Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

Anomaly detection in log-event sequences: A federated deep learning approach and open challenges

Patrick Himler [*], Max Landauer, Florian Skopik, Markus Wurzenberger

Austrian Institute of Technology, Giefinggasse 4, Vienna, 1220, Austria

A R T I C L E   I N F O

Keywords:
Log event sequences
Anomaly detection
Deep learning
LSTM
Federated learning

A B S T R A C T

Anomaly Detection (AD) is an important area to reliably detect malicious behavior and attacks on computer systems. Log data is a rich source of information about systems and thus provides a suitable input for AD. With the sheer amount of log data available today, for years Machine Learning (ML) and more recently Deep Learning (DL) have been applied to create models for AD. Especially when processing complex log data, DL has shown some promising results in recent research to spot anomalies. It is necessary to group these log lines into log-event sequences, to detect anomalous patterns that span over multiple log lines. This work uses a centralized approach using a Long Short-Term Memory (LSTM) model for AD as its basis which is one of the most important approaches to represent long-range temporal dependencies in log-event sequences of arbitrary length. Therefore, we use past information to predict whether future events are normal or anomalous. For the LSTM model we adapt a state of the art open source implementation called LogDeep. For the evaluation, we use a Hadoop Distributed File System (HDFS) data set, which is well studied in current research. In this paper we show that without padding, which is a commonly used preprocessing step that strongly influences the AD process and artificially improves detection results and thus accuracy in lab testing, it is not possible to achieve the same high quality of results shown in literature. With the large quantity of log data, issues arise with the transfer of log data to a central entity where model computation can be done. Federated Learning (FL) tries to overcome this problem, by learning local models simultaneously on edge devices and overcome biases due to a lack of heterogeneity in training data through exchange of model parameters and finally arrive at a converging global model. Processing log data locally takes privacy and legal concerns into account, which could improve coordination and collaboration between researchers, cyber security companies, etc., in the future. Currently, there are only few scientific publications on log-based AD which use FL. Implementing FL gives the advantage of converging models even if the log data are heterogeneously distributed among participants as our results show. Furthermore, by varying individual LSTM model parameters, the results can be greatly improved. Further scientific research will be necessary to optimize FL approaches.

1. Introduction

AD has been an actively researched field for decades in various domains (Chandola, Banerjee, & Kumar, 2009). The goal of AD is to find events and event sequences that deviate from normal behavior as efficiently and timely as possible (Chalapathy & Chawla, 2019). In this paper, we investigate the detection of anomalies in the cyber security domain using log data analysis. In recent years, cyber attacks have become more sophisticated and the attack surface has tremendously increased because of nowadays complex computer systems and networks. AD is used in Intrusion Detection Systems (IDS) to automatically detect and classify intrusions, attacks, or violations of security policies in infrastructures at network-level and host-level (Vinayakumar et al., 2019). Conventional signature-based IDS, using patterns of already known attacks and malicious behavior, have become insufficient. The need for more adaptability, to enable detection of unknown attacks such as zero-day exploits, is immanent. As a result, we see a shift towards anomaly based IDS, which use various data sources like textual

[*] Corresponding author.
E-mail addresses: patrick.himler@ait.ac.at (P. Himler), max.landauer@ait.ac.at (M. Landauer), florian.skopik@ait.ac.at (F. Skopik), markus.wurzenberger@ait.ac.at (M. Wurzenberger).

https://doi.org/10.1016/j.mlwa.2024.100554
Received 22 December 2023; Received in revised form 4 April 2024; Accepted 26 April 2024
Available online 27 April 2024

log data and network traffic data to reliably discover deviations from a desired system behavior. AD supports reaching the goals of minimal maintenance, human interaction, and delay in response time (Wurzenberger, Skopik, Settanni, & Fiedler, 2018). Currently, especially DL, a branch of highly performant ML algorithms, is a hot spot in AD research. DL aims to discover the essential differences between normal and abnormal data with high accuracy (Liu & Lang, 2019). DL supports security experts to react quickly and effectively to known and unknown attacks, and to gain a broad overview of the threat landscape.

However it soon became apparent that DL for AD approaches also incorporate limitations. For example, the transfer of sensitive data over the Internet to a central node raises privacy concerns (Rahman, Tout, Talhi, & Mourad, 2020). One promising solution to overcome those limitations is a distributed paradigm called FL. FL enables model training directly on user devices or local servers without sharing raw data with a central server. Instead, only model updates or aggregated gradients are exchanged, ensuring privacy is maintained. This decentralized approach minimizes the risk of data breaches or privacy violations compared to conventional centralized methods. FL can leverage the vast amount of distributed data available across different devices or locations, making it highly scalable. In AD, this distributed nature allows the algorithms to learn from diverse data sources, capturing a broader spectrum of anomalies and improving its generalization ability. Furthermore, FL can tolerate device failures or communication issues without effecting the overall training process, enhancing robustness in real-world deployment scenarios. Conventional AD models often require large volumes of data to be transferred to a central server for training, leading to high communication costs, especially in resource-constrained environments or low-bandwidth networks. FL alleviates this burden by performing model training locally on each device or edge server, with only model updates being transmitted. This reduces communication overhead and minimizes latency, making FL suitable for real-time AD applications (McMahan, Moore, Ramage, Hampson, & y Arcas, 2017). AD systems must adapt to evolving data distributions and emerging anomalies over time. FL supports continuous learning by allowing models to be updated incrementally on distributed data sources without the need for centralized retraining. This enables AD systems to quickly adapt to changing environments and maintain high detection accuracy over time.

However, despite its promising benefits, FL also faces several challenges and there remain several research gaps. While FL addresses privacy concerns by keeping raw data decentralized, it still introduces security and privacy risks, such as model poisoning attacks. Protecting FL systems against such adversarial threats requires robust security mechanisms and privacy-preserving techniques, which may add complexity to the overall system design. Although FL reduces communication overhead compared to centralized approaches, the coordination and synchronization of model updates across a large number of devices or clients can still incur non-negligible communication costs, especially in scenarios with high device churn rates or unreliable network connections. Efficient strategies for minimizing communication overhead while ensuring convergence and model consistency remain areas of ongoing research in FL.

It is also useful to utilize log data for training DL models. Log data provide more detailed insights for behavioral understanding of a system than network traffic data, because not only what a systems communicates on the network is analyzed, but also internal processes. Contrary to many available state of the art AD systems, which process network traffic data, a paradigm shift towards log data should be considered, because log data is generally available in unencrypted form and contains low-level traces of system activities (Wurzenberger et al., 2018). Most of the time, working with log data requires preprocessing steps, where the textual log data are parsed into numeric formats. After parsing, a DL model can process log data and classify them into normal and anomalous log events (He, Zhu, He, & Lyu, 2016). To detect not only individual anomalous log lines, i.e. point anomalies, but also contextual anomalies, several log lines are combined into log-event sequences. If parts of the data set are anomalies only in a certain context but not isolated from that context, we speak of contextual anomalies. To detect contextual anomalies, we need to consider both contextual attributes, like passed time between data instances and behavioral attributes such as occurrences of other instances before or after (Song, Wu, Jermaine, & Ranka, 2007). The predestined DL algorithm to achieve this is LSTM because it uses past information to predict whether future events are normal or anomalous. LSTM is a type of Recurrent Neural Network (RNN) architecture that is particularly effective for sequence prediction tasks, including time series data analysis. The authors of Vinayakumar, Soman, and Poornachandran (2017) claim that LSTM as a variant of RNN+ algorithm is one of the most important approaches to represent long-range temporal dependencies in log sequences of arbitrary length. For AD, this means that we use past information to predict whether future events are benign or anomalous. This past information is especially important when we want to detect contextual and group anomalies.

Besides highlighting limitations when analyzing log-event sequences with state of the art DL approaches, which have been partially addressed in Himler, Landauer, Skopik, and Wurzenberger (2023) this work focuses on comparing the FL approach with the centralized solution. The contributions are as follows:

- A critical discussion of state of the art DL approaches.
- Experiments with LogDeep utilizing the HDFS data set to confirm results from Du, Li, Zheng, and Srikumar (2017) and provide a basis for comparison with the FL implementation.
- Implementation of LogDeep in the flower framework[1] and experiments with evenly and unevenly split HDFS data sets.
- A discussion of current limitations and outlook for future research topics in this area.

The remainder of the paper is structured as follows: Section 2 gives an overview of state of the art approaches for log-based AD using DL models. Furthermore, we explain FL and its basic features. Next, Section 3 introduces our chosen approach describing all necessary steps from raw log data to AD. Section 4 evaluates our approach and compares it with state of the art solution (Du et al., 2017). Important limitations are critically discussed and open research challenges are listed in Section 5. Finally, Section 6 concludes this paper.

2. Background and related work

DL shows good performance and flexibility, especially when data sets become larger and the structure of data becomes more complex (Chalapathy & Chawla, 2019). The complexity lies in the fact that anomalies often show clear abnormal characteristics in low dimensional space that are virtually not noticeable in higher dimensional space. The choice of the DL architecture depends usually on the type of input data.

2.1. Long Short-Term Memory

LSTM are good for processing sequential data (Chalapathy & Chawla, 2019) and can be classified among others into supervised, semi-supervised and unsupervised based on the need for data labels during training. Labels indicate whether a respective data instance is normal or anomalous. The differences between the individual approaches are as follows (Chandola et al., 2009; Landauer, Onder, Skopik & and Wurzenberger, 2023; Villa-Pérez, Alvarez-Carmona, Loyola-Gonzalez, Medina-Pérez, Velazco-Rossell, & Choo, 2021):

[1] https://flower.dev/

- Supervised: A fully labeled training set containing both normal and anomalous data is required. A common approach is to build a predictive model for both data classes. Then unseen data instances are tested with the model to determine which class they belong to. At first glance, it is easy to create a model like this, but it has two major disadvantages: First, usually data sets contain fewer anomalies than normal data which leads to an imbalanced class distribution. Second, it is not trivial and thus challenging to label the anomaly class.
- Semi-supervised: The prerequisite is that the training set only contains normal data. After training with a portion of normal data, the resulting model is corresponding to normal expected behavior and can then be used to identify anomalies in the test data set. This technique assumes availability of normal training data sets which can be challenging in some application domains.
- Unsupervised: The system learns independently to distinguish between normal and anomalous data without the prerequisite of a labeled training set. This approach makes use of the intrinsic properties of data instances. In principle, it is often the case that investigated data sets have fewer anomalies than normal data. The trained model should be robust against those few anomalies. If this assumption does not apply, it will lead to a high false alarm rate.

We have chosen to look into semi-supervised approaches for this paper, because they reflect a realistic scenario with regard to real world applications to first learn a model in an anomaly-free area with normal data and then switch to live operation. This is also reflected in the assumption that anomalies are predominantly rare and sometimes difficult to capture as opposed to normal data (Villa-Pérez et al., 2021). In recent years, many DL models have been proposed to analyze log data and detect anomalies (Farzad & Gulliver, 2020; Nedelkoski, Bogatinovski, Acker, Cardoso, & Kao, 2020; Yang et al., 2021). We conduct a survey of state of the art approaches and narrow down the results by filtering by the number of citations (more than 300), the year of publication (not older than 5 years) and the data sets used. After filtering, DeepLog, LogAnomaly and LogRobust described in the following sections stood out.

2.1.1. *DeepLog*

The authors of Du et al. (2017) have developed an approach called DeepLog that uses an LSTM as DL model that processes log lines in sequences. The model thereby learns log patterns during normal execution and detects anomalies when log patterns deviate from the trained model. Even though this approach was published in 2017, it still has great significance to this day and is frequently used as a benchmark in scientific publications. As a starting point of development, the authors describe that log entries in most cases have fixed patterns and also follow grammar rules. But they also state that it is still difficult to make a generalization about interesting features for different data sets. To evaluate their implementation, the authors use the HDFS data set (Xu, Huang, Fox, Patterson, & Jordan, 2009) and the OpenStack data set (Du et al., 2017). The HDFS data set consists of log lines from a primary data storage system. The OpenStack data set contains administrative logs of virtual machine instances. DeepLog works in a semi-supervised way. Therefore only anomaly-free sequences are used for training the LSTM model. First, the Spell (Du & Li, 2016) parser extracts so-called log keys (=constant part) and parameter values (=variable part) from each log line. This way two separate AD systems can be set up. First, DeepLog verifies if the log key to be examined is a known one. In case it is, it checks if parameter values indicate anomalies. The sequence in which log keys occur can be used to detect so-called execution path anomalies. This type of anomalies corresponds to the contextual anomalies explained above. The LSTM model uses a set of log keys with a fixed size, called window size, and leverages the gathered knowledge to predict which log key should follow. After training the LSTM model, DeepLog outputs possible candidates that were predicted with their respective probabilities. In contrast to this, parameter values are used to find irregularities in log lines with the same log keys. A matrix is built up where each column corresponds to a log key and the corresponding parameter values are entered in the rows. For the evaluation of the matrix, an LSTM model can also be used. The individual parameter values are used as input in the order in which they occur and an attempt is made to generate a prediction for the following parameter value based on this existing history. The structure of the DeepLog architecture is shown in Fig. 1.

The two most important input parameters DeepLog needs are length of the window under consideration and number of top candidates for prediction. The choice of these parameters depends on the problem at hand. For example, if one chooses a too large window size, DeepLog provides a better overall picture resulting form a wide view in the past, which leads to performance losses and long training time. The choice for the number of candidates results in a trade off between detection rate and false alarm rate.

2.1.2. *LogAnomaly*

LogAnomaly follows a similar approach as DeepLog. The authors of Meng et al. (2019) claim that if one just looks at log keys rigidly, one receives many false alarms, because log line structures and dependencies can be highly complex. Therefore, they analyze not only log keys but also semantics of the logs. For this, they use a method they call template2vec. This method extracts semantics including synonyms and antonyms. They use LSTM as DL model to predict consecutive logs and HDFS as data set for evaluation. In addition, they also consider the Blue Gene/L (BGL) data set (Oliner & Stearley, 2007), which consists of logs from a supercomputer and was manually labeled. The authors designate LogAnomaly as an unsupervised approach, but they use labels of the data sets as ground truth for evaluation. According to the experiments performed, LogAnomaly performs better than DeepLog based on the evaluation metrics for the data sets investigated. However, a case study in that paper shows that DeepLog triggered an alarm faster than LogAnomaly in a single scenario.

2.1.3. *LogRobust*

In Zhang et al. (2019) the authors draw attention to instability of log data. Instability is caused by the evolution of the logging process itself and processing noise in log data. Evolution is based on constant development of software and associated changes in source code and logging statements. The introduction of noise already happens during data collection. But logs can also be misinterpreted during parsing, which both reduces the accuracy of an AD system. To counteract these adverse influences, the authors of LogRobust rely on semantic vectors. Like LogAnomaly, semantic properties of log lines are extracted, but in contrast to previously discussed approaches, LogRobust can also process new log lines if they are similar to known ones. To test this, slightly modified log lines are inserted in the HDFS data set to be examined. It is important to note that training of the model is still performed with unmodified log lines, i.e., the original HDFS training data set. LogRobust also uses log-event sequences as input for a LSTM, but it is a supervised approach and uses 6000 normal log-event sequences and 6000 malicious log-event sequences which are chosen randomly from the HDFS data set to train the model. As expected due to the supervised approach metrics after training are better, for testing without modified log lines, than for the other two approaches shown above. Further evaluation shows that LogRobust still achieves good metrics even with high injection rates of modified log lines in the test data. The authors have not published the source code for their work, but re-implementations are available.

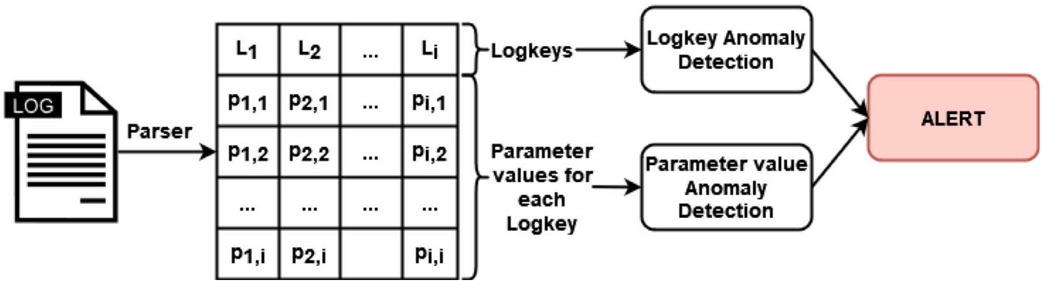

Fig. 1. DeepLog architecture based on Du et al. (2017).

2.2. Federated Learning

DL nowadays benefits from the sheer volume of data. However, with this large quantity there arise issues with transferring this data to a central server. FL tries to avoid this problem by having the training data never leave local devices. A shared DL model is learned through locally aggregated updates. Because the DL model is learned on local devices, it also reacts better to environmental changes over time (Ito, Tsukada, & Matsutani, 2021). The localization of data also takes privacy and legal concerns into account. This could make it feasible in the future that there is even a secure data exchange between organizations that currently do not cooperate with each other. Coordination and collaboration will become very important in the future, especially in the cyber security domain. FL was first introduced in 2017, by researchers of Google (McMahan et al., 2017). The term FL origins from the fact that the model learning task is performed by a loose federation of devices called clients which are coordinated by a central server. Each client trains a local DL model only with a local training data set and sends updates for a global model to a central server. It must be mentioned that the central server that manages the training and distribution of the global model must be trusted. Another reason why it is interesting to deal with FL is to reduce communication costs. Nowadays, devices like smartphones have fast processors and Graphics Processing Units (GPU). Assuming that local data sets are small, the described hardware can simply take over computation of DL models instead of many devices taking up bandwidth unnecessarily by sending data to a central server, thus reducing communication costs (McMahan et al., 2017). A final statement on the saving of computational resources is difficult, because it depends strongly on the other load of a client and which energy resource, e.g., battery, is available. Basically, FL consists of the following steps (Li, Fan, & Lin, 2020):

1. All participating clients get a generic global model from a central server for local training.
2. Each client learns a local DL model with local data.
3. After the training phase, clients send back their local parameter updates to the central server for aggregation.
4. The central server averages those updates and sends back an updated version of the global model.
5. Those processes are repeated until desired performance with respect to chosen metrics is achieved. One iteration of those processes is called a round.

Fig. 2 depicts the process.

The authors of Yang, Liu, Chen, and Tong (2019) proposed the following categorization of FL approaches based on how data is distributed among participating clients in feature and sample space, where a sample is a single entry in a data set and a feature is a measurable value of such entry (Bishop, 2006).

- Horizontal FL: The feature space is the same but the sample space is different. An example would be two regional banks which have different customers (= samples) but the business transactions (= features) are very similar.
- Vertical FL: Unlike horizontal FL, it is the other way around, i.e. in both data sets there are identical samples that can be recognized by an identifier but have different features. For example, two different companies have the same customers (= samples) with the same residential address, but the companies provide different services (= features) to these customers.
- Federated transfer learning: Data sets on the clients differ in both sample and feature space. There is only a small overlap, e.g., in the feature space, which allows conclusions to be drawn about the entire sample- and feature space. This approach is still at the beginning of a development but is mentioned here for the sake of completeness.

The logs of the data sets, which we examine in this work, are recorded centrally. We use the horizontal approach, because the logs have the same feature space and are divided into parts individually processed by a number of clients for experiments. So each client on its own can make a local AD system and collaboratively learn from others simultaneously. Specifically, local biases that arise due to lack of heterogeneity in the training data can be overcome (Lavaur, Pahl, Busnel, & Autrel, 2022). But existing FL approaches for AD are still missing a uniform structure and are far from complete, especially when utilizing log data.

2.2.1. Federated Learning for Anomaly Detection

The authors of Guo, Wu, Zhu, Yang and Han (2021) published one of the few papers that apply FL in the context of AD to log data. A central point of this research is that it deals with the issue of transmission of updates between clients and servers. It has been found that attackers can intercept communicated gradient updates and thus violate the privacy protection of local data. Therefore, it is advantageous if this communication is encrypted. However, encryption always leads to a computational overhead which is proportional to the size of gradient updates. The authors therefore propose a lightweight FL method for AD called FLOGCNN. The lightweight model is achieved due to model parameter reduction. As a result the AD model uses one-dimensional convolution with very few parameters. For the evaluation of the presented approach the authors use the HDFS data set and Thunderbird data set (Oliner & Stearley, 2007). Thunderbird is an publically available data set of logs collected from a Thunderbird supercomputer system at Sandia National Labs in Albuquerque. The data set distinguishes between alert and non-alert messages. This is a labeled data set which can also be used for supervised approaches. For FLOGCNN two roles are defined. A server aggregates gradient updates and distributed participants, which are here called log owner. The server randomly selects a number of log owners and sends initial model parameters to them. Log owners learn independently and send updates back to the server. For evaluation purposes, the authors compare their implementation with a centralized LogRobust implementation. For both implementations the source code is not publicly available. The author's implementation achieves slightly worse results for the metrics than LogRobust. As a positive result the authors state a lower training time. A limitation of this approach is that it only shows how to deal with

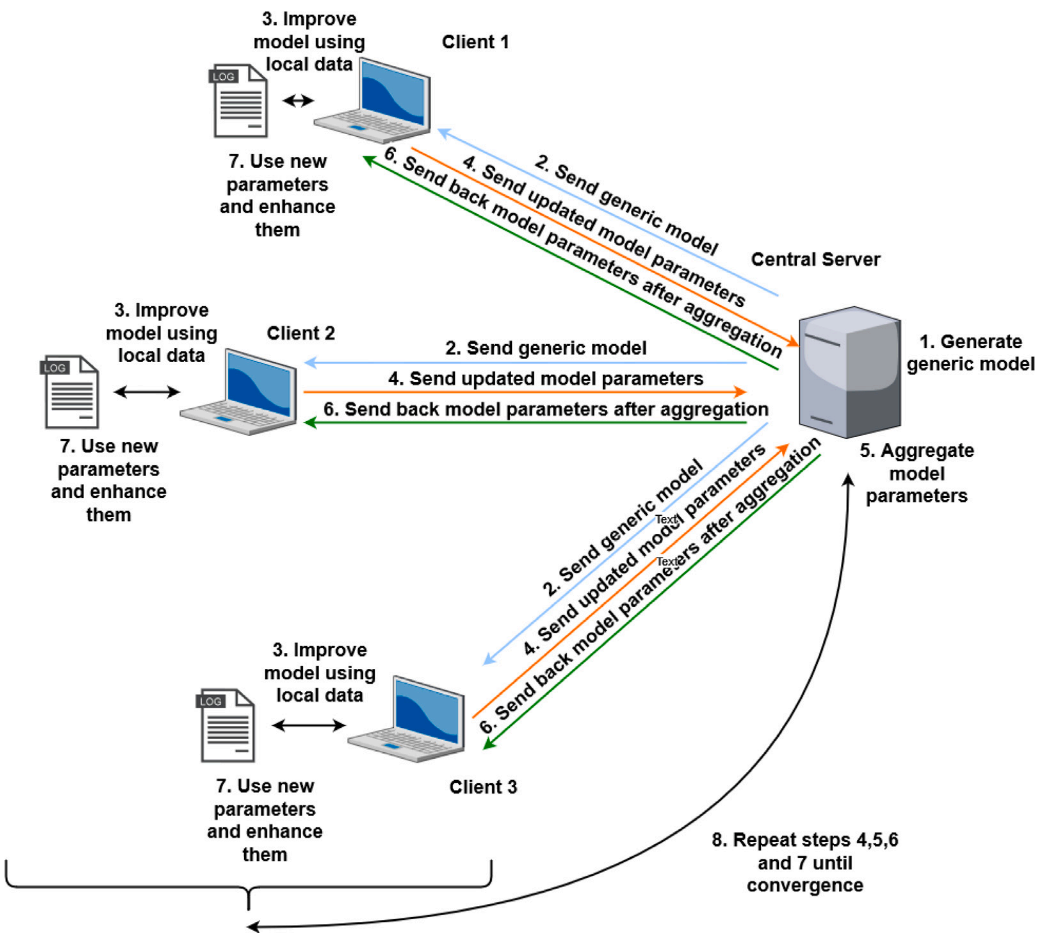
3. Improve model using local data
Client 1
7. Use new parameters and enhance them
2. Send generic model
4. Send updated model parameters
6. Send back model parameters after aggregation
Central Server
1. Generate generic model
3. Improve model using local data
Client 2
7. Use new parameters and enhance them
5. Aggregate model parameters
3. Improve model using local data
Client 3
7. Use new parameters and enhance them
8. Repeat steps 4,5,6 and 7 until convergence
LOG

Fig. 2. Illustrative FL scenario based on Rahman et al. (2020).

HDFS and Thunderbird logs. These are data sets with few features and little variation. Therefore, feature reduction is easy. It is questionable whether the presented architecture will perform as well, when other log data sets are used.

3. Approach for Anomaly Detection with log-event sequences

In this section we provide an introduction to the data sets used and present our approach for AD analyzing log-event sequences. The preprocessing of log lines is an important step to prepare data as input for DL networks. DeepLog which was described in the previous chapter is adapted for this work. We decided to first implement DeepLog in a centralized way and verify it with existing results. This is followed by a decentralized FL implementation and here we also explain our chosen FL aggregation algorithm.

3.1. HDFS data set

The HDFS is a file system designed for storing large files, batch processing, and to run on commodity hardware. The data set was generated 2009, in a private cloud environment (Amazon's Elastic Compute Cloud) using benchmark workloads and is described in detail in Xu et al. (2009). It consists of 11.2 million system log entries. It was manually labeled by Hadoop domain experts, to identify anomalies, where the anomalies describe incorrect execution paths. 2.9% of all system log entries were labeled as anomalous by those experts. The raw HDFS logs are semi-structured and consist of a header- and a content part. The log data are sliced into sequences according to block_ID's. Then each trace associated with a specific block_ID is assigned a ground truth label: normal/anomaly. In this paper we use the HDFS data set to test our adapted LogDeep implementation and to compare it with published results of Du et al. (2017).

3.2. Centralized LogDeep adaption

A detailed search of codebases such as Github and Gitlab revealed LogDeep (Donglee-Afar, 2020) to be the most promising re-implementation, based on ratings, year of release, detailed documentation and ease of adaptation. LogDeep combines the work of Du et al. (2017), Zhang et al. (2019) and Meng et al. (2019) to a framework for AD utilizing log data and has already been adapted in other scientific works, such as Guo, Yuan, and Wu (2021). Because of the scientific relevance, we focused on the DeepLog part from this implementation. As previously described, DeepLog uses LSTM as DL model. The main adjustable parameters in the LogDeep implementation are:

- L: Number of layers of LSTM.
- α: Number of memory units in one LSTM block.
- Window size: Length of window under consideration.
- Candidates: Number of candidates that were predicted with their respective probabilities.

After the parameters have been set, LogDeep reads the training log-event sequences. The log-event sequences must have at least the length of the specified window size. In principle, training of the DL model and AD, that uses the trained model, are separate processes. The reason for this is that after the training, the model is stored and can be transferred somewhere else in order to perform AD there as well. The Adam optimizer is used for the LSTM. This optimizer is often used for tasks where sparse gradients are to be expected. The advantages of this optimizer are increased computational performance and low memory requirements (Kingma & Ba, 2014). The LSTM delivers candidates with a certain probability which log key is expected next in the log-event sequences. Therefore, we use the cross entropy as loss function here, which quantifies the difference between probability distributions. After training, LogDeep can be used for AD. For this purpose, normal and anomalous test data are read in separately. The model calculates which log key comes next for each individual window and compares whether this matches the log keys in the test data. If the following log key is not contained in the proposed candidates, the whole log-event sequence is marked as anomaly.

3.2.1. *Padding*

While analyzing the source code of LogDeep, we came across an interesting fact. If we take a closer look at the HDFS data set, we see that train and test normal log-event sequences consist of at least 10 consecutive log keys. However, in the test abnormal log-event sequences, also shorter sequences occur. This results in a problem for the window size, because the LSTM determines how far back it stores past information. In order to be able to analyze these short sequences with the model, so-called padding is applied. In this case, log-event sequences are filled with a log key that not occurred in the data, in order to achieve desired window size. This log key must be unique and must not have occurred before, otherwise the data set would be corrupted. Based on this discovery, we conducted another experiment to show what effect padding has on LogDeep. Section 4.1 presents the related results. If padding is omitted, LogDeep cannot process and discards all test abnormal log-event sequences in the AD phase, which number of log keys is less than the window size. The size of the training and test normal data set remains the same, only the size of the test abnormal data set is reduced.

3.3. *Federated Learning LogDeep adaption*

The FL implementation is based on the flower framework which was released in July 2020 (Beutel et al., 2020). This framework is a scaleable open source framework, in terms of number of clients. With this it is very convenient to implement existing DL setups in a federated setting and evaluate their convergence- and training time. One of the main advantages of flower is that it is programming language- and ML framework-agnostic by design. The flower core framework, as common in FL, consists of a server and client part. The flower server is further divided into three main components: Client Manager, FL loop and Strategy. The client manager is responsible for communication between connected clients. The strategy describes the aggregation algorithm of updates. The algorithm we have chosen will be discussed below in Section 3.4. The FL loop works as a central entity. It queries strategies, receives updates from connected clients, computes global models and returns computed models to clients via client manager. The client side only waits for instructions from the server and executes them. The client manager works with so called flower messages. These messages are based on Remote Procedure Call (RPC) streams. RPC is efficient for low bandwidth transmission, because of the binary serialization format. So the client manager works as a RPC server for sending and receiving flower messages (Beutel et al., 2020). Fig. 3 depicts the flower architecture.

At the start of the FL process, the server passes model parameters with the fit method to all the clients. All the clients receive parameters for their local model and start the first training phase. After completion of the first training phase, all clients send their current local model parameters as an update to the server with the method get_parameters and evaluate the local model with the normal and abnormal test data. The server aggregates all received updates and sends them back again. After that, all clients have the updated model parameters and the test data is evaluated again with them in the so-called evaluation phase. With this same level of knowledge, the clients start the next round and try again to improve the model with their local training data. The process described here can be repeated as often as desired and is shown for 2 rounds in Fig. 4.

Table 1
Specifications of the laptop and the individual OpenStack instances.

	OpenStack (central)	OpenStack (FL)
Processor	8 VCPU	2 VCPU
RAM	16 GB	4 GB
OS	Ubuntu 20.04	Ubuntu 20.04

Table 2
Splitting of HDFS data set in train- and test sequences.

Train sequences	Test normal sequences	Test abnormal sequences
4855	553,366	16,838

3.4. *Federated Averaging*

The Federated Averaging (FedAvg) aggregation algorithm assumes K participating clients, where each client has a number of n_k log lines as training data and a local model weight w_{t+1}^k. For the $(t+1)$-th round aggregation is given by Eq. (1).

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{1}$$

where the global model weight update is denoted with w_{t+1}. n indicates total number of log lines at K clients (Li, Cheng, Liu, Wang, & Chen, 2019). Besides FedAvg, there exist other more complex aggregation algorithms, which for example select only a part of total available clients K and which may differ from round to round. A listing and description of further aggregation algorithms, for example Federated Stochastic Gradient Descent (FedSGD) can be found in Lavaur et al. (2022). In this work we only consider FedAvg for simplicity. Moreover, aggregating the gradient at each epoch would be more costly in terms of bandwidth consumption and computing power.

4. Evaluation

The re-implementation of DeepLog called LogDeep (Donglee-Afar, 2020) was used as baseline for all further experiments. In the first step the code base was analyzed. With the original code base only the HDFS data set can be processed. In our implementation the parameters for the selected data set can be set and padding, which is described in Section 3.2.1, can be activated or deactivated. The following requirements for the current implementation apply: python >= 3.6 and pytorch >= 1.1.0. Basically, DL requires a large amount of processor power and Random Access Memory (RAM). However, nowadays, more and more edge devices have the required resources and researchers work on improving the resource-hungry requirements of DL. The experiments were performed on OpenStack instances. The specifications of the individual OpenStack instances can be found in Table 1. A central OpenStack instance was used to confirm results from Du et al. (2017). The second type of OpenStack instance describes the specification of FL clients and servers.

4.1. *Centralized LogDeep with HDFS data set*

First, we verified that the evaluation metrics of LogDeep correspond to those published in Du et al. (2017). For this purpose we set the window size to 10. The hidden size, which defines the number of hidden states, was set to 64. Further, the number of layers was set to 2, number of candidates to 9, batch size to 2048, and number of epochs to 300. These values were also taken from Du et al. (2017). As a basis for evaluation we used the freely available HDFS data set in parsed form and preselected splitting of train and test data (LogPAI, 2021). This way our implementation can be compared with other scientific publications and errors caused by parsing can be omitted. The number of log keys is 29 and the exact breakdown of the data set can be taken from Table 2.

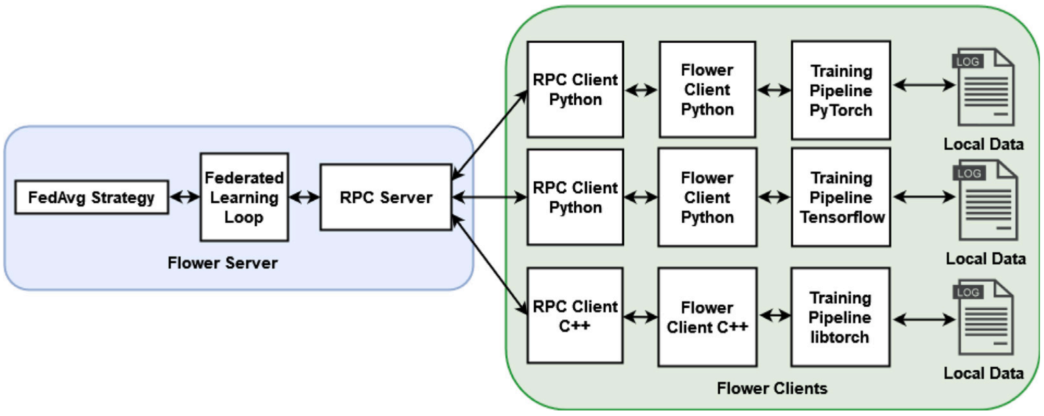

Fig. 3. Flower architecture based on Beutel et al. (2020).

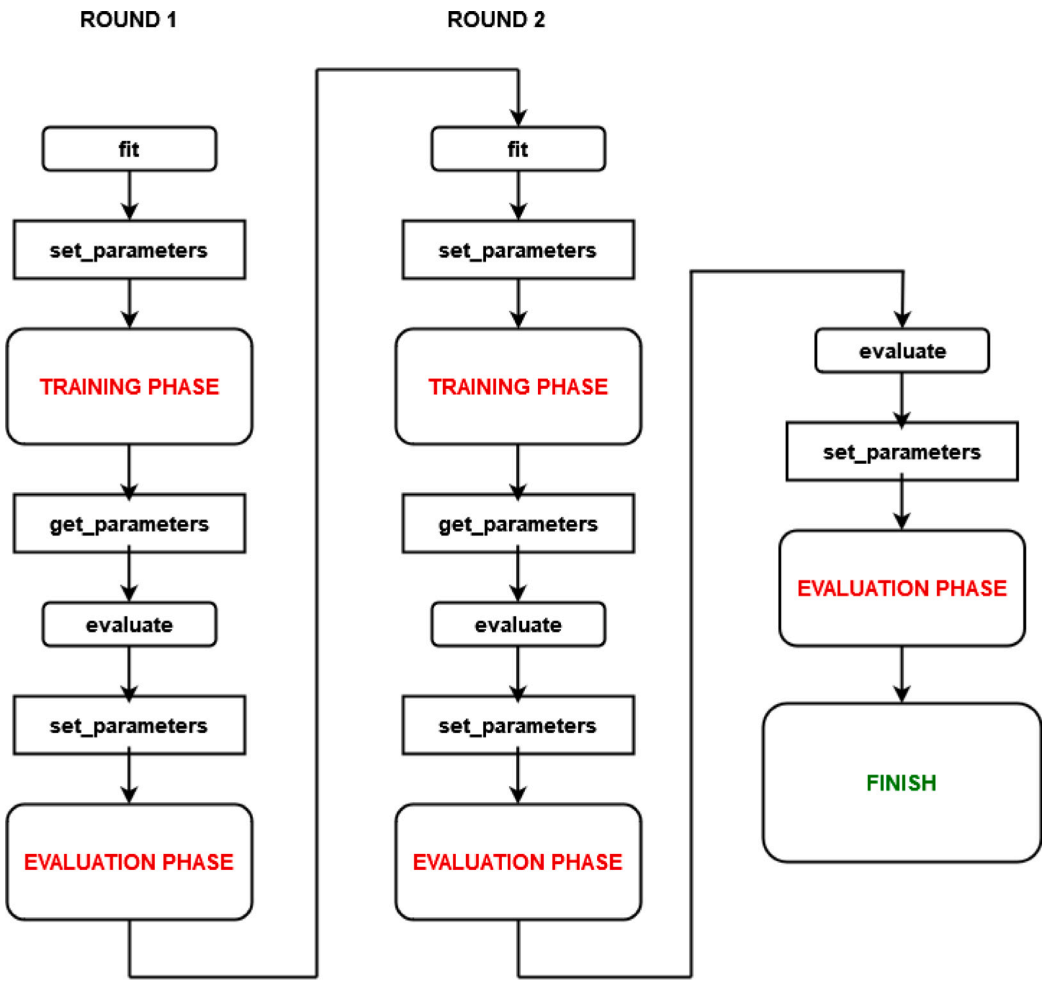

Fig. 4. Flowchart of flower framework.

Table 3
Comparison between DeepLog and LogDeep.

	Accuracy	Precision	Recall	F1-Score
DeepLog (Du et al., 2017)	–	95%	96%	96%
LogDeep (Donglee-Afar, 2020)	99.76%	95.63%	96.35%	95.99%

The Results show that the re-implementation achieves approximately the same metrics as in Du et al. (2017). The comparison is listed in Table 3.

The effects of padding can be analyzed using the HDFS data set. Since only the test abnormal sequences contain short log-event sequences smaller than the window size, padding is only applied there.

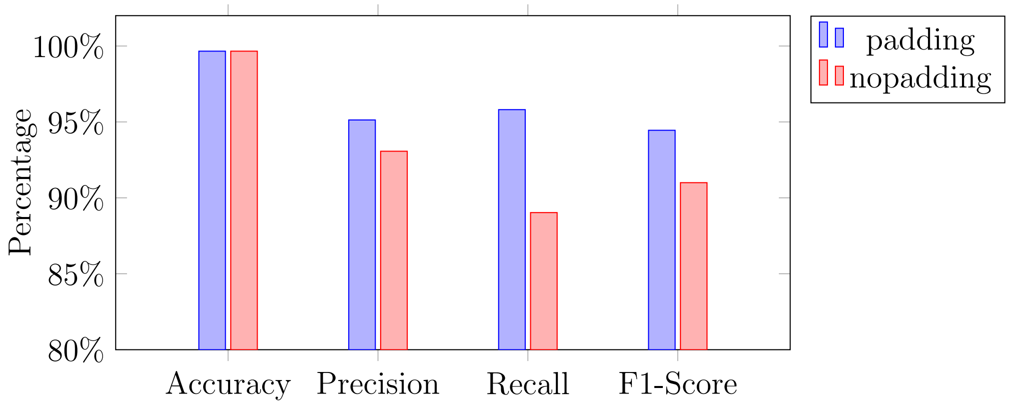

Fig. 5. Comparison between padding and nopadding for LogDeep with HDFS log-event sequences.

Table 4
Comparison between padding and nopadding for LogDeep with HDFS log-event sequences.

	Accuracy	Precision	Recall	F1-Score
padding	99.66%	95.13%	95.81%	94.45%
nopadding	99.66%	93.07%	89.03%	91.00%

Table 5
Comparison between LogDeep(Centralized) and LogDeep(FL) for HDFS data set.

	Accuracy	Precision	Recall	F1-Score
LogDeep (Centralized Section 4.1)	99.66%	95.13%	95.81%	94.45%
LogDeep (FL) even splitted	99.59%	92.21%	94.22%	93.16%
LogDeep (FL) uneven splitted	99.65%	94.25%	93.73%	93.99%

Table 6
Even splitting of HDFS data set for 5 FL clients.

Client No.	Train normal seq.	Test normal seq.	Test abnormal seq.
1	971 (20%)	553,366	16,838
2	971 (20%)	553,366	16,838
3	971 (20%)	553,366	16,838
4	971 (20%)	553,366	16,838
5	971 (20%)	553,366	16,838

By appending a unique log key, the log-event sequences are artificially extended in order to be processed with the LSTM. In principle, this results in a trade off. On the one hand, artificial lengthening gives better results for the metrics. Attaching a unique log key to the test abnormal sequences, labels the sequence explicitly, since the padding character is unique for anomalous sequences, which makes AD trivial. Therefore, the results are artificially improved due to padding, which is misleading. On the other hand, without padding, log-event sequences that are too short would simply be discarded and not fed into the AD process. The number of test abnormal sequences is reduced from 16,838 to 10,647. The small amount of test abnormal sequences compared to the test normal sequences indicates an imbalanced data set. In summary, the AD algorithm itself is not inadequate, but the data set to which it is applied is. The results of this experiment can be seen in Table 4 and Fig. 5, where a significant reduction in the detection rate of anomalies can be seen in the form of a reduction in recall.

4.2. *Federated Learning LogDeep with HDFS data set*

For FL experiments we ported the working LogDeep implementation to 5 clients. Each client runs the same LogDeep implementation, but with different data sets. The number of clients was chosen due to resource limitations. The server used FedAvg as update aggregation methods and started the FL process only when all 5 clients were available. We mainly investigate the capability of FL, how clients learn DL models for AD with different sized data sets and number of epochs. The number of epochs were decreased to 3 and compared to the results for number of epochs of 300 in Section 4.2.3. Basically, we used padding to make the results comparable with the centralized implementation. With the training data set evenly and unevenly distributed over 5 clients, almost the same performance can be achieved as with the centralized implementation. The comparison between centralized and FL implementation is shown in Table 5, where uneven and even splitting of the training data set are listed separately.

4.2.1. *Federated Learning LogDeep with even splitted HDFS data set*

First we divided training data evenly among all clients. With even as well as with uneven splitting the sequences are shuffled. Otherwise it could be that certain sequences only appear at certain times, e.g., at night, which would only be seen by one client, if we keep the temporal order of sequences. The even distribution is described in Table 6.

The results of the metrics for even splitting of training data set for 5 clients and 3 rounds can be seen in Fig. 6, Figs. 7, 8 and 9. On the x-axis the respective rounds are shown where the letter L indicates a local evaluation after training phase and letter G indicates a global evaluation after the updates from all clients have been aggregated.

The accuracy, as shown in Fig. 6, is already very high with almost 98% in the first round and then seems to stagnate. The reason for this is the imbalanced HDFS data set as explained in Section 4.1. The precision increases continuously as Fig. 7 shows because the false positive log sequence decreases over the rounds. A short-term high value of the recall is shown in Fig. 8, which decreases again in the course of the rounds due to refinement of the model parameters. This is related to the desired increase in precision, which in turn can be at the expense of the recall. Finally, the value of the F1 score climbs steadily, as can be seen in Fig. 9.

4.2.2. *Federated Learning LogDeep with uneven splitted HDFS data set*

We have selected the following uneven distribution key for the percentage distribution: Client 1 receives 5%, Client 2 receives 20%, Client 3 receives 20%, Client 4 receives 5% and Client 5 receives 50%. In our opinion, this distribution could describe a realistic scenario, where one client keeps a lot of log data locally and can use it for model training and other clients keep much less log data locally but can learn from other participating clients. The uneven distribution is described in Table 7.

The first signs that FL has a useful effect can be seen in the following results. The clients with the lowest percentage of training data sets first achieve poor results and then improve on account of collaborative learning. The results of the metrics for 5 clients and 3 rounds can be seen in Figs. 10, 11, 12 and 13.

The accuracy is again high from the start as Fig. 10 shows because of the imbalanced data set as explained in Section 4.2.1. Fig. 11 shows

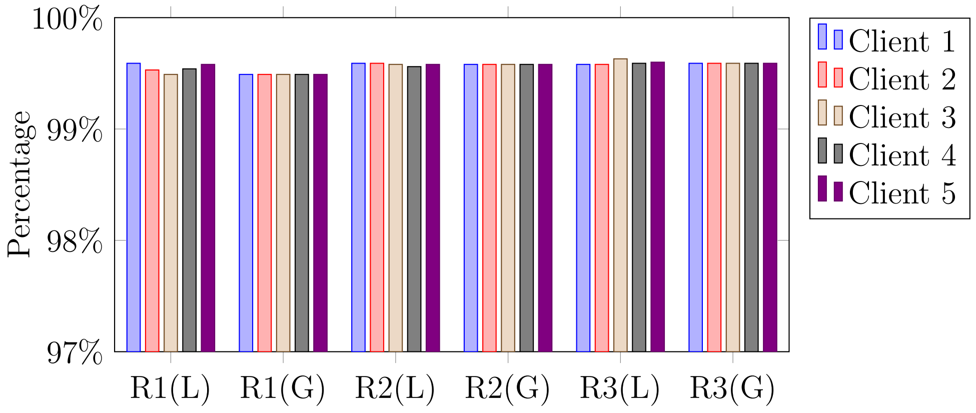

Fig. 6. Accuracy for 5 Clients with even splitted HDFS data set.

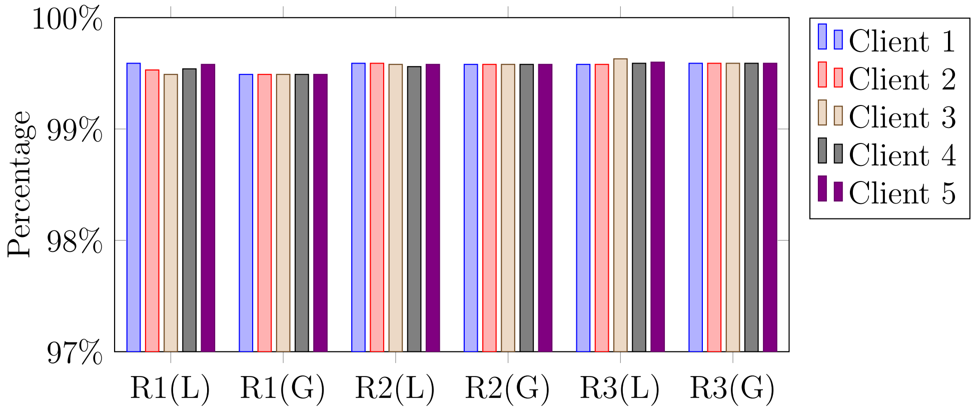

Fig. 7. Precision for 5 Clients with even splitted HDFS data set.

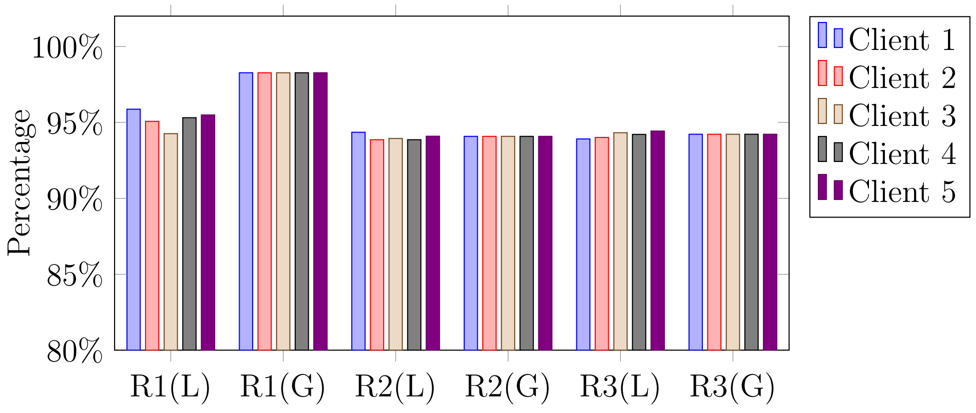

Fig. 8. Recall for 5 Clients with even splitted HDFS data set.

Table 7
Uneven splitting of HDFS data set for 5 FL clients.

Client No.	Train normal seq.	Test normal seq.	Test abnormal seq.
1	243 (5%)	553,366	16,838
2	971 (20%)	553,366	16,838
3	971 (20%)	553,366	16,838
4	243 (5%)	553,366	16,838
5	2427 (50%)	553,366	16,838

for Client 4 a precision of only around 70%. This could be due to the fact that Client 4 only receives 5% of the train sequences. Client 1 also received only 5% of the train sequences but due to the random splitting of the HDFS train dataset it is possible that LSTM from Client 1 simply received a more significant portion in terms of AD than Client 4. Because of the padding function the values for the recall are also already high from Round 1 on as Fig. 12 shows. The small amount of data received by Client 4 and its impact on the F1 score can be seen in Fig. 13.

4.2.3. Federated Learning with HDFS data set: Number of epochs

For this series of experiments we vary the number of epochs for the uneven distributed HDFS training data set. The purpose is to show that FL is able to reduce the number of epochs and thus shorten training time while still producing acceptable results. For this experiment, we reduce the number of epochs from 300 to 3. A total of 3 FL rounds

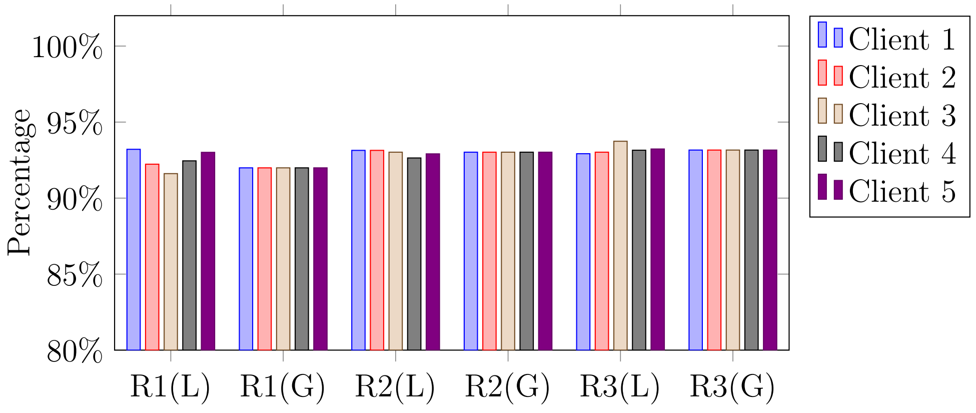

Fig. 9. F1-Score for 5 Clients with even splitted HDFS data set.

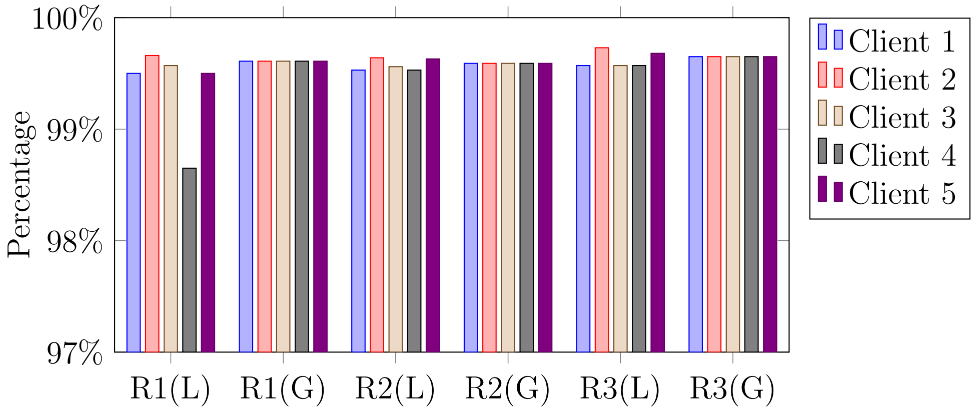

Fig. 10. Accuracy for 5 Clients with uneven splitted HDFS data set.

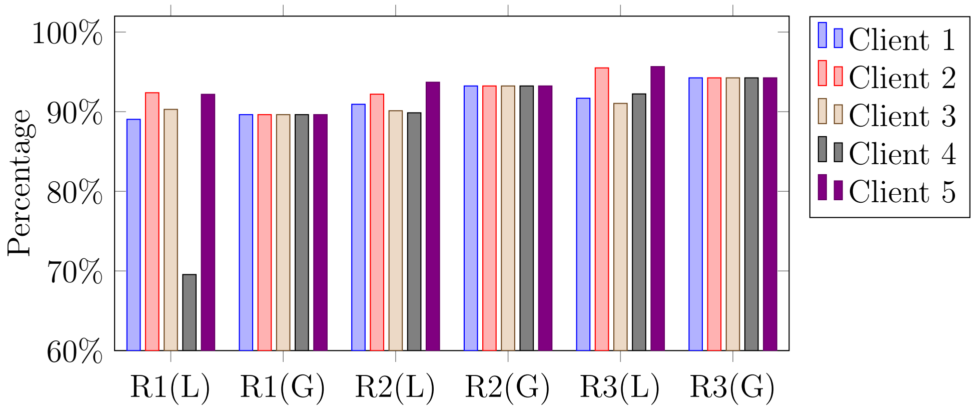

Fig. 11. Precision for 5 Clients with uneven splitted HDFS data set.

are run. In this experiment, the effect of FL is seen most clearly. By reducing the epoch, the model still converges very quickly. A comparison between 300 and 3 epochs can be seen in Table 8. Nearly equal metrics can be obtained. The reduced precision comes from increased false positive alarms, but even more anomalies can be detected. The results of the metrics for 5 clients and 3 rounds with 3 epochs of training can be seen in Figs. 14, 15, 16 and 17.

Figs. 14, 15, and 17 clearly demonstrate the effect of uneven splitting. The same distribution of training sequences has been used as shown in Table 7. Compared to the results from Sections 4.2.1 and 4.2.2, the reduction of the epochs leads to a slower increase of the accuracy and precision. Because of the padding function, the recall as shown in Fig. 16 remains at a high level from the beginning.

Table 8
Comparison between LogDeep(FL) 300 Epochs and LogDeep(FL) 3 Epochs.

	Accuracy	Precision	Recall	F1-Score
LogDeep (FL) 300 Epochs	99.65%	94.25%	93.73%	93.99%
LogDeep (FL) 3 Epochs	99.54%	87.15%	99.17%	92.77%

5. Discussion

The evaluation in Section 4 demonstrates that LSTM models in general and the proposed LogDeep re-implementation are able to detect contextual anomalies with high accuracy. While validating already published scientific results for the HDFS data set, we noticed that a

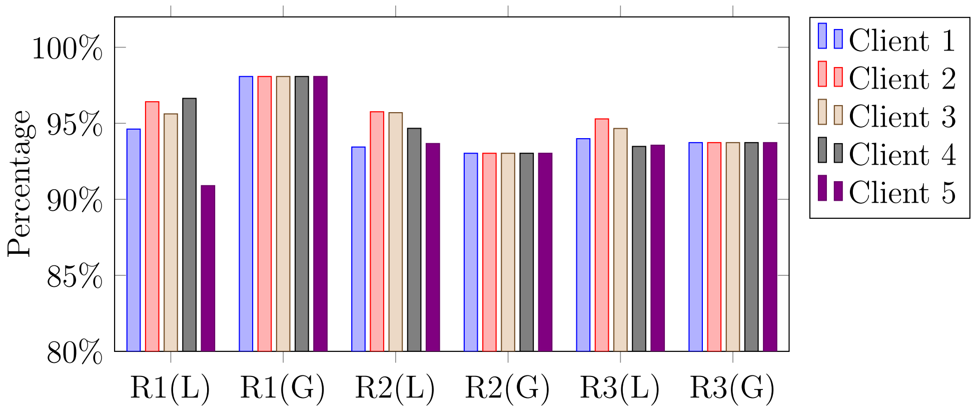

Fig. 12. Recall for 5 Clients with uneven splitted HDFS data set.

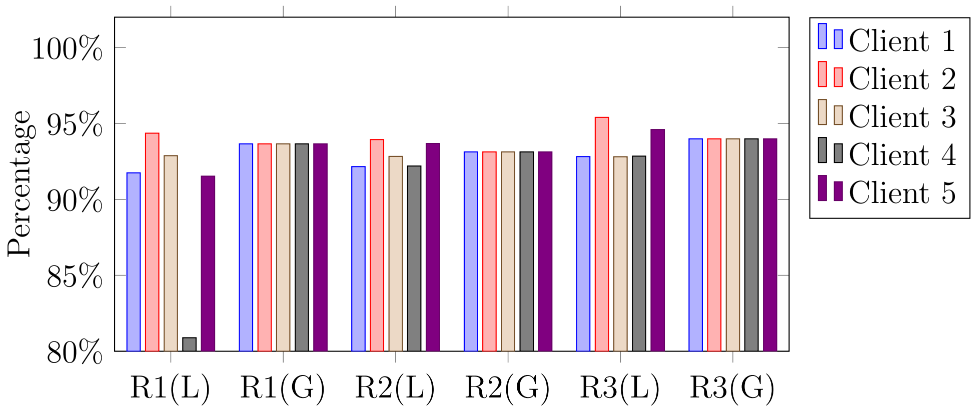

Fig. 13. F1-Score for 5 Clients with uneven splitted HDFS data set.

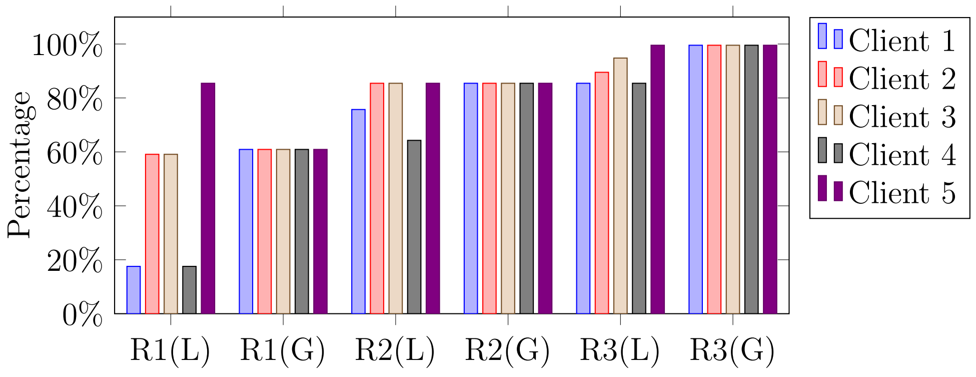

Fig. 14. Accuracy for 5 Clients with uneven splitted HDFS data set and 3 epochs training.

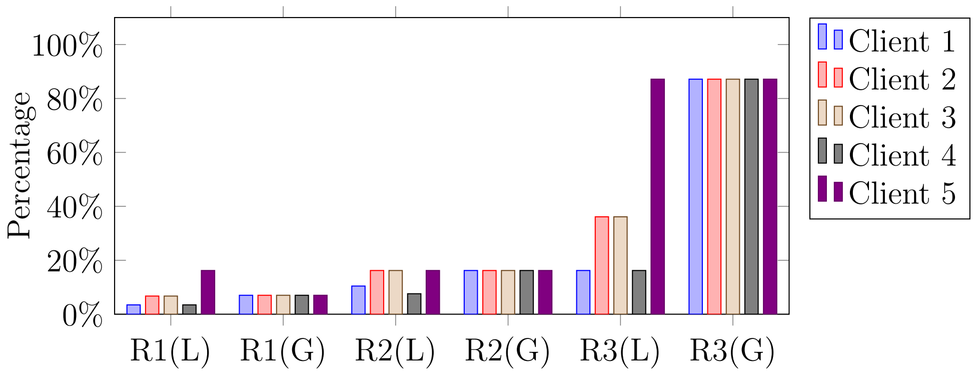

Fig. 15. Precision for 5 Clients with uneven splitted HDFS data set and 3 epochs training.

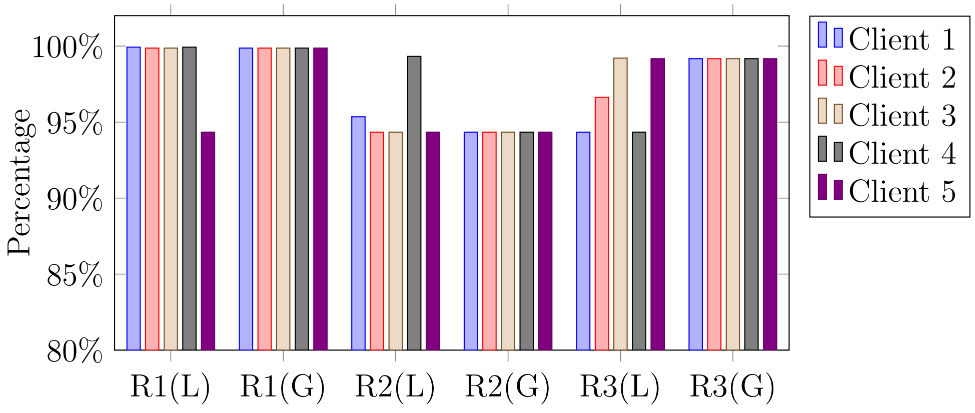

Fig. 16. Recall for 5 Clients with uneven splitted HDFS data set and 3 epochs training.

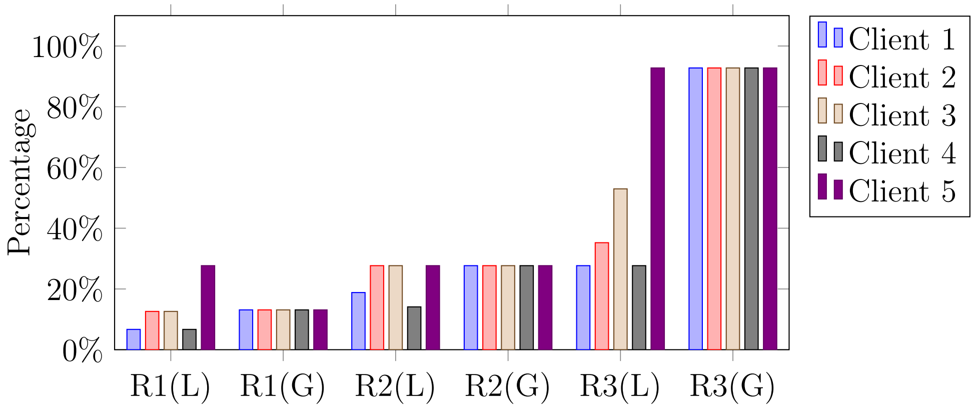

Fig. 17. F1-Score for 5 Clients with uneven splitted HDFS data set and 3 epochs training.

padding function was used. Padding results in an artificial lengthening of log-event sequences. That means if log-event sequences are not at least as long as the window size, they will not be considered at all and discarded. Especially for the HDFS data set this has undesired effects. The train and test normal sequences are sufficiently long log-event sequences. This is important for the applied window size of the LSTM model. Contrarily, the test abnormal sequences are artificially lengthened with a unique log key. For AD it is trivial to detect these log-event sequences, because they do not occur during training and testing with normal sequences (Himler et al., 2023). Similar observations have been made by Landauer, Skopik, and Wurzenberger (2023).

In the next steps, we implemented the central approach in a FL environment. For this purpose we used the open source framework flower, which allows to transfer central approaches into FL approaches. For the experiments we implemented only 5 clients due to limitations in computational resources. Furthermore, we have only examined FedAvg as an aggregation algorithm. Newer aggregation algorithms are currently being researched. These aim to improve performance, for example, by selecting the correct clients and excluding unsatisfying local models (Lavaur et al., 2022; Li, Ma, Deng, Choo, & Yang, 2022). For the HDFS data set slightly worse metrics for the FL approach could be achieved compared to a centralized approach. This can be explained by the aggregation algorithm FedAvg where only an averaging of the aggregated updates takes place. Nevertheless, the results show that in case of an uneven distribution over several clients, where one client holds a lot of log data locally, FL allows other clients to learn a lot from it and adapt their models locally. The results can be seen in Table 5 and demonstrate the advantages of FL. In addition, our experiments with reducing the number of epochs in Section 4.2.3 are already showing promising results, which will lead to reduced training time for DL models in the future. Our results in Section 4.2.2 show that a heterogeneous distribution of log data, FL can support clients to learn and benefit from each other without an actual exchange of data sets taking place. In the course of our experiments, we have considered the distribution of data sets and the number of epochs as one of the main control factors. The clients with the lowest percentage of training data sets first achieve not so good results and then can improve the performance due to collaborative learning. The experiments in Section 4.2.3, the effect of FL is seen most clearly. By reducing the number of epochs, the model still converges very quickly. It can be seen quite exactly the distribution of train sequences in percentages as in Table 7 to the individual Clients in the first round of the local calculation of the model. Compared to the results from Sections 4.2.1 and 4.2.2, the reduction of the epochs leads to a slower increase for the Accuracy and Precision. Nevertheless, open challenges remain for further research in this area, a few of which are:

- Further research would be needed to determine if LSTM is the best DL model to analyze log-event sequences. Current research shows the emergence of combining individual DL models that were previously treated separately. The advantages of different approaches are able to mitigate the disadvantages of others and thus lead to better results in detection accuracy and/or performance (Wang, Zhang, Wang, & Cao, 2021).
- The handling of imbalanced data sets where many normal log lines face very few anomalous log lines or vice versa would need further investigation. Furthermore, it is conceivable to reconsider the log line sampling methodology, such as augmenting the count of anomalous log lines through oversampling techniques (Farzad, 2020).
- To be able to generalize the results of this paper, in the future the experiments should be carried out on a larger scale involving a larger number of clients. A small number of clients could result in

a skewed representation of data, which can lead to biased model updates.

- As the number of clients continues to grow, it is important to implement better aggregation algorithms than FedAvg. By including clients with superior models, overall performance can be increased. In contrast one or more clients can intentionally or unintentionally degrade the overall model by setting certain model parameters. Targeted attacks on FL networks such as data poisoning attacks are described in Tolpegin, Truex, Gursoy, and Liu (2020).
- Despite that advantage of FL that raw data has not to be shared, local models still have to be shared. Furthermore, there is an additional effort for the calculation of the final score for the AD.

6. Conclusion and future work

In this paper we showed the necessary design steps and prerequisites to consider for developing an approach for AD utilizing DL which can handle a HDFS data set, and integrated this approach into a FL environment. Our implementation uses a LSTM and AD for log-event sequences. In course of the implementation, limitations came to light that should be considered in future design processes. It is imperative to have a basic understanding of how the log data to be analyzed are collected and structured. One of the biggest challenges for AD is to find the most effective DL method for the use case at hand and its suitability for a given data set.

We adapted a state of the art open source application called LogDeep and were able to replicate the results from paper Du et al. (2017) which uses a HDFS (Xu et al., 2009) data set. However, we found a limitation that is caused by the used HDFS data set, which in turn can be bypassed with the padding function. Through this function it is possible to extend too short log-event sequences with respect to the window size, which is important for the application of LSTM. This has to be done in the test abnormal sequences set which renders detection of these sequences trivial. Consequently, the excellent detection performance reported in the papers of Du et al. (2017) and Landauer, Skopik et al. (2023) cannot be achieved without the padding function. With the limitations in mind, we ported our central implementation of LogDeep to a FL environment. There is evidence that FL has a positive impact on the AD of log data, but due to the limitations described above, further scientific work is needed to substantiate this statement. In general FL represents a promising approach for future AD applications. Because of the ongoing interconnection of modern infrastructures the secure exchange of log data across users, organizations, and countries is of major importance. The application of FL provides AD systems a large knowledge base to spot attacks and adversarial behavior in log data. This in turn contributes to better countermeasures and greater resilience of future systems.

CRediT authorship contribution statement

Patrick Himler: Conceived and designed the analysis, Collected the data, Performed the analysis, Wrote the paper. **Max Landauer:** Conceived and designed the analysis, Contributed data or analysis tools, Performed the analysis, Wrote the paper. **Florian Skopik:** Conceived and designed the analysis, Collected the data, Performed the analysis, Wrote the paper. **Markus Wurzenberger:** Conceived and designed the analysis, Collected the data, Performed the analysis, Wrote the paper.

Declaration of competing 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.

Data availability

Data will be made available on request.

Acknowledgments

Funded by the European Union through the European Defence Fund under GA no. 101121403 - NEWSROOM and GA no. 101103385 - AInception. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

References

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., et al. (2020). Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Berlin, Heidelberg: Springer-Verlag.

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, *41*(3), 1–58.

Donglee-Afar (2020). LogDeep. GitHub repository, GitHub https://github.com/donglee-afar/logdeep.

Du, M., & Li, F. (2016). Spell: Streaming parsing of system event logs. In *2016 IEEE 16th international conference on data mining* (pp. 859–864). IEEE.

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). ACM.

Farzad, A. (2020). Log message anomaly detection with oversampling. *International Journal of Artificial Intelligence and Applications (IJAIA)*, *11*(4).

Farzad, A., & Gulliver, T. A. (2020). Unsupervised log message anomaly detection. *ICT Express*, *6*(3), 229–237.

Guo, Y., Wu, Y., Zhu, Y., Yang, B., & Han, C. (2021). Anomaly detection using distributed log data: A lightweight federated learning approach. In *2021 international joint conference on neural networks* (pp. 1–8). IEEE.

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

He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering* (pp. 207–218). IEEE.

Himler, P., Landauer, M., Skopik, F., & Wurzenberger, M. (2023). Towards detecting anomalies in log-event sequences with deep learning: Open research challenges. In *Proceedings of the 2023 European interdisciplinary cybersecurity conference* (pp. 71–77).

Ito, R., Tsukada, M., & Matsutani, H. (2021). An on-device federated learning approach for cooperative model update between edge devices. *IEEE Access*, *9*, 92986–92998.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, *12*, Article 100470.

Landauer, M., Skopik, F., & Wurzenberger, M. (2023). A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. arXiv preprint arXiv:2309.02854.

Lavaur, L., Pahl, M.-O., Busnel, Y., & Autrel, F. (2022). The evolution of federated learning-based intrusion detection and mitigation: A survey. *IEEE Transactions on Network and Service Management*, *19*(3), 2309–2332.

Li, S., Cheng, Y., Liu, Y., Wang, W., & Chen, T. (2019). Abnormal client behavior detection in federated learning. arXiv preprint arXiv:1910.09933.

Li, L., Fan, Y., & Lin, K.-Y. (2020). A survey on federated learning. In *2020 IEEE 16th international conference on control & automation* (pp. 791–796). IEEE.

Li, B., Ma, S., Deng, R., Choo, K.-K. R., & Yang, J. (2022). Federated anomaly detection on system logs for the internet of things: A customizable and communication-efficient approach. *IEEE Transactions on Network and Service Management*, *19*(2), 1705–1716.

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, *9*(20), 4396.

LogPAI (2021). *Loghub: A large collection of system log datasets for AI-driven Log analytics*. Zenodo, URL https://zenodo.org/record/3227177.

McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (pp. 1273–1282). PMLR.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., et al. (2019). Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*: *vol. 19*, (no. 7), (pp. 4739–4745).

Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., & Kao, O. (2020). Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE international conference on data mining* (pp. 1196–1201). IEEE.

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* (pp. 575–584). IEEE.

Rahman, S. A., Tout, H., Talhi, C., & Mourad, A. (2020). Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Network*, *34*(6), 310–317.

Song, X., Wu, M., Jermaine, C., & Ranka, S. (2007). Conditional anomaly detection. *IEEE Transactions on knowledge and Data Engineering*, *19*(5), 631–645.

Tolpegin, V., Truex, S., Gursoy, M. E., & Liu, L. (2020). Data poisoning attacks against federated learning systems. In *Computer security–ESORICs 2020: 25th European symposium on research in computer security, ESORICs 2020, guildford, UK, September 14–18, 2020, proceedings, part i 25* (pp. 480–501). Springer.

Villa-Pérez, M. E., Alvarez-Carmona, M. A., Loyola-Gonzalez, O., Medina-Pérez, M. A., Velazco-Rossell, J. C., & Choo, K.-K. R. (2021). Semi-supervised anomaly detection algorithms: A comparative summary and future research directions. *Knowledge-Based Systems*, *218*, Article 106878.

Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. *Ieee Access*, *7*, 41525–41550.

Vinayakumar, R., Soman, K., & Poornachandran, P. (2017). Long short-term memory based operation log anomaly detection. In *2017 international conference on advances in computing, communications and informatics* (pp. 236–242). IEEE.

Wang, Q., Zhang, X., Wang, X., & Cao, Z. (2021). Log sequence anomaly detection method based on contrastive adversarial training and dual feature extraction. *Entropy*, *24*(1), 69.

Wurzenberger, M., Skopik, F., Settanni, G., & Fiedler, R. (2018). AECID: A self-learning anomaly detection approach based on light-weight log parser models. In *ICISSP* (pp. 386–397).

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (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., Wang, Z., Wang, W., Jiang, J., Dong, X., et al. (2021). Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd international conference on software engineering* (pp. 1448–1460). IEEE.

Yang, Q., Liu, Y., Chen, T., & Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, *10*(2), 1–19.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., 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).