 s and 1170.75 s with $ws = 10$, whereas OC-SVM takes only 0.17 s with $ws = 300$. The longer model re-training time of both semi-supervised deep ML techniques is expected as they achieve their highest detection accuracy with smaller window sizes, resulting in more log event sequences fed to the corresponding ML models (see Table 8 for the window sizes associated with the highest detection accuracy and Table 5 for the number of log event sequences generated across different window sizes). More in detail, on Hades, DeepLog and Logs2Graphs take 83774 sequences in input, generated with $ws = 10$, whereas OC-SVM is fed with only 2751 sequences generated with $ws = 300$ However, on Spirit, DeepLog outperforms both OC-SVM and Logs2Graphs in terms of model re-training time, since it is fed with much fewer log event sequences (184060, generated

with $ws = 15$) than OC-SVM and Logs2Graphs, which are fed with 282913 log event sequences, generated with $ws = 10$.

In terms of prediction time, Logs2Graphs outperforms both OC-SVM and DeepLog on Spirit. For instance, Logs2Graphs takes only 39.07 s with $ws = 10$, whereas OC-SVM takes 11 060.41 s ($\approx$ 3 h) with the same window size and DeepLog takes 51.38 s with $ws = 15$. Further, although OC-SVM is quicker (for prediction) than DeepLog and Logs2Graphs on Hades, the difference in prediction time is not significant. This suggests that deep ML techniques (notably Logs2Graphs) are more suitable at predicting log anomalies, especially for small window sizes.

- *Large window sizes* {50, 100, 150, 200, 250, 300}. On large window sizes, Logs2Graphs is much slower than OC-SVM and DeepLog on Thunderbird. More in detail, Logs2Graphs takes 1310.45 s with $ws = 15$, whereas OC-SVM and DeepLog take 742.9 s with $ws = 200$ and 100.25 s with $ws = 250$, respectively. The shorter model re-training time of OC-SVM and DeepLog is due to the fewer log event sequences fed to the corresponding models on large window sizes.

(a) HDFS dataset   (b) Hadoop dataset
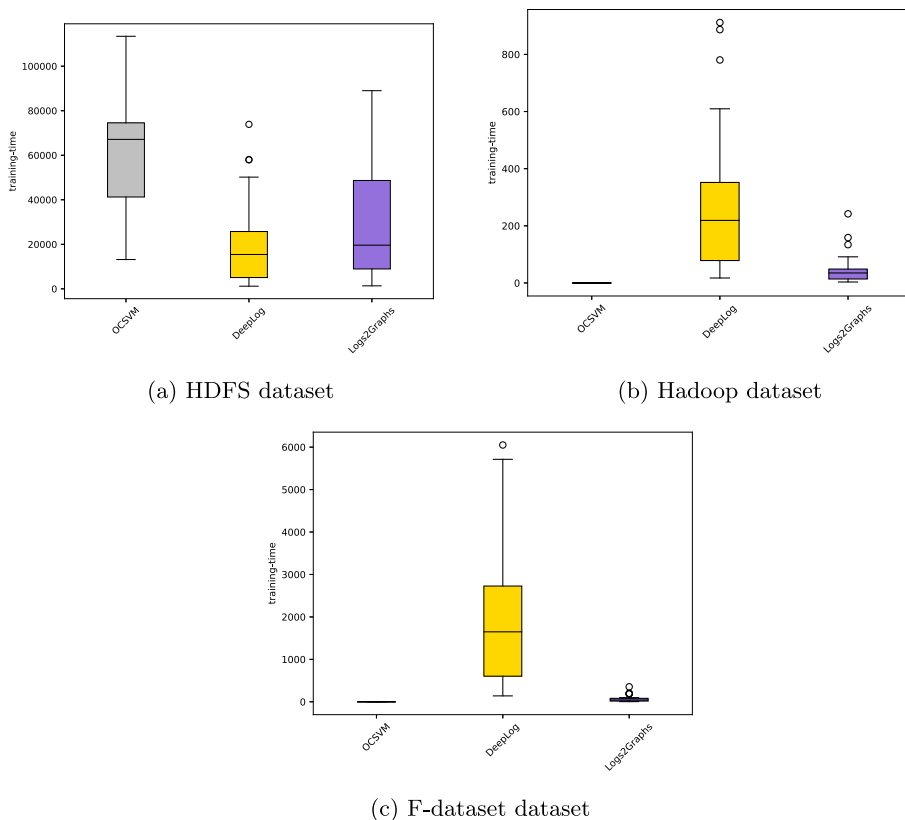
(c) F-dataset dataset

Fig. 11 Sensitivity of the time performance (in seconds) of semi-supervised traditional and deep ML techniques on session-based datasets (The y-scale of the three plots is different due to the different training size of the three datasets)

Statistical analysis (see § 5.5.4) indicates that the time performance of both semi-supervised traditional and deep ML techniques in terms of model re-training and prediction time is not significantly different, showing a *p-value* of 0.4437 and 0.9744, respectively.

6.4.2 Sensitivity of Training Time

As depicted in Fig. 11a, DeepLog is less sensitive to hyperparameter tuning (with outliers) than OC-SVM and Logs2Graphs on HDFS: its model training time ranges from 1183.43 s to 73 879.67 s (avg $\approx$ 20 133.51 s, stdDev $\approx$ 18 286.96 s), whereas OC-SVM takes from 13 157.73 s to 113 438.99 s (avg $\approx$ 60 369.42 s, stdDev $\approx$ 25 798.33 s) and Logs2Graphs takes from 1346.73 s to 89 010.09 s (avg $\approx$ 30 675.87 s, stdDev $\approx$ 25 202.10 s) on the same dataset. However, on Hadoopand F-dataset (Fig. 11b and c, respectively), the time performance of OC-SVM is far less sensitive to hyperparameter tuning than Deeplog and Logs2Graphs, with outliers of the latter techniques on both datasets. For instance, on F-dataset, the model training of OC-SVM ranges from 0.02 s to 0.16 s (avg $\approx$ 0.10 s, stdDev $\approx$ 0.04 s), whereas DeepLog takes from 139.69 s to 6050.34 s (avg $\approx$ 1972.07 s, std-Dev $\approx$ 1665.83 s) and Logs2Graphs takes from 4.06 s to 353.71 s (avg $\approx$ 65.05 s, stdDev $\approx$ 74.23 s) for its model training.

Figure 12 shows the sensitivity of training time of semi-supervised traditional and deep ML techniques across different window sizes on log message-based datasets.

- *Small window sizes* {10, 15, 20}. Overall, DeepLog shows much less sensitivity in terms of model training time to hyperparameter tuning than OC-SVM and Logs2Graphs, on small window sizes across all the log message-based datasets. For instance, on Hades (see Fig. 12a) with $ws = 10$, DeepLog takes from 12.18 s to 618.90 s (avg $\approx$ 184.27 s, stdDev $\approx$ 162.77 s) for its model training whereas OC-SVM takes from 460.82 s to 2054.20 s (avg $\approx$ 1498.93 s, stdDev $\approx$ 511.85 s), while Logs2Graphs takes from 193.34 s to 10 034.02 s (avg $\approx$ 2520.01 s, stdDev $\approx$ 2194.36 s) on the same dataset and window size. This indicates that training size has more impact on the sensitivity of OC-SVM and Logs2Graphs to hyperparameter tuning than that of DeepLog in terms of model training time.
- *Large window sizes* {50, 100, 150, 200, 250, 300}. Overall, all the semi-supervised ML techniques show less sensitive model training time to hyperparameter tuning on large window sizes, ranging from $ws = 50$ to $ws = 300$ across all log message-based datasets. The model training time of OC-SVM, however, is less sensitive to hyperparameter tuning than that of the remaining semi-supervised techniques. For instance, on Spirit, with $ws = 150$, OC-SVM takes from 10.44 s to 61.20 s (avg $\approx$ 38.83 s, stdDev $\approx$ 15.07 s), whereas the training time of DeepLog ranges from 13.52 s to 847.51 s (avg $\approx$ 154.71 s, stdDev $\approx$ 166.61 s) and Logs2Graphs ranges from 83.91 s to 4523.87 s (avg $\approx$ 1381.98 s, stdDev $\approx$ 1281.64 s). Thus, training size affects more significantly the sensitivity of the model training time of DeepLog and Logs2Graphs than OC-SVM.

Statistical analysis (see § 5.5.4) shows that sensitivity of the training time of the semi-supervised traditional and deep ML techniques across hyperparameter settings is not significantly different, with a *p-value* of 0.2.

The answer to RQ4 is that the time performance of semi-supervised traditional and deep ML techniques and the sensitivity of their model training time to hyperparameter tuning greatly vary across datasets. We therefore cannot draw general conclusions with that respect.

More in detail, OC-SVM shows a better performance in terms of i) model re-training time than DeepLog on Hadoop, F-dataset, Hades, BGL and Thunderbird and Logs2Graphs on HDFS, Hadoop, F-dataset, Hades and Thunderbird and ii) prediction time than DeepLog and Logs2Graphs on Hadoop, F-dataset, and Hades. DeepLog, however, is faster than OC-SVM and Logs2Graphs in terms of i) model re-training on HDFS and Spirit and ii) prediction time on HDFS and Thunderbird.

Further, the time performance of OC-SVM is less sensitive to hyperparameter tuning than that of DeepLog and Logs2Graphs on the session-based datasets (Hadoop and F-dataset) and all log message-based datasets (Hades, BGL, Thunderbird and Spirit) with large window sizes, ranging from 50 to 300. In contrast, the model training time of DeepLog is less sensitive to hyperparameter tuning than that of OC-SVM and Logs2Graphs on HDFS and all log message-based datasets on small window sizes, ranging from 10 to 20. Besides, OC-SVM is faster in terms of model re-training than DeepLog and Logs2Graphs on large window sizes, as the former technique is fed with fewer log event sequences. Logs2Graphs, however, is faster at predicting log anomalies than OC-SVM and DeepLog on small window sizes.

6.5 Discussion

6.5.1 Findings and Implications

In this empirical study, we have systematically evaluated and compared nine supervised and semi-supervised, traditional and deep ML techniques in terms of detection accuracy, time performance, and sensitivity of their i) detection accuracy and ii) time performance to hyperparameter tuning on seven benchmark datasets. Overall, the answer to the four research questions addressed in this study suggests that more attention should be given to traditional supervised ML techniques when it comes to the detection of execution path log anomalies before considering more complex, deep ML techniques.

Our findings indicate that supervised traditional and deep ML techniques i) perform very closely in terms of detection accuracy and prediction time on most of the benchmark datasets and ii) far outperform semi-supervised ML techniques. Further, supervised traditional ML techniques show less sensitivity to hyperparameter tuning compared to deep ML techniques. Despite the considerable emphasis, across the scientific literature, on deep learning-based ML techniques for addressing the LAD problem, our study shows that traditional ML techniques (notably RF) are competitive with deep learning ones on seven benchmark datasets w.r.t. four evaluation criteria that are relevant in practice. *RF is therefore our recommendation to detect execution path log anomalies from a practical standpoint.*

Moreover, our findings show that most of the supervised ML techniques yield their highest detection accuracy on small window sizes (10, 15 and 20). This is a rather useful insight as small window sizes enable the earlier detection of anomalies in practice. We also remark that deep semi-supervised ML techniques perform better on small window sizes across most of the log message-based datasets. However, this trend is not applicable to the traditional semi-supervised OC-SVM, which tends to perform better on larger window sizes (from 50 to 300) across most of the datasets. Overall, our findings suggest that for supervised ML techniques, smaller window sizes are recommended as they generate more training sequences, leading to higher detection accuracy due to the data-hungry nature of these models. For semi-supervised techniques, practitioners should adjust the window size based on the model: smaller window sizes work better for deep semi-supervised models, while larger window sizes are more effective for traditional methods like OC-SVM to achieve high detection accuracy.
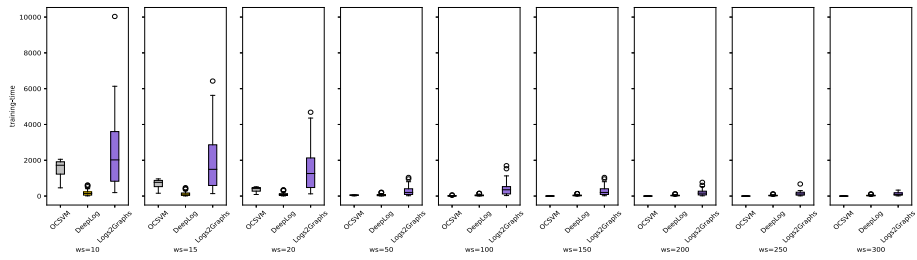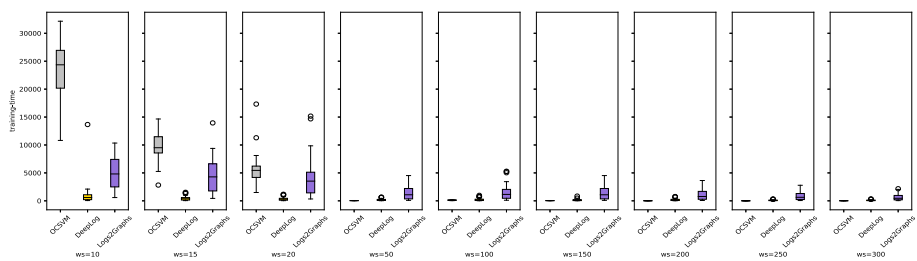
(a) Sensitivity on Hades dataset
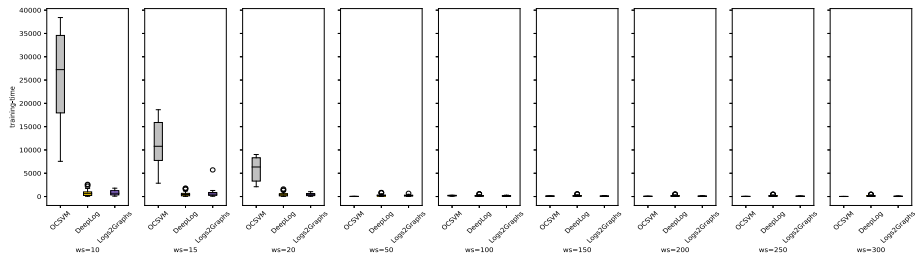
(b) Sensitivity on BGL dataset
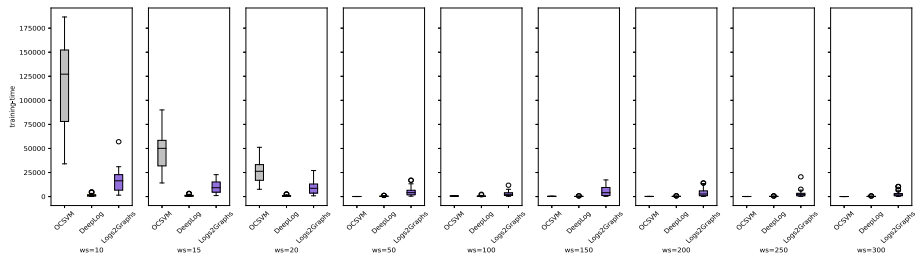
(c) Sensitivity on Thunderbird dataset

(d) Sensitivity on Spirit dataset

Fig. 12 Sensitivity of the time performance (in seconds) of semi-supervised traditional and deep ML techniques on log message-based datasets

Further, in terms of the best hyperparameter settings observed for the different ML techniques in our experiments, RF tends to achieve its highest detection accuracy with higher numbers of decision trees ($dTr \geq 50$) across all datasets, indicating that a large number of decision trees is generally required to guarantee a high detection accuracy. For SVM, the best value of hyperparameter C was 1000 across most of the datasets, indicating that a strong

regularization parameter is often required to handle the complexity of decision boundaries in log-based anomaly detection datasets. Further, the highest detection accuracy of LSTM is observed on the *adam* optimizer on six out of the seven datasets, except for BGL, on which the best optimizer was *rmsprop*. LogRobust showed its highest detection accuracy with the *rmsprop* optimizer across most datasets, except for BGL and Spirit, where *adam* was preferred, indicating that the choice of optimizer is dataset-dependent. For the number of epochs (epN), the optimal values ranged from 10 to 150, with most datasets requiring higher values ($epN \geq 100$), suggesting that a sufficient number of epochs is necessary for LogRobust to achieve an optimal detection accuracy across datasets. Similarly, the two versions of NeuralLog (NeuralLog1 and NeuralLog2) showed their highest detection accuracy with higher number of epochs (epN) within the range 100, 150 across most of the datasets, indicating that transformer-based techniques require a large number of epochs to effectively capture the intricate log patterns in log-based data to detect log anomalies. For semi-supervised ML techniques (OC-SVM, DeepLog, and Logs2Graphs), no consistent trends in best hyperparameter settings were observed across datasets. For instance, in OC-SVM, the hyperparameter ν varied greatly across datasets, with values such as 0.2, 0.1, and 0.9, reflecting a wide range of optimal regularization values depending on the dataset.

These findings may guide AIOps engineers in selecting the right ML technique, to find a trade-off between detection accuracy and time performance when addressing the LAD problem. The hyperparameter tuning conducted in this study allows AIOps engineers to assess the suitability of a specific ML technique to detect log anomalies for a specific context and dataset w.r.t. their overall detection accuracy, their time performance (model training time and prediction time) and their sensitivity to hyperparameter tuning. Moreover, AIOps engineers can prioritize the tuning of hyperparameters that have the most significant impact on the detection accuracy and time performance of the model of their choice, thus reducing the required time and computational resources.

6.5.2 Threats to Validity

Two types of threats to validity can affect the findings of our study.

Internal threats. We relied on publicly available implementations of DeepLog (wuyifan18 2020), LogRobust (Le and Zhang 2022), NeuralLog (LogIntelligence 2021) and Logs2Graphs (ZhongLIFR 2024). These third-party implementations might be faulty and could introduce bias in the results. To mitigate this, we carefully performed code reviews and used the replication package of the existing empirical study (Le and Zhang 2022). We remark that most of the results reported in LAD studies (Le and Zhang 2022; Liu et al. 2021; Qi et al. 2022; Du et al. 2021; Xie et al. 2022), for most of the ML techniques we used in our work (e.g., DeepLog, LogRobust, SVM, RF), are not reproducible, mostly because hyperparameter settings are not fully shared by these studies. We also note that most of the LAD studies do not share their code, making it more difficult for us to reproduce the same results. This has been also confirmed by Landauer et al. (2024), who studied the characteristics of common benchmark datasets and their impact on the effectiveness of existing ML techniques at detecting execution path log anomalies. The internal validity of our empirical study could also be threatened by the choice of specific window sizes; other window size values could lead to different results in terms of detection accuracy and time performance. To mitigate this, we considered various fixed window sizes, including the ones that have been adopted in existing studies (see Table 2).

Another threat is the choice of different hyperparameter settings for each ML technique, which we had to limit due to the high computational cost (notably the model training time)

of our experiments. To mitigate this, we motivated the choice of different hyperparameter settings for each ML technique based on the literature (§ 5.4.1). Different results in terms of detection accuracy and time performance could be obtained with different hyperparameter settings.

Further, in this paper, we have not considered ways to enhance the detection accuracy, such as improving data preprocessing. This omission could also impact the detection accuracy and generalizability of the results. While we acknowledge the impact of data preprocessing on the detection accuracy of ML techniques, recent empirical studies (Khan et al. 2024; Wu et al. 2023) suggest that certain preprocessing improvements, such as refining log parsing and log representation techniques, may not significantly enhance the detection accuracy for log-based anomaly detection techniques. For instance, Khan et al. (2024) found no strong correlation between log parsing accuracy and the anomaly detection accuracy, and Wu et al. (2023) showed that semantic-based log representations yielded similar detection accuracy across different techniques.

External threats. The selection of only three semi-supervised and five supervised traditional and deep ML techniques may limit the generalization of our findings. To mitigate this threat, we relied on commonly adopted and diverse supervised and semi-supervised, traditional (§ 2.3) and deep (§ 2.4) ML techniques, including RNN, Transformers and GNN-based learning models from recent studies.

7 Conclusion and Future Work

In this large empirical study, we assessed the anomaly detection accuracy and the time performance, in terms of model training and log anomaly prediction, of different semi-supervised and supervised, traditional and deep ML techniques. We further studied the sensitivity of detection accuracy and model training time, for each of these techniques, to hyperparameter tuning across datasets. This is of significant importance for practitioners as using techniques that are less sensitive reduces the effort entailed by applying them.

Our study shows that supervised traditional and deep ML techniques fare similarly in terms of detection accuracy and prediction time on most of the benchmark datasets. Further, as expected, supervised traditional ML techniques far outperform supervised deep learning ones in terms of re-training time. Among the former, Random Forest shows the least sensitivity to hyperparameter tuning regarding its detection accuracy and time performance.

Though they offer advantages when dealing with datasets containing few anomalies, semi-supervised techniques yield significantly worse detection accuracy than supervised ML techniques. The time performance and sensitivity to hyperparameter tuning of semi-supervised traditional and deep ML techniques widely vary across datasets.

The results of this study enable system and AIOps engineers to select the most accurate ML technique for detecting log anomalies, taking into account time performance which has significant practical implications. Though they need to be confirmed with further studies, our results are of practical importance because they suggest—when accounting for accuracy, training and prediction time, and sensitivity to hyperparameter tuning—that supervised, traditional techniques are a better option for log anomaly detection, with a preference for Random Forest. Given the emphasis on the use of deep learning in the research literature, this may come as a surprise.

As part of future directions, we plan to study the impact of different data distributions and model complexity on the detection accuracy and time performance of the different LAD

techniques. Further, detecting log anomalies is not sufficient for system engineers as it does not provide them with enough details about the cause(s) of anomalies. This warrants the design of solutions to facilitate the diagnosis of anomalies, which we will address as part of future work.

Author Contributions Shan Ali: Conceptualization, Investigation, Data Curation, Formal analysis, Software, Writing - original draft. Chaima Boufaied: Conceptualization, Investigation, Data Curation, Formal analysis, Writing - original draft. Domenico Bianculli: Conceptualization, Supervision, Writing - review & editing. Paula Branco: Conceptualization, Supervision, Writing - review & editing. Lionel Briand: Supervision, Writing - review & editing, Funding acquisition.

Data Availability The replication package accompanying this work is available at https://figshare.com/articles/software/LADEmpStudy/22756871?file=50577753. We make available i) the pre-processed datasets, as well as the corresponding pre-processing scripts; ii) the implementations of the different alternative traditional and deep ML techniques considered in this study; and iii) the detailed results.

Declarations

Conflicts of Interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Adeba JL, Kim D-H, Kwak J (2024) Sarlog: semantic-aware robust log anomaly detection via bert-augmented contrastive learning. IEEE Int Things J

Almodovar C, Sabrina F, Karimi S, Azad S (2023) Logfit: log anomaly detection using fine-tuned language models

Bengio Y (2012) Practical recommendations for gradient-based training of deep architectures. In: Neural networks: tricks of the trade: second edition. Springer, pp 437–478

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(2)

Bernard S, Heutte L, Adam S (2009) Influence of hyperparameters on random forest accuracy. In: Multiple classifier systems: 8th international workshop, MCS 2009, Reykjavik, Iceland, June 10-12, 2009. Proceedings 8. Springer, pp 171–180

Breiman L (2001) Random forests. Mach Learn 45:5–32

Catillo M, Pecchia A, Villano U (2022) Autolog: anomaly detection by deep autoencoding of system logs. Expert Syst Appl 191:116263

Chen J, Chong W, Yu S, Xu Z, Tan C, Chen N (2022) Tcn-based lightweight log anomaly detection in cloud-edge collaborative environment. In: 2022 Tenth international conference on advanced cloud and big data (CBD). IEEE, pp 13–18

Chen Z, Liu J, Gu W, Su Y, Lyu MR (2021) Experience report: Deep learning-based system log analysis for anomaly detection. arXiv:2107.05908

Chen M, Zheng AX, Lloyd J, Jordan MI, Brewer E (2004) Failure diagnosis using decision trees. In: International conference on autonomic computing, 2004. Proceedings. IEEE, pp 36–43

Cortes C, Vapnik V (1995) Support vector machine. Mach Learn 20(3):273–297

Cotroneo D, De Simone L, Liguori P, Natella R, Bidokhti N (2019) How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp 200–211

Creech G, Hu J (2013) Generation of a new ids test dataset: time to retire the kdd collection. In: 2013 IEEE wireless communications and networking conference (WCNC). IEEE, pp 4487–4492

Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805

Digital Research Alliance of Canada. https://alliancecan.ca/en

Du M, Li F, Zheng G, Srikumar V (2017) 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

Dunn OJ (1964) Multiple comparisons using rank sums. Technometrics 6(3):241–252

Du Q, Zhao L, Xu J, Han Y, Zhang S (2021) 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. Springer, pp 335–347

Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014) Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 15(1):3133–3181

Fix E, Hodges J (1989) Discriminatory analysis. Nonparametric discrimination: consistency properties. Int Stat Rev /Revue Internationale de Statistique 57(3):238–247

Genuer R, Poggi J-M, Tuleau C (2008) Random forests: some methodological insights. arXiv:0811.3619

Gong K, Luo S, Pan L, Zhang L, Zhang Y, Yu H (2024) Logeta: time-aware cross-system log-based anomaly detection with inter-class boundary optimization. Future Gener Comput Syst 157:16–28

Guo H, Yang J, Liu J, Bai J, Wang B, Li Z, Zheng T, Zhang B, Peng J, Tian Q (2024) Logformer: a pre-train and tuning pipeline for log anomaly detection. In: Proceedings of the AAAI conference on artificial intelligence, vol 38, pp 135–143

Guo H, Yuan S, Wu X (2021) Logbert: log anomaly detection via bert. In: 2021 International joint conference on neural networks (IJCNN). IEEE, pp 1–8

Han X, Cheng H, Xu D, Yuan S (2021) Interpretablesad: interpretable anomaly detection in sequential log data. In: 2021 IEEE international conference on big data (Big Data). IEEE, pp 1183–1192

Hancock JT, Khoshgoftaar TM, Johnson JM (2023) Evaluating classifier performance with highly imbalanced big data. J Big Data 10(1):42

Hashemi S, Mäntylä M (2021) Onelog: towards end-to-end training in software log anomaly detection. arXiv:2104.07324

He S, Zhu J, He P, Lyu MR (2020) Loghub: a large collection of system log datasets towards automated log analytics. arXiv:2008.06448

He P, Zhu J, Zheng Z, Lyu MR (2017) Drain: an online log parsing approach with fixed depth tree. In: 2017 IEEE international conference on web services (ICWS). IEEE, pp 33–40

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

Huang S, Liu Y, Fung C, He R, Zhao Y, Yang H, Luan Z (2020) Hitanomaly: hierarchical transformers for anomaly detection in system log. IEEE Trans Netw Serv Manag 17(4):2064–2076

Huang S, Liu Y, Fung C, Wang H, Yang H, Luan Z (2023) Improving log-based anomaly detection by pre-training hierarchical transformers. IEEE Trans Comput

Huo Y, Su Y, Lee C, Lyu MR (2023) Semparser: a semantic parser for log analytics. In: 2023 IEEE/ACM 45th international conference on software engineering (ICSE). IEEE, pp 881–893

Joulin A, Grave E, Bojanowski P, Douze M, Jégou H, Mikolov T (2016) Fasttext. Zip: compressing text classification models. arXiv:1612.03651

Khan ZA, Shin D, Bianculli D, Briand L (2022) Guidelines for assessing the accuracy of log message template identification techniques. In: Proceedings of the 44th international conference on software engineering, pp 1095–1106

Khan ZA, Shin D, Bianculli D, Briand LC (2024) Impact of log parsing on deep learning-based anomaly detection. Empir Softw Eng 29(6):139

Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. Ann Math Stat 462–466

Kruskal W (1952) Kruskal and wallis' test. J Am Stat Assoc 583–618

Landauer M, Onder S, Skopik F, Wurzenberger M (2023) Deep learning for anomaly detection in log data: a survey. Mach Learn Appl 12:100470

Landauer M, Skopik F, Wurzenberger M (2024) A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. Proc ACM Softw Eng 1(FSE):1354–1375

Le V-H, Zhang H (2021) Log-based anomaly detection without log parsing. In: 2021 36th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 492–504

Le V-H, Zhang H (2022) Log-based anomaly detection with deep learning: how far are we? https://github.com/LogIntelligence/LogADEmpirical

Le V-H, Zhang H (2022) Log-based anomaly detection with deep learning: How far are we? In: Proceedings of the 44th international conference on software engineering. ICSE '22, pp. 1356–1367. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3510003.3510155

Lee C, Yang T, Chen Z, Su Y, Yang Y, Lyu MR (2023) Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In: 2023 IEEE/ACM 45th international conference on software engineering (ICSE). IEEE, pp 1724–1736

Li X, Chen P, Jing L, He Z, Yu G (2022) Swisslog: robust anomaly detection and localization for interleaved unstructured logs. IEEE Trans Depend Secure Comput

Lin Y, Deng H, Li X (2024) Fastlogad: log anomaly detection with mask-guided pseudo anomaly generation and discrimination. arXiv:2404.08750

Lin Q, Zhang H, Lou J-G, Zhang Y, Chen X (2016) Log clustering based problem identification for online service systems. In: Proceedings of the 38th international conference on software engineering companion, pp 102–111

Li Z, Shi J, Van Leeuwen M (2024) Graph neural networks based log anomaly detection and explanation. In: Proceedings of the 2024 IEEE/ACM 46th international conference on software engineering: companion proceedings, pp 306–307

Liu X, Liu W, Di X, Li J, Cai B, Ren W, Yang H (2021) Lognads: network anomaly detection scheme based on log semantics representation. Future Gener Comput Syst 124:390–405

LogIntelligence (2021) Log-based Anomaly Detection Without Log Parsing, ASE, Research Track. https://github.com:LogIntelligence/NeuralLog (2021)

LOGPAI:Log Analytics Powered by AI. https://github.com/logpai

Lu S, Wei X, Li Y, Wang L (2018) 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). IEEE, pp 151–158

Meng W, Liu Y, Zhu Y, Zhang S, Pei D, Liu Y, Chen Y, Zhang R, Tao S, Sun P et al (2019) Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: IJCAI, vol 19, pp 4739–4745

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:1301.3781

Nguyen H-T, Nguyen L-V, Le V-H, Zhang H, Le M-T (2024) Efficient log-based anomaly detection with knowledge distillation. In: 2024 IEEE international conference on web services (ICWS). IEEE, pp 578–589

Okewu E, Adewole P, Sennaike O (2019) Experimental comparison of stochastic optimizers in deep learning. In: Computational science and its applications–ICCSA 2019: 19th international conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part V 19. Springer, pp 704–715

Oliner A, Stearley J (2007) What supercomputers say: a study of five system logs. In: 37th Annual IEEE/IFIP international conference on dependable systems and networks (DSN'07). IEEE, pp 575–584

Oshiro TM, Perez PS, Baranauskas JA (2012) How many trees in a random forest? In: Machine learning and data mining in pattern recognition: 8th international conference, MLDM 2012, Berlin, Germany, July 13-20, 2012. Proceedings 8. Springer, pp 154–168

Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1532–1543

Perin G, Picek S (2021) On the influence of optimizers in deep learning-based side-channel analysis. In: Selected areas in cryptography: 27th international conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27. Springer, pp 615–636

Probst P, Boulesteix A-L (2018) To tune or not to tune the number of trees in random forest. J Mach Learn Res 18(181):1–18

Probst P, Boulesteix A-L, Bischl B (2019) Tunability: importance of hyperparameters of machine learning algorithms. J Mach Learn Res 20(1):1934–1965

Probst P, Wright MN, Boulesteix A-L (2019) Hyperparameters and tuning strategies for random forest. Wiley Interdisc Rev Data Min Knowl Disc 9(3):1301

Python Software Foundation: Time module in Python. https://docs.python.org/3/library/time.html

Qi J, Luan Z, Huang S, Fung C, Yang H, Li H, Zhu D, Qian D (2023) Logencoder: log-based contrastive representation learning for anomaly detection. IEEE Trans Netw Serv Manag

Qi J, Luan Z, Huang S, Wang Y, Fung C, Yang H, Qian D (2022) Adanomaly: adaptive anomaly detection for system logs with adversarial learning. In: NOMS 2022-2022 IEEE/IFIP network operations and management symposium. IEEE, pp 1–5

Salton G, Buckley C (1988) Term-weighting approaches in automatic text retrieval. Inf Process Manag 24(5):513–523

Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443–1471

Wang Z, Tian J, Fang H, Chen L, Qin J (2022) Lightlog: a lightweight temporal convolutional network for log anomaly detection on the edge. Comput Netw 203:108616

Wang P, Zhang X, Cao Z, Xu W, Li W (2024) Loggt: cross-system log anomaly detection via heterogeneous graph feature and transfer learning. Expert Syst Appl 251:124082

Wu X, Li H, Khomh F (2023) On the effectiveness of log representation for log-based anomaly detection. Empir Softw Eng 28(6):137

wuyifan18 (2020) Pytorch Implementation of DeepLog. https://github.com/wuyifan18/DeepLog

Xia B, Bai Y, Yin J, Li Y, Xu J (2021) Loggan: a log-level generative adversarial network for anomaly detection using permutation event modeling. Inf Syst Front 23:285–298

Xiao R, Li W, Lu J, Jin S (2024) Contexlog: non-parsing log anomaly detection with all information preservation and enhanced contextual representation. IEEE Trans Netw Serv Manag

Xie Y, Ji L, Cheng X (2020) An attention-based gru network for anomaly detection from system logs. IEICE Trans Inf Syst 103(8):1916–1919

Xie Y, Zhang H, Babar MA (2022) Loggd: detecting anomalies from system logs with graph neural networks. In: 2022 IEEE 22nd international conference on software quality, reliability and security (QRS). IEEE, pp 299–310

Xu W, Huang L, Fox A, Patterson D, Jordan MI (2009) Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles, pp 117–132

Yang L, Chen J, Gao S, Gong Z, Zhang H, Kang Y, Li H (2024) Try with simpler-an evaluation of improved principal component analysis in log-based anomaly detection. ACM Trans Softw Eng Methodol 33(5):1–27

Yang L, Chen J, Wang Z, Wang W, Jiang J, Dong X, Zhang W (2021) Semi-supervised log-based anomaly detection via probabilistic label estimation. In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE). IEEE, pp 1448–1460

Yao J, Shepperd M (2021) The impact of using biased performance metrics on software defect prediction research. Inf Softw Technol 139:106664

Yin Z, Kong X, Yin C (2024) Semi-supervised log anomaly detection based on bidirectional temporal convolution network. Comput Secur 140:103808

Yu J, Kang S (2019) Clustering-based proxy measure for optimizing one-class classifiers. Pattern Recognit Lett 117:37–44

Yu B, Yao J, Fu Q, Zhong Z, Xie H, Wu Y, Ma Y, He P (2024) Deep learning or classical machine learning? an empirical study on log-based anomaly detection. In: Proceedings of the 46th IEEE/ACM international conference on software engineering

Zang R, Guo H, Yang J, Liu J, Li Z, Zheng T, Shi X, Zheng L, Zhang B (2024) Mlad: a unified model for multi-system log anomaly detection. arXiv:2401.07655

Zhang C, Wang X, Zhang H, Zhang J, Zhang H, Liu C, Han P (2023) Layerlog: log sequence anomaly detection based on hierarchical semantics. Appl Soft Comput 132:109860

Zhang M, Chen J, Liu J, Wang J, Shi R, Sheng H (2022) Logst: log semi-supervised anomaly detection based on sentence-bert. In: 2022 7th International conference on signal and image processing (ICSIP). IEEE, pp 356–361

Zhang X, Xu Y, Lin Q, Qiao B, Zhang H, Dang Y, Xie C, Yang X, Cheng Q, Li Z et al (2019) 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

ZhongLIFR (2024) Logs2Graph. https://github.com/ZhongLIFR/Logs2Graph

Zhu B, Li J, Gu R, Wang L (2020) An approach to cloud platform log anomaly detection based on natural language processing and lstm. In: 2020 3rd International conference on algorithms, computing and artificial intelligence, pp 1–7

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Shan Ali[1] (ID) **· Chaima Boufaied[2] · Domenico Bianculli[3] · Paula Branco[1] · Lionel Briand[1,4]**

✉ Shan Ali
  shan.ali@uottawa.ca

  Chaima Boufaied
  cboufaied@psu.edu.sa

  Domenico Bianculli
  domenico.bianculli@uni.lu

  Paula Branco
  pbranco@uottawa.ca

  Lionel Briand
  lbriand@uottawa.ca

[1] University of Ottawa, Ottawa, Canada

[2] Prince Sultan University, Riyadh, Saudi Arabia

[3] University of Luxembourg, Esch-sur-Alzette, Luxembourg

[4] Research Ireland Lero Centre, University of Limerick, Limerick, Ireland