Automated Software Engineering (2022) 29:61
https://doi.org/10.1007/s10515-022-00365-7

SiaLog: detecting anomalies in software execution logs using the siamese network

Shayan Hashemi[1] · **Mika Mäntylä**[1]

Received: 31 August 2021 / Accepted: 18 September 2022 / Published online: 13 October 2022

Abstract

Detecting anomalies in software logs has become a notable concern for software engineers and maintainers as they represent anomalies in software execution paths and states. This paper propose a novel anomaly detection approach based on the Siamese network on top of Recurrent Neural Networks(RNN). Accordingly, we introduce a novel training pair generation algorithm to train the Siamese network which reduces generated training significantly while maintaining the F_1 score. Additionally, we propose a hybrid model by combining the Siamese network with a traditional feedforward neural network to make end-to-end training possible, reducing engineering effort in setting up a deep-learning-based log anomaly detector. Furthermore, we provides validations of the approach on the Hadoop Distributed File System (HDFS), Blue Gene/L (BGL), and Hadoop map-reduce task log datasets. To the best of our knowledge, the proposed approach outperforms other methods on the same dataset at the F_1 scores of respectively 0.99, 0.99, and 0.94 on HDFS, BGL, and Hadoop datasets, resulting in a new state-of-the-art performance.To further evaluate the proposed method, we examine our method's robustness to log evolutions by evaluating the model on synthetically evolved log sequences; we got the F_1 score of 0.95 on the HDFS dataset at the noise ratio of 20%. Finally, we dive deep into some of the side benefits of the Siamese network. Accordingly, we introduce an unsupervised log evolution monitoring method alongside a visualization technique that facilitates model interpretability.

Keywords Log analysis · Anomaly detection · Siamese network · Deep learning

✉ Shayan Hashemi
shayan.hashemi@oulu.fi

Mika Mäntylä
mika.mantyla@oulu.fi

[1] M3S Research Unit, ITEE, University of Oulu, Oulu, Finland

1 Introduction

Log files are an unstructured text-based history of events that shed light on the software state during its execution. Each line of log files indicates a different event and may hold different types of information such as log-type, timestamp, process ID, thread ID, and log message. Analyzing log events allows developers to extract helpful information from the software state during the run-time. One of the log analysis applications is anomaly detection. Log anomaly detection may assist developers in software testing, debugging, or run-time monitoring.

Throughout recent years, deep learning has become the most predominant method in almost every machine learning problem. They have been utilized in tasks such as object detection and localization (Redmon et al. 2016; Liu et al. 2016), machine translation (Wu et al. 2016; Bahdanau et al. 2014), and action recognition (Alhersh and Stuckenschmidt 2019; Jhuang et al. 2013). Furthermore, they have been used to improve software testing, debugging, and stability. For instance, Deep Neural Networks (DNNs) are used in applications such as software defect prediction (Esteves et al. 2020), performance analysis (Velez et al. 2020), or reopened bugs accuracy prediction (Xia et al. 2015). Moreover, log anomaly detection is no exception, and DNNs have been widely utilized in this research area alongside other Machine Learning (ML) approaches.

There are two different approaches among the deep methods in log anomaly detection (Chalapathy and Chawla 2019). The first one is a binary classification task. It takes a sequence as input and outputs a binary value indicating if the sequence is an anomaly. The latter approach is sequence modelling, which trains only on the non-anomaly data and learns to model the system's normal behaviour, resulting in low probabilities for anomaly behaviour.

As non-anomaly data volume is significantly higher than anomaly data, sequence modelling is more common in log anomaly detection. However, training solely on non-anomaly data may result in models being unaware of anomaly events, making the approach unreliable in anomaly situations. Furthermore, since logs evolve due to software updates, models trained with non-anomaly data have limited capabilities to detect anomaly situations in evolved non-anomaly situations.

On the other hand, binary classification solves the previously mentioned problem by training the model on both anomaly and non-anomaly data. However, it comes with its own challenges; one of them is training on an unbalanced dataset. The obstacle comes into place when the proportion of anomaly to non-anomaly data is too small. More specifically, datasets contain dramatically more anomaly samples in comparison to non-anomaly ones.

Nonetheless, many solutions have been introduced to surmount the unbalanced data obstacle. Oversampling and undersampling are two straightforward approaches that strive to equalize the number of samples in two classes. Another way of dealing with unbalanced datasets is weighted training. It manipulates the cost function so that both classes' influences on the model's parameters are equal. However, setting training weights and oversampling may result in overfitting,

while undersampling ignores a colossal proportion of negative samples during the training process. A more steady solution may be synthetic data generation. Furthermore, it eliminates the disadvantages of oversampling yet results in equilibrium. However, it requires innovative methods to generate legitimate and reliable samples. This paper proposes a new approach based on the Siamese network (Bromley et al. 1994) to handle the unbalanced data in log anomaly detection.

The primary purpose of the Siamese network is metric learning, and it is vastly used in one-shot learning tasks such as face verification (Chopra et al. 2005; Wang et al. 2014), signature verification (Dey et al. 2017; Ahrabian and BabaAli 2019), and visual object tracking (Zhang et al. 2018; Bertinetto et al. 2016; Guo et al. 2017). Furthermore, it has also been utilized in video game anomaly detection (Wilkins et al. 2020). The proposed Siamese network-based model takes advantage of both non-anomaly and anomaly data while not demanding balanced training data.

More in-depth, we attempt to learn an embedding function for log sequences that maps sequences of the same class (non-anomaly or anomaly) adjacent to each other while maximizing the distance between opposing classes' sequences. We also propose a sampling technique inspired by negative sampling (Mikolov et al. 2013) to generate pairs for the Siamese network's training process. The proposed algorithm significantly reduces the training costs of the Siamese network.

Furthermore, we evaluate the proposed method through various experiments. Accordingly, we examine the impact of different pair generation algorithms on the Siamese network, try different classifiers on top of the embedding neural network, and compare the best performer to state-of-the-art methods. Moreover, we evaluate our model's robustness on evolved log sequences and propose a method to monitor log evolutions at production time. Besides, we reveal a solution to visualize the embedded sequences to make human administration of log sequences possible. Finally, we construct a hybrid model by imposing the Siamese network on a feedforward neural network, investigating the Siamese network's positive impact. Replication package of our work is available[1]. Our main contribution is the Siamese network utilization in the anomaly detection task RQ 1. We provide additional research contributions via research questions RQ2 and RQ3:

RQ 1. Design: *How could the Siamese network be employed for software log anomaly detection task?* We propose an architecture of an arbitrary classifier on top of an embedding function, trained within the Siamese network, alongside a pair generation algorithm. However, we need to answer two more subquestions to respond to the question thoroughly:

RQ 1.1. Architecture: *How could a proper neural network architecture be found for the embedding function?* We conduct an experiment to spot a high performance architecture using the Hyperband algorithm, see Sect. 5.1.

RQ 1.2. Pair generation: *How to generate pair for training the Siamese network to avoid generating all possible pairs while maintaining the accuracy?* We propose a new pair generation algorithm, named KPOne, to avoid

[1] Replication package at https://github.com/M3SOulu/SiaLogReplicationPackage.

training the Siamese network using all possible pairs. KPOne reduces the required training pairs by several orders of magnitude, see Section 5.2, with negligible classification accuracy loss, see Sect. 5.3.

RQ 2. Performance: How does the proposed methods perform in terms of accuracy and computational cost? We divide this question into following multiple more specific subquestions:

RQ 2.1. Accuracy: *How accurately does the proposed method perform in open public datasets compared to the state-of-the-art methods?* We present the state-of-the-art performance for three open public datasets, see Sect. 5.4.
RQ 2.2. Low-cost model: *Is it possible to produce a low-cost embedding function for the Siamese network at a low accuracy loss?* We show that in some cases SiaLog hyper-parameter search results in a low-cost model by default (BGL and Hadoop). However, when it is not achieved, a low-cost model could be handcrafted (HDFS), see Sect. 5.5.
RQ 2.3. Hybrid model: *Is it possible to merge the Siamese network's architecture and a feedforward model into a single deep neural network?* Yes, we propose an architecture (SiaLog Hybrid) to train the Siamese network and a neural network-based classifier together to make the end-to-end training possible. Furthermore, since the classification model is a single deep neural network it could benefit from parallelization of the GPU during inference, see Sect. 5.6.

RQ 3. Side benefits: *What are side benefits of using the Siamese network other than anomaly detection?* With every machine learning innovations comes new research questions and concerns. Thus, we choose three of the most important research questions that could be resolve as a side benefit of the proposed method. Research questions are listed as follow:

RQ 3.1. Robustness: *How accurately does SiaLog perform in noisy environments?* We examine noisy log sequences on three different datasets. The investigations show that performance to noise is data specific. In some cases (Hadoop), even a small noise ratio of 5% results in low performance (F_1 of around 0.85 and lower), while the impact is significantly lower for other data sets (HDFS, BGL), see Sect. 6.1.
RQ 3.2.Unsupervised log evolution monitoring: *Is it possible to monitor log evolutions without the need for labeled data?* Yes, we introduce a metric (fitness score) to monitor log evolutions using the embedded sequences based on mixture of gaussians and negative log probability. The fitness score demonstrates the embedding function's adaptiveness with the distribution of evolved logs, see Sect. 6.2.
RQ 3.4.Visualization: *How do the Siamese network embedded sequences are visualized in a plot?* For all datasets, we illustrate a visualization of the embedded sequences using three different dimension reduction algorithms (PCA, UMAP, and T-SNE), revealing that anomaly sequences are readily separable from non-anomaly ones. Furthermore, such visualizations could

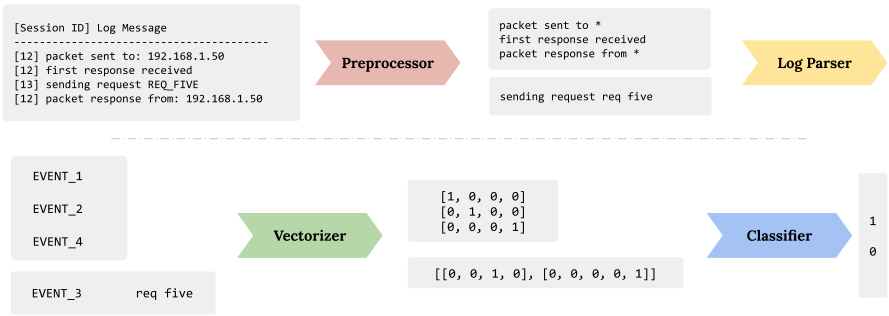

Fig. 1 System log anomaly detector's architecture as in Zhang et al. (2019)

be a valuable part of a log analyzer tool in future studies, see Sect. 6.3 and Fig. 7.

The remainder of the paper is organized as follow: Sect. 2 is dedicated to explaining required knowledge, reviewing famous previous works, and discussing datasets. The Siamese network, the methodology, and pair generation algorithms are explained in Section 3. The preprocessing, datasets, and evaluation metrics are discussed in Sect. 4. Section 5 comprises the reports of various experiments investigating the proposed method on deeper levels, while additional practical advantages are mentioned in Sect. 6. Finally, the conclusion and future work proposals are offered in Sect. 7.

2 Background and related works

2.1 General log analyzer architecture

Log anomaly detectors consist of multiple components, which are visualized in Fig. 1. The figure illuminates four components of log anomaly detectors: preprocessor, log parser, log vectorizer, and classifier.

The first component, the preprocessor,'s mission is to prepare log events for subsequent components. The preparations may include eliminating unnecessary information (such as IP addresses or invalid characters), extracting features from timestamps and log levels, and clustering logs based on their threads or process IDs. The preprocessor unit's output is passed to the next component, the Log parser (Zhu et al. 2019). The log parser identifies the log message parameters and extracts templates. Log message event types could be inferred by matching a log message with identified templates. Depending on the vectorizer's capabilities, which is the next component, event parameters might be carried along with the event type. The log vectorizer produces vectors from event types and parameters (if any). The vectors may take the form of one-hot encoded, semantic, or template IDs, depending on the classifier's architecture. Then, vectors are given to the classifier, which is the last component. The classifier's goal is to distinguish anomalous vectors. Machine

learning algorithms are prevalent for this component, as they have shown promising results in sequence modelling and classification.

2.2 Related works

One of the most well-known and effective anomaly detection methods is Principal Component Analysis (PCA), mentioned by Xu et al. (2009b). The method first forms a session-event matrix, similar to the document-term matrix in Natural Language Processing (NLP), where each cell indicates the number of occurrences of a particular event that occurred in an individual session. Next, the matrix is passed to an analysis of principal components. Then the anomalies are detected by distinguishing the session vector's projection length in the residual space.

In another approach, Lou et al. (2010) uses the session-event matrix and mine invariants that satisfy the majority of the sessions. Thus, anomalies occur in sessions that lack the satisfaction of the mined invariants. While all mentioned works focus on designing general-purpose algorithms, Yu et al. (2016) presents a method that compares the log messages to a set of automata to calculate the workflow divergence and is labeled as an anomaly as a result. However, it focuses on the log anomaly detection in OpenStack's logs specifically.

As the Deep Neural Networks have grown more mature in recent years, they have gained popularity among log anomaly detection research. Many approaches are leveraging different types of Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) or Gated Recurrent Unit (GRU) (Chung et al. 2014), while others are detecting anomalies by making use of Convolutional Neural Networks (CNNs) (LeCun et al. 2015b).

DeepLog, as the most well-known log anomaly detection method, proposed by Du et al. (2017), uses DNN in the classifier component. After parsing log events, DeepLog encodes the event types and parameters into vectors. Next, the model, which is based on LSTM, trains on data from non-anomaly execution only to predict the next log event given previous events. After the training, the model predicts a low probability for some events in anomaly sequences as it has trained on non-anomaly data only.

Although the methods mentioned before accurately detect log anomalies, Zhang et al. (2019) suggests that advances made by previous works are based on a close(d)-world assumption where logs are static, while, in real-world applications, logs are continuously evolving. Log evolutions are considered undoubtedly important these days, as many companies are continuously delivering software updates to their customers (Leppänen et al. 2015). Thus, Zhang et al. (2019) suggest LogRobust, a novel method for log anomaly detection. LogRobust proposes a new vectorization technique called "semantic vectorization" to approximately compensates for the evolution of log messages. It also suggests utilizing the attention-based Bidirectional Long Short-Term Memory (Bi-LSTM) to encounter the execution path evolutions. Furthermore, the authors present a technique to emulate log evolutions by applying noise.

LogAnomaly (Meng et al. 2019) presents another novel yet practical approach for vectorization called "template2vec" that takes synonyms and antonyms into account, making the vectorization process more reliable. Furthermore, LogAnomaly claims that it can detect sequential anomalies as well as quantitive ones. While every previously mentioned deep method applies LSTM to model log sequences (predict the next log event), LogAnomaly uses an LSTM on Term Frequency-Inverse Document Frequency (TF-IDF) vectors to construct a binary classifier.

On the other end of the spectrum, Lu et al. (2018) applies CNN instead of LSTM to form a binary classifier. The research also introduces an effective embedding method to transform one-hot encoded log events to vectors called "log-key2vec". This method results in efficient dimension reduction of one-hot encoded vectors.

Logsy (Nedelkoski et al. 2020) is a first paper proposing the use of Transformer (Devlin et al. 2018). Logsy embeds log messages into a vector space so that the non-anomaly messages congregate around the origin while anomaly messages embed with some distance from the origin. Its contributions also include a novel loss function, which makes the learning process of embedding operation possible.

All previous deep-learning-based methods, regardless of their core components, obeyed one of the two previously mentioned approaches. They either applied binary classification or modeled the sequence. However, this paper presents a third option that utilizes the Siamese network to circumvent the previously mentioned challenges in a different matter. Harnessing the Siamese network's power, our method proposes a new approach to embed the log sequences into vectors, so that embedded sequence vectors of different classes are readily separable and classifiable in the new space.

3 Proposed method

As earlier mentioned, previous deep methods either train on non-anomaly events only or apply binary classification to detect anomalies. However, both of those approaches are prone to deficiencies.

In the first (non-anomaly events only) approach, the model training would not encounter log events that only occurred in an anomaly situation. For instance, in a distributed data storage solution software, a hard drive failure event is not a regular event by any means. Furthermore, in the HDFS dataset, from the twenty-nine total events, only nineteen of them occurred in non-anomaly situations. Not training on a proportion of the input space may result in unexpected model behavior. Going more in-depth, as the model has not been trained on anomaly-only events, it shows random behavior at the time of facing those events. In the latter (binary classification) approach, the model's training suffers from the unbalanced dataset. Although some solutions have been discussed for the unbalanced data problem, all of them are accompanied by their limitations.

Throughout the rest of the section, we propose a novel approach based on the Siamese networks due to their excellent performance in one-shot learning problems (Chopra et al. 2005; Wang et al. 2014; Zhang et al. 2018; Bertinetto et al. 2016; Guo et al. 2017) and their stability on unbalanced data (Sun et al. 2019). Our proposed method takes advantage of both data classes without any sampling tricks or

Fig. 2 The proposed method's step-by-step overall view

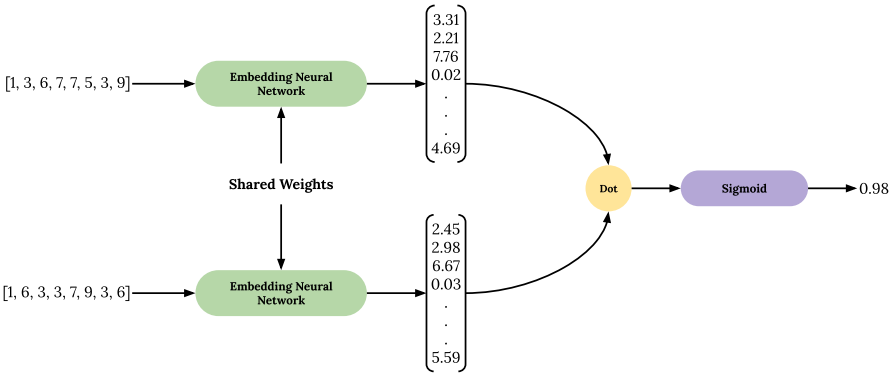

Fig. 3 The Siamese network's architecture

weighted training. Figure 2 demonstrates the steps required to achieve anomaly detection using the Siamese network. After acquiring the proper data, training pairs are generated from it, the Siamese network is trained on pairs, the embedding model is extracted from the Siamese network, the data is embedded to vectors using the embedding model, and an arbitrarily classifier is trained using the embedded vectors.

3.1 The Siamese network

The Siamese network, illuminated in Fig. 3, was initially invented to resolve the one-shot learning problem (Bromley et al. 1994) by forming a similarity-based embedding function. It packs two neural networks with shared weights (they are indeed the same neural networks and may be considered one; however, discriminating them makes the Siamese network's architecture more interpretable) and a similarity metric. During the training, at first, pairs of samples are passed to the neural networks. Next, the neural network embeds them into vectors. Then, the similarities between the vectors are measured. Lastly, the optimization process updates the weights of the neural networks with respect to the fact that similar pairs (same class) should hold high similarity values for their output vectors, while it is the contrary for dissimilar pairs (pair from different classes). At the end of the training process, the model embeds the same class samples close to each other while different class samples are embedded away from each other. In this paper, we use the Siamese network to train a deep embedding neural network that transforms log sequences into vectors so that embedded vectors of sequences of the same class are close to each other while being apart from the other class.

After the Siamese network converges, we extract the embedding neural network and embed all training sequences into vectors. As the embedded vectors of different classes are well separated, they are excellent training data for an arbitrary classifier. So, we train a classifier to work on top of the embedding neural network to form an anomaly detection method. During the test time, the embedding neural network transforms the input sequences into vectors and passes them to the classifier to be classified as non-anomaly or anomaly sequences.

Since the invention of the Siamese network, different loss function has emerged for it. One of them is the contrastive loss function (Hadsell et al. 2006). It operates utilizing the Euclidean distance, confirming enough space between embedded vectors of different classes while keeping vectors from the same class close to each other. However, during our experiments, we inquired about another loss function based on the sigmoid of inner product and cross-entropy loss function (LeCun et al. 2015a), which performed better than the contrastive loss. Going more in-depth, we use the sigmoid function on embedded sequences' inner product to construct a similarity measure. Supposing x_1 and x_2 are respectively the first and second embedded vectors, the measure may be formulated as:

$$sim(x_1, x_2) = \sigma(x_1 \cdot x_2).$$

On top of the similarity measure, we use the cross-entropy loss function. So, the final loss function may be formulated as:

$$J(x_1, x_2, y) = -(y.\log(sim(x_1, x_2)) + (1 - y).\log(1 - sim(x_1, x_2))).$$

3.2 Pair generation

As the Siamese network requires its training input to be in pairs, a proper pair generation method is required. Generated training pairs must include two types of pairs in order to train the Siamese network. The first type is similar pairs in which the entities are from the same class, with the training target being one. The second type is dissimilar pairs in which the entities are from different classes, with the training target set to zero. To shed more light, assume that A is an anomaly sequence, while N is a non-anomaly sequence. From four possible pair permutations, (A, A) and (N, N) are considered as similar pairs, while (A, N) and (N, A) are dissimilar ones. The following paragraphs contain two pair generation algorithms for training the Siamese network.

Algorithm 1: Generating pairs using the All algorithm

```
GenerateAllPairs (D)
    inputs : The dataset D, which contains sequences denoted by s and
             targets denoted by t
    output: Pairs generated using the All algorithm
    foreach (s1, t1) ∈ D do
        foreach (s2, t2) ∈ D do
            if t1 == t2 then
                addPair(s1, s2, 1)
            if t1 != t2 then
                addPair(s1, s2, 0)
```

The first approach, which is quite straightforward, generates every possible pair. Going more in-depth, every sequence in the dataset pairs with all other sequences except for itself. The pseudo-code could be seen in Algorithm 1. Although this method is sensible and easy to implement, it is impractical for massive datasets. Alongside the exponential growth of pairs quantity, this approach generates dramatically more similar pairs than dissimilar ones. We call this approach the "All" pair generation algorithm.

Algorithm 2: Generating pairs using the KPOne algorithm

```
GenerateKPOnePairs (N, P, K)
    inputs : The data subsets N and P, which subsequently contain negative
             (non-anomaly) and positive (anomaly) sequences. The constant
             K where K ∈ ℕ and is the proportion of dissimilar to similar
             pairs.
    output: Pairs generated using the KPOne algorithm
    foreach n ∈ N do
        sn = sampleSet(N);
        addPair(n, sn, 1);
        for 0 to K do
            sp = sampleSet(P);
            addPair(n, sp, 0);
    foreach p ∈ P do
        sp = sampleSet(P);
        addPair(p, sp, 1);
        for 0 to K do
            sn = sampleSet(N);
            addPair(p, sn, 0);
```

The second approach focuses on training efficiency. In this approach, which is inspired by Mikolov et al. (2013), for each sequence within the dataset, we sample

Table 1 The explanation of the datasets used for SiaLog experiments

Dataset Name	Samples		Sequence creation method
	Anomaly	Non-anomaly	
HDFS Xu et al. (2009a)	16838	558223	*block_id*
Hadoop Lin et al. (2016)	3328	78829	Log file
BGL Oliner and Stearley (2007)	7632	72257	Label locality

one sequence from the same class and K sequences from the different class, generating $K + 1$ pairs for each sequence. In other words, this approach samples a subset of all pairs instead of generating them all. The pseudo-code is observable in Algorithm 2. This method reduces training time and computational cost, making it feasible for training the Siamese network. We name this approach the "K Plus One (KPOne)" pair generation algorithm. As the K value increases, so does the computational effort. We noticed improvements in our experiments while increasing K until $K = 3$.

The number of samples generated in each epoch, and the computational cost accordingly, may vary significantly based on the choice of the pair generation algorithm. Assuming that n_n and n_a are subsequently the number of non-anomaly and anomaly samples within a dataset. The number of pairs generated by the All algorithm is

$$N_{All} = n_a^2 + n_n^2 + 2n_a n_n - n_a - n_n$$

, while the number of generated pairs for the KPOne algorithm is

$$N_{KPOne} = Kn_a + Kn_n + n_a + n_n$$

when K is the dissimilar samples count. It is blindingly obvious that for large numbers of n_a and n_n, the value of N_{KPOne} is dramatically smaller than N_{all}. So, the computational cost of the All pair generation algorithm is larger than the KPOne.

4 Data and Performance measures

This section explains the datasets, preprocessing steps, and evaluation metric in our experiments.

4.1 Datasets

As we strive to assess the authenticity of the proposed method, we evaluate our method on multiple publicly available datasets. We chose HDDS, BGL, and Hadoop datasets as they possess labels for anomalous log events. In what comes next, the

datasets, which are provided by He et al. (2020), are explained, while Table 1 summarizes all informations.

HDFS: Hadoop Distributed File System (HDFS) is a distributed file system that is significantly fault-tolerant and low-cost to deploy. The dataset was first introduced by Xu et al. (2009a) and later considered a benchmark in log anomaly detection domain. It is produced by running map-reduce tasks on more than 200 Amazon's EC2 nodes and labelled via Hadoop domain experts. The sequences are created based on the block ID of each log message. After the preprocessing, we encountered 16838 anomaly and 558223 non-anomaly samples.

Hadoop: Hadoop is a big data processing framework allowing distributed processing of extensive data. The dataset, launched by Lin et al. (2016), contains a five-machine Hadoop cluster log, each having an Intel(R) Core(TM) i7-3770 CPU and 16GB of RAM. The logs are generated by running two separate applications, Word Count, which counts the number of words in the input, and Page Rank, an algorithm used by search engines. The anomalies in the dataset are machine down, network disconnection, and disk full. However, we merge these three types of anomaly into one. Furthermore, as some sequences in the dataset are too long, we produced subsequences from them using a sliding window(window size of 100). The outcome was comprised of 78829 anomaly samples and 3328 non-anomaly samples after the preprocessing. As the number of anomaly samples was more than non-anomaly ones, as anomalies are a combination of multiple anomaly classes, we invert the dataset's target variable (changing anomalies to non-anomalies and vice versa) to preserve the anomaly detection task's nature.

BGL: The dataset, produced by Oliner and Stearley (2007), is collected from a BlueGene/L supercomputer at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 CPUs and 32,768GB of RAM. The sequence creation process was based on the label locality of log events. Furthermore, as some sequences became too long, we generated subsequences utilizing a sliding window (windows size of 300 and stride of 50) from them. We ended up with 7632 anomaly and 72257 non-anomaly samples.

4.2 Preprocessing

As previously mentioned, experiments use HDFS, BGL, and Hadoop log datasets. As our research focuses on classification, we prepare a vectorized variant for each dataset. For the HDFS, we use a vectorized dataset provided by He et al. (2020), while for BGL and Hadoop, we parse the data using the Drain algorithm (He et al. 2017), as it performs better according to Zhu et al. (2019).

Although the datasets are preprocessed, parsed, vectorized, and ready for classification, we discovered many redundant sequences, especially in the HDFS dataset. Redundancy not only raises the required processing power for training but also compromises the authenticity of the evaluation as some test samples may appear in the training set. So, our first and only step of pretraining is to remove redundant sequences. After removing the redundant sequences, the HDFS dataset contained 4,124 unique anomaly and 14,259 unique non-anomaly sequences, while the number

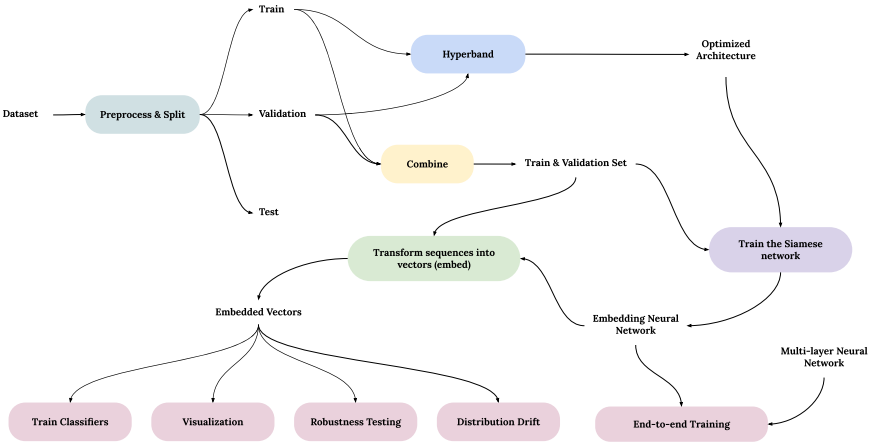

Fig. 4 Overall view of data splitting and experiments

of non-anomaly and anomaly samples are 350 and 22,262 consequently in the BGL dataset. Furthermore, the Hadoop dataset contained 3,328 anomaly and 78,829 non-anomaly samples after removing redundancies.

4.3 Data splitting

We split each dataset into train and test sets (90% for training and 10% for testing) using simple random sampling technique, mentioned by Reitermanova et al. (2010). The trainset is used to train the Siamese network and classifiers, while the test set is used exclusively for evaluation. However, before training the Siamese network, we take a small portion of the training data (equal to 3% of all data) and use it as the validation set. Then we start training the Siamese network utilizing a pair-generation algorithm. The validation set's purpose is to find the most suitable neural network architecture and hyper-parameters and control the embedding function and classifier overfitting. After founding proper architecture and hyper-parameters, the validation set serves no purpose. Thus, it is merged into the training set for retraining the neural network. Figure 4 illuminates an overview of data splitting and experiments processes, presenting an overall view of the whole process.

4.4 Performance measure

The nature of the anomaly detection task is unbalanced, meaning that there are significantly more negative samples in comparison to positive ones. In such circumstances, the binary classification accuracy is not a valid metric for measuring performance. So, we use another metric called "F_1 score" to measure and compare performance. Suppose *TP*, *TN*, *FP*, and *FN* are respectively true positives, true negatives, false positives, and false negatives. The "precision" metric formulated as

Table 2 The embedding neural network's architecture found by the Hyperband algorithm the HDFS dataset, described layer by layer

Property	1st	2nd	3rd	4th	5th	6th	7th
Layer type	Embedding	Bi-LSTM	Bi-LSTM	LSTM	Dense	Dense	Dense
Output units	32	128	128	64	128	640	128
Activation	N.A	Tanh	Tanh	Tanh	ReLU	ReLU	Linear

Table 3 The embedding neural network's architecture found by the Hyperband algorithm for the BGL dataset, described layer by layer

Property	1st	2nd	3rd	4th	5th
Layer type	Embedding	LSTM	Dense	Dense	Dense
Output units	128	128	128	128	128
Activation	N.A	Tanh	ReLU	ReLU	Linear

$$precision = \frac{TP}{TP + FP}$$

shows the accuracy of the model's positive prediction. On the other hand, the "recall" metric demonstrates the model's reliability in predicting all positive samples and formulates as

$$recall = \frac{TP}{TP + FN}$$

Finally, the F_1 score is the harmonic mean of precision and recall simplified to

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

However, we multiply F_1 scores by one hundred to expose more details in the results.

5 Experiments, results, and comparisons

This section focuses on spotting a proper architecture for embedding neural networks, validating different pair generation algorithms, analysing different classifiers, comparing our method to other state-of-the-art methods, and introducing low-cost and hybrid models.

5.1 Embedding neural network's architecture

Motivation: As the heart of our method is the embedding neural network trained inside the Siamese network, we want the embedding neural network to perform at its best. Spotting an optimal architecture and hyper-parameters is a challenging

Table 4 The embedding neural network's architecture found by the Hyperband algorithm the Hadoop dataset, described layer by layer

Property	1st	2nd	3rd	4th	5th	6th
Layer type	Embedding	GRU	Dense	Dense	Dense	Dense
Output units	64	64	64	64	64	64
Activation	N.A	Tanh	SiLU	SiLU	SiLU	Linear

Table 5 The results shows the performance of the Siamese network on different datasets using different pair generation algorithms

Dataset	Algorithm	Loss		Pairs	k value	Reduction factor
		Train	Test			
HDFS	All	~0.00	0.002	211,231,337	3	3,302
	KPOne	0.01	0.03	63,968		
BGL	All	0.004	0.005	503,488,232	3	5,566
	KPOne	0.005	0.013	90,448		
Hadoop	All	~0.00	~0.00	6,487,347,580	10	7,178
	KPOne	0.015	0.016	903,727		

step in deep learning projects. So, our goal is to spot a suitable architecture and hyper-parameters.

Method: Multiple algorithms, such as Grid Search, Random Search, Bayesian Optimization, and Evolutionary Optimization, have been proposed to tune neural network architecture and hyper-parameters. However, we choose the Hyperband algorithm (Li et al. 2017) for its performance and computational efficiency to attain a solid architecture along with its hyper-parameters. The Hyperband algorithm was executed three times (to avoid local optima) with default parameters on all available pairs in the training set to minimise the Siamese loss on the validation set. In other words, Hyperband used the training set to train multiple different architectures and the validation set to compare the architectures to find the best performance.

Findings: Tables 2, 3, and 4 contain details of embedding neural networks architectures and hyper-parameters found by the Hyperband algorithm. Tables 2 shows that multiple layers of LSTMs are required to achieve decent results on the HDFS dataset. On the other hand, Table 4 show the model performs better using GRU layers on Hadoop. Moreover, Table 3 reveals that only one LSTM layer is enough for the BGL Datasets, as anomalies in that dataset is more reliant on individual events rather than entire sequences.

5.2 Pair generation algorithms comparison

Motivation As discussed before, generating pairs using the All pair generation algorithm is computationally expensive. Therefore, we proposed an algorithm for

Table 6 The accuracy comparison between different classifiers and embedding SiaLog (NN) trained using different pair generation algorithm. The first model is trained using the All pair generation algorithm while the second one is trained using the KPOne

Dataset	Classifier	F_1 Score	
		All algorithm	KPOne algorithm
HDFS	SiaLog (KNN)	99.39	99.39
	SiaLog (SVM)	99.57	99.51
	SiaLog (NN)	99.62	99.51
	SiaLog (LR)	99.39	99.39
BGL	SiaLog (KNN)	98.11	98.00
	SiaLog (SVM)	96.89	96.47
	SiaLog (NN)	96.89	95.76
	SiaLog (LR)	99.47	99.31
Hadoop	SiaLog (KNN)	94.69	93.44
	SiaLog (SVM)	95.01	94.23
	SiaLog (NN)	95.82	94.21
	SiaLog (LR)	94.69	93.17

generating pairs to reduce the computational cost. In this experiment, we aim to compare the two pair generation algorithms.

Method We trained two models for each dataset with the same architecture found in the previous experiment, using pairs generated with the All and KPOne pair generation algorithms. Furthermore, we tried different values of K in the KPOne algorithm and found out that $k = 3$ works the best for the HDFS and BGL datasets while $k = 10$ performed better for Hadoop. After the training, we compare the Siamese network's loss value and the classifiers' accuracy across the models. It must be stated that the test loss value is calculated after the hyper-parameter optimization process in the previous section. In fact, the Hyperband algorithm neither trained nor optimized based on any pairs containing any sequence from the test set.

Findings Table 5 shows that the All pair generation algorithm results in smaller loss values for the Siamese network. However, the difference in the number of generated pairs is significant. Furthermore, Table 6 (in the next subsection) demonstrates that the classification result differences are negligible. All in all, considering the computational cost, the All algorithm might not fit into as many cases.

5.3 Classifiers comparison

Motivation: A classifier is needed to classify the embedded sequences after training the embedding neural network within the Siamese network. In this experiment, we aim to evaluate several classifiers for this purpose.

Method: We choose Logistic Regression (LR), Support Vector Machine (SVM), K Nearest Neighbours (KNN), and multi-layer neural networks as candidate classifiers, as we found them more popular compared to other classifiers. The neural networks classifier consists of two layers. The first one is activated using the Rectifier Linear Unit (ReLU), while the second layer (output layer) leverages the sigmoid activation function for binary classification. At first, we embed all train sequences

Table 7 The comparison of SiaLog and other state-of-the-art deep methods

Dataset	Method	Precision	Recall	F1 Score
HDFS	DeepLog	0.90	0.81	0.85
	LogRobust*	0.97	0.98	0.98
	LogAnomaly*	0.89	0.81	0.85
	CNNLog	0.99	0.97	0.98
	SiaLog	0.99	0.99	0.99
	Feedforward model	0.89	0.95	0.97
BGL	LogRobust*	0.87	0.9	0.88
	CNNLog	0.93	0.97	0.95
	SiaLog	0.98	1.00	0.99
	Feedforward model	0.98	0.98	0.98
Hadoop	LogRobust*	0.82	0.99	0.90
	CNNLog	0.99	0.85	0.92
	SiaLog	0.93	0.94	0.94
	Feedforward model	0.90	0.91	0.91

* Custom parsers and vectorizers of LogRobust and LogAnomaly are replaced with standard parsers

into vectors. Then, train the classifiers on the embedded sequences. During the test time, each sequence is embedded using the embedding neural network and passed to the classifier for prediction.

Findings: As Table 6 exposes the results, all classifiers achieve outstanding results. SiaLog achieved F_1 scores between 99.39 to 99.62 on the HDFS, 95.76 to 99.47 on BGL, and 93.17 to 95.82 on Hadoop with four different classifiers. Furthermore, we found the difference between the All and KPOne pair generation algorithms is negligible. Achieving accurate and consistent results among different classifiers bears witness to the fact that the embedding neural network is working precisely and as expected. For each dataset, we choose the classifier with the best results as SiaLog for upcoming experiments.

5.4 Comparison to state-of-the-art methods

Motivation: This section compares the Best performers from the previous subsection against state-of-the-art deep log anomaly detection approaches.

Method: We bring SiaLog results from the previous experiment and select DeepLog (Du et al. 2017), LogRobust (Zhang et al. 2019), LogAnomaly (Meng et al. 2019), and CNNLog (Lu et al. 2018) as competitors. We also train a neural network with the same architecture as combining the embedding and classifier neural networks into a single unit. This neural network, mentioned as the Feedforward model, allows us to investigate if utilizing the Siamese network yields any benefit.

All competitor methods are reimplemented in our environment. However, we replaced custom parsers and vectorizers in compatitors with the standard parser and vectorizer. In particular, LogAnomaly and LogRobust use a novel vectorizer approach alongside a custom parsing method. On the other hand, our approach and

Table 8 The handcrafted embedding neural network's architecture found by cross-validation between ten different candidate models for SiaLog Low-cost

Property	1st	2nd	3rd	4th	5th
Layer type	Embedding	Bi-LSTM	Dense	Dense	Dense
Output units	24	64 (32 × 2)	64	64	64
Activation	N.A	Tanh	Leaky ReLU	Leaky ReLU	Linear

environment uses standard parser and vectorizer. So, we evaluated them by replacing their parsers and vectorizers with standard components of our test environment while keeping their deep neural network architecture.

Findings: Table 7 shows that our SiaLog outperform all previous methods and its Feedforward rival. We see that, in the HDFS dataset, SiaLog has the F_1 score of 0.99 followed by LogRobust and CNNLog with the F_1 score of 0.98, the Feedforward model (F_1 score 0.97), and eventually DeepLog and LogAnomaly with the F_1 score of 0.95. Nevertheless, to the best of our knowledge, SiaLog achieves the best results ever on the HDFS dataset when redundant sequences have been removed. For the BGL dataset, SiaLog, with the F_1 score of 0.99, outperforms other approaches followed by the Feedforward model, CNNLog, and LogRobust, F_1 scores of 0.98, 0.95, and 0.88 respectively. Furthermore, it was the same story for the Hadoop datasets. However, since the dataset is more complicated compared to BGL, due to longer sequences and larger number of event types, most methods performed less accurately compared to BGL and HDFS. In this dataset, SiaLog performed the best at the F_1 score of 0.94, followed by the CNNLog, Feedforward model and LogRobust, F_1 scores of 0.92, 0.91, and 0.90. All in all, SiaLog outperformed both their Feedforward rivals and the state-of-the-art methods in all evaluations.

5.5 SiaLog low-cost

Motivation: In previous experiments, we found an architecture offering the state of the art performance for anomaly detection in the HDFS, BGL, and Hadoop datasets. However, training a model with architectures found by Hyperband is expensive for the HDFS dataset and was done in an HPC environment. In this experiment, we endeavour to handcraft a new architecture that is less taxing to train. After all, the software industry might not have the possibility or time to train models in an HPC environment. Furthermore, experiments, development, and utilization are cheaper and faster for the low-cost model. Finally, as the low-cost model is computationally less demanding, it is economical, fast, and scalable at the production time. However, despite all benefits, the low-cost model sacrifices accuracy to achieve the aforementioned goals.

Method: With the goal to find a suitable architecture, we first handcraft different architectures that are significantly less expensive to train than the architecture found by the Hyperband. Later, we train all models and choose the best architecture according to the F_1 scores. Alongside the F_1 score, we record two different metrics

Table 9 The table is the comparison of low-cost architecture with the architecture found by Hyperband. FLOPS column indicates the amount of floating-point operations required for the embedding neural network to transform a sequence into a vector. Moreover, the Parameters column reveals the number of trainable parameters in each architecture. Furthermore, the required training time for each architecture is mentioned in the Training time column

Architecture	F_1 Score	FLOPS	Parameters	Training time
SiaLog	99.62	222K	805K	150h 42min
SiaLog Low-cost	98.78	71K	27K	11h 17min

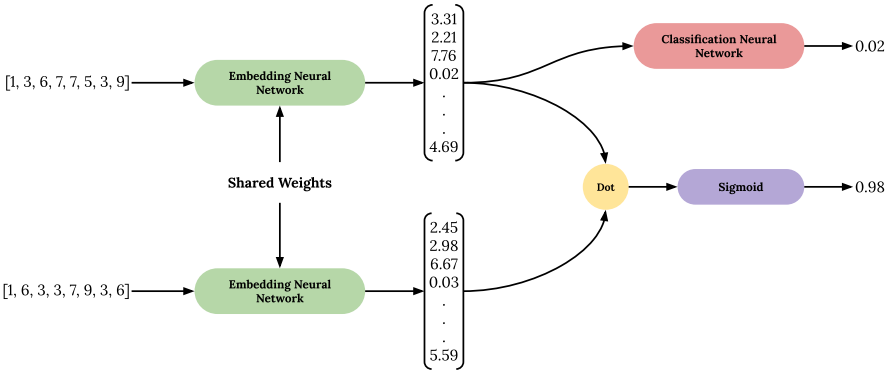

Fig. 5 The modified Siamese network's architecture for end-to-end training

for both models. The first metric is the number of floating-point operations (FLOPS) for one forward pass of the neural network. FLOPS is an implicit indication of computational cost during both development and production. Additionally, we calculate the number of parameters for each model. The number of parameters specifies the memory amount required to store and load the model and explicitly affects the training speed. Finally, we compare training time in a typical deep learning machine's hardware (a fourteen-cores Intel Xeon CPU paired up with two Nvidia Tesla P100 GPUs).

Findings: Table 8 demonstrates the chosen handcrafted architecture for the HDFS dataset while Table 9 compares the Best Performer model and low-cost model in computational cost, model size, and accuracy. The comparison sheds light on the fact that despite being computationally more affordable, three times less floating-point operation, 30 times fewer parameters, and reducing the training time by the factor of 13, the low-cost architecture does not considerably compromise the F_1 score, from 99.62 to 98.78. For example, the low-cost model could be retrained overnight with typical hardware while it is not possible for the best performer in typical hardware. The low computational cost makes the low-cost model suitable for environments where logs evolve rapidly, but less accuracy is tolerated.

Note: Accordingly, the architecture found by Hyperband was cheap enough for the BGL and Hadoop datasets. Thus, neither Hadoop nor BGL datasets were involved in this experiment, and this experiment is for HDFS only.

Table 10 The comparison of end-to-end training model (training classifier alongside the embedding neural network) with the best performer and feedforward model

Dataset	Model	Precision	Recall	F_1 Score
HDFS	SiaLog Hybrid	0.99	0.98	0.99
	SiaLog	0.99	0.99	0.99
	Feedforward model	0.99	0.95	0.97
BGL	SiaLog Hybrid	0.98	0.99	0.99
	SiaLog	0.98	1.00	0.99
	Feedforward model	0.98	0.98	0.98
Hadoop	SiaLog Hybrid	0.92	0.93	0.93
	SiaLog	0.93	0.94	0.94
	Feedforward model	0.90	0.91	0.91

5.6 SiaLog hybrid - combining the siamese and feedforward networks

Motivation: As previous experiments indicate, neural networks have been one of the well-performing classifiers in all datasets. Since the embedding function and the classifier could be neural networks, we strive to train them together, making end-to-end training possible. The end-to-end architecture may reduce design and engineering efforts as the classifier and embedding neural networks train simultaneously.

Method: Before training, we place the classifier network after the last component of the embedding neural network in the Siamese network. Therefore, the modified Siamese network is going to have two outputs. The first one is the similarity indicator, while the second one is the predicted label for the first entry of the Siamese network. Therefore, the modified Siamese network's loss is the cumulative loss of the Siamese network and cross-entropy classification. Figure 5 visualizes the architecture the modified Siamese network. Furthermore, to analyze the impact of the Siamese network on the accuracy, we compare the Hybrid model with the Best performer and Feedforward model mentioned in 5.4.

Findings: Table 10 confirms that the Hybrid model performs better than the Feedforward model and is almost on par with the Best performer.

6 Practical advantages

This section notes some practical advantages that become possible with the Siamese network. The first two advantages are related to log evolution, while the last one is related to visualization.

6.1 Robustness

Motivation: Software logs are continually evolving due to execution environments variations or developers' updates (Zhang et al. 2019). As training deep models is dramatically power consuming, it is not feasible to train the model for

Table 11 The evaluation results of different classifiers on synthetically evolved datasets. The noise ration indicates the ratio of the test set samples that are affected by synthetic log evolutions. Furthermore, all numbers are multiplied by one hundred to expose more information

Dataset	Method	F_1 Score / Noise Ratio				
		0%	5%	10%	20%	30%
HDFS	SiaLog (KNN)	99.39	97.97	97.07	95.30	92.77
	SiaLog (SVM)	99.51	97.92	97.11	95.16	92.68
	SiaLog (NN)	99.51	97.92	97.11	95.16	92.68
	SiaLog (LR)	99.39	98.02	97.11	95.43	93.07
	LogRobust	98.20	95.93	93.83	88.74	84.83
BGL	SiaLog (KNN)	98.00	98.00	98.00	98.00	98.00
	SiaLog (SVM)	96.47	96.47	96.47	96.47	96.47
	SiaLog (NN)	95.76	95.76	95.76	95.76	95.76
	SiaLog (LR)	95.35	95.35	95.35	95.35	95.35
	LogRobust	95.35	95.35	95.35	95.35	95.35
Hadoop	SiaLog (KNN)	93.44	93.28	91.88	88.64	85.20
	SiaLog (SVM)	94.23	93.24	91.99	88.82	84.01
	SiaLog (NN)	94.21	93.65	92.27	89.09	84.75
	SiaLog (LR)	93.17	93.56	92.15	89.05	85.23
	LogRobust	90.17	88.16	86.13	81.36	77.42

every minor software update or modification in execution environments. Accordingly, Zhang et al. (2019) introduces three methods for emulating log evolution synthetically by adding noise to log sequences. It is not rational to train models on synthetically generated data. However, synthetically generated data may help in evaluating and analyzing model performance on evolved logs.

Method: In this experiment, we apply the three methods of adding noise to log sequences (Zhang et al. 2019) to imitate log sequence evolutions. The methods comprise of duplicating, removing, and shuffling one or multiple element(s) within a sequence. Since generating a noisy dataset is a random process, we performed each test five times and calculated the results' averages.

Findings: Table 11 shows the classifiers' evaluation of synthetically evolved log sequences with different noise ratios. Harnessing the power of the Siamese network, classifiers maintained their accuracy formidably despite the evolutions. In the HDFS dataset, the F_1 score dropped from 0.99 to 0.92 in all classifiers when moving from the noise ratio of 0% to 30%, while in previous works, as show in Table 11, the F_1 score dropped from 0.98 to 0.84. Furthermore, in the Hadoop dataset, the F_1 score dropped from 0.94 to 0.84 in our method, while the accuracy loss was from 0.90 to 0.77 for previous work (LogRobust). This experiment shows our method's robustness appears to be stronger than previous works in the HDFS and Hadoop datasets.

Note: As the BGL is an event-based dataset, the F_1 score did not alter at all with any noise ratio, as noising methods were targeting sequences, not events.

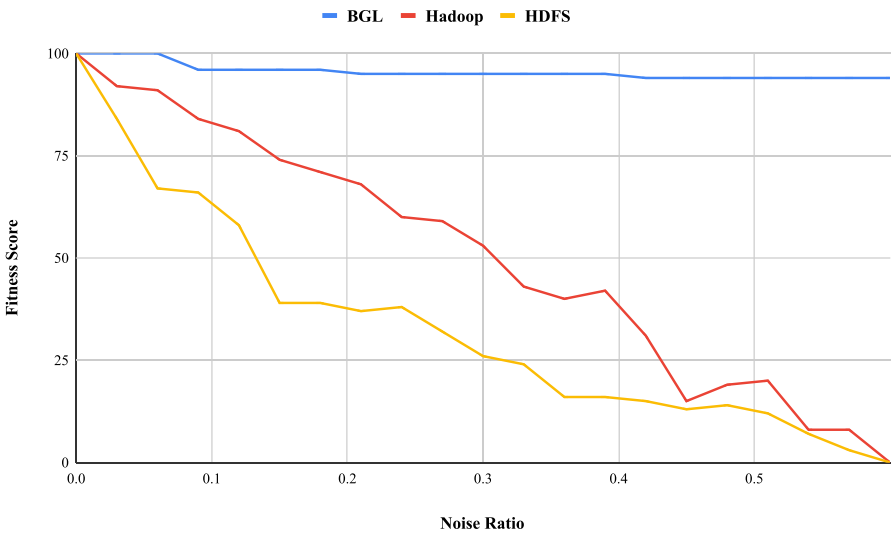

Fig. 6 The purge in fitness score as the noise ratio increase. It should be noted that positive scores are due to computing scores using probability density function

6.2 Unsupervised log evolution monitoring

Motivation: In the previous experiment, we confirmed that the proposed model is considerably robust to log sequence evolution. However, if log sequences proceed to evolve, the retraining process is inevitable. Since the retraining process is computationally expensive and time-consuming, we strive to find a solution to avoid unnecessary retraining. More in-depth, we seek a numeral value to present the trained model's reliability on evolved sequences. Although the F_1 score accuracy is the best measurement for this affair, we do not possess the sequence labels in the production time as the incoming data is entirely new. Hence, we require a new metric that indicates reliability without any labelling requirement.

Method: Since the embedding neural network transforms sequences into vectors, we may exploit embedded vectors' distribution to monitor log sequences' evolution. Thus, we introduce the fitness score as the indication of evolutions in log sequences. At first, the training sequences are embedded into vectors using the embedding neural network and modelled by a Gaussian mixture. Accordingly, the fitness score is computed as the average log-likelihood of embedded vectors of evolved sequences. The more the log sequences evolve, the lower the fitness value will be. Possessing such a metric, we may define a threshold and avoid the retraining process for trivial evolutions in production. Moreover, we may retrain the model as soon as the fitness score surpassed the threshold number. Needless to say, the threshold number might vary from task to task or even dataset to dataset.

Findings: We used the previously mentioned methods to imitate log evolutions and recorded the fitness score as the evolutions increased. Figure 6

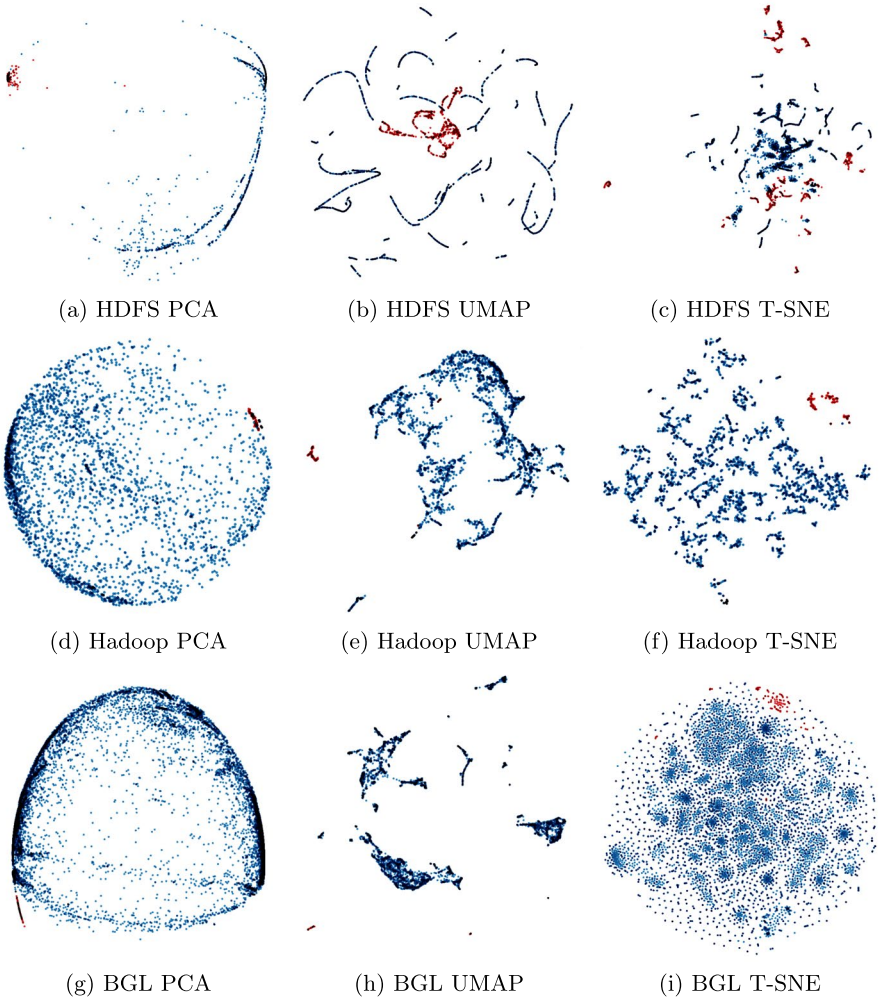

Fig. 7 The visualization of embedded sequences for different datasets. The dimension reduction happened using PCA, UMAP, and T-SNE algorithms. The blue dots represent normal sequences while the red dots represent anomalous sequences

visualizes the purge in fitness score as the evolutions grow. The purge might be an indication of the fitness score's reliability.

Note1: As the BGL is an event-based dataset, the fitness score did not alter dramatically with any noise ratio, as noising methods were targeting sequences, not events.

Note2: The fitness scores of the HDFS and Hadoop datasets range were different from each other. However, we scale their range to fit into a single plot.

6.3 Sequence visualization

Motivation: We have proposed multiple methods of evaluating the authenticity and reliability of the embedding neural network and model in previous experiments. However, human supervision for AI systems can bring brighter insights. One of the best solutions to human supervision is visualization. Furthermore, the visualization (of the embedding neural network's output in our case) gives humans the ability to supervise the embedding neural network's output and allows manual analysis.

Method: As the trained embedding network allows us to transform log sequences into vectors, we can use dimension reduction algorithms such as T-SNE (Maaten and Hinton 2008), UMAP (McInnes et al. 2018), and PCA (Abdi and Williams 2010) to reduce embedded sequences dimensions so they become visualizable and perceptible for humans. Accordingly, we embed all sequences from the train and test sets to vectors, reduce their dimensions, and plot the results on a canvas.

Findings: Figure 7 visualizes the embedded sequences using different dimension reduction methods on different datasets. The embedded non-anomaly sequences are coloured as blue, while the anomaly ones are coloured as red. The figure demonstrates that embedded sequences of different classes (non-anomaly/anomaly) are readily separable regardless of the dimension reduction algorithm. This fact might explain the high accuracy among all the classifiers.

7 Threats to validity

Though SiaLog outperformers almost every other method of software log anomaly detection task in various public open datasets, we believe there are a couple of limitations that come with it. Regarding technical limitations, the first important limitation is the computational cost. Since SiaLog trains a Siamese Network internally and all samples are trained in pairs, it comes with a hefty computational cost. Additionally, SiaLog uses RNN layers (LSTM and GRU) within the embedding neural network, making training longer as it reduces the ability to train the embedding neural network in parallel.

Second, the lack of advanced mechanisms in the embedding function, such as residual layers and attention, makes SiaLog more vulnerable to noise. Utilizing advanced layers and mechanisms could resolve this issue. However, the computational cost will further increase. Moreover, SiaLog demands lots of labeled datasets. Although all previous methods require labeled data somehow, SiaLog requires a fair share of anomaly data to produce a sufficient amount of pairs to train the Siamese Network, which we consider a limitation.

Third, the last technical limitation is the development complexity. As SiaLog is composed of multiple components and training the Siamese Network is more complicated than a regular deep neural network or a classic machine learning model, we found the implementation of our method more complex compared to the previous methods.

Regarding internal validity, We believe our work scores high since we tested our approach in multiple scenarios, including various datasets, model capabilities, and

noisy environments. Achieving consistent scores in different scenarios bears witness to the fact that our proposed method works as expected and results in more accurate software log anomaly detection. We also offer replication package though which our internal validity can be further scrutinized[2].

Regarding external validity, we think it could be higher as we did not have access to any suitable industrial dataset. This makes it hard to argue how our approach would work in industrial context. On the other hand, the purpose of this work was to present novel approach to anomaly detection and its initial validation. Future studies are needed to investigate the benefits and drawbacks of SiaLog or similar Siamese architectures in software log anomaly detection.

8 Future works

Although we introduced various benefits for SiaLog alongside anomaly detection, interesting future investigations remained. Future works may apply different side applications such as Root Cause Analysis by applying the Siamese network. On the other hand, More computationally cost-efficient neural networks such as CNNs might be applied inside the Siamese neural network to further reduce the computational cost in future studies. Furthermore, we think utilizing and benchmarking custom log parsers or even pair generation algorithm are also fruitful areas for future researches as well.

9 Conclusion

This paper proposed a novel approach to detect anomalies in software execution logs using the Siamese network structure combined with LSTM and GRU layers. We compared the results to the state-of-the-art deep-learning-based methods on the HDFS, BGL, and Hadoop log dataset for anomaly detection and showed that the proposed method achieves the best results on the aforenamed dataset. Furthermore, we conclude that the ability to achieve state-of-the-art performance is due to the Siamese network as the Feedforward neural network with the same architecture offered a considerably lower F1-score (0.996 vs 0.973). Furthermore, we proposed a novel algorithm to generate pairs to train the Siamese network. The algorithm reduces the training process's computational cost while maintaining accuracy. We also showed that the Siamese network achieves satisfactory results with smaller and computationally cheaper neural networks as well.

Moreover, we introduced multiple practical advantages of the Siamese network. We assess the robustness of our model to log evolutions. Additionally, we introduced an unsupervised method for log evolution measurement. Finally, we visualize the embedding function's output vectors using dimension reduction algorithms to make the neural network's output more perceptible.

[2] Replication package at https://github.com/M3SOulu/SiaLogReplicationPackage.

It is worth mentioning that the improvements made by SiaLog are not the most significant in this domain. However, it could be argued that room for improvements in the available datasets was small. On the other hand, SiaLog's practical advantages are, to the best of our knowledge and by the time of writing this paper, not offered by any counterpart method or technique. This bears witness to the fact that SiaLog could be a rational consideration for pragmatic software log anomaly detection.

Acknowledgements This work has been supported by the Academy of Finland (grant IDs 298020 and 328058). Additionally, the authors gratefully acknowledge CSC - IT Center for Science, Finland, for their generous computational resources.

Funding Open Access funding provided by University of Oulu including Oulu University Hospital.

References

Abdi, Hervé, Williams, Lynne J.: Principal component analysis. Wiley Interdiscip. Rev. Comput. Stat. **2**(4), 433–459 (2010)

Ahrabian, Kian, BabaAli, Bagher: Usage of autoencoders and siamese networks for online handwritten signature verification. Neural Comput. Appl. **31**(12), 9321–9334 (2019)

Alhersh, T., Stuckenschmidt, H.: On the combination of imu and optical flow for action recognition. In: 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pages 17–21. IEEE, (2019)

Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. *arXiv preprint*arXiv:1409.0473, (2014)

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., Torr, P. H.: Fully-convolutional siamese networks for object tracking. In: European conference on computer vision, pages 850–865. Springer, (2016)

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a "siamese" time delay neural network. In: Advances in neural information processing systems, pages 737–744, (1994)

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

Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 539–546. IEEE, (2005)

Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint*arXiv:1412.3555, (2014)

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

Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Lladós, J., Pal, U.: Signet: Convolutional siamese network for writer independent offline signature verification. *arXiv preprint*arXiv:1707.02131, (2017)

Du, ., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1285–1298, (2017)

Esteves, Geanderson, Figueiredo, Eduardo, Veloso, Adriano, Viggiato, Markos, Ziviani, Nivio: Understanding machine learning software defect predictions. Autom. Softw. Eng. **27**(3), 369–392 (2020). https://doi.org/10.1007/s10515-020-00277-4. (**ISSN 1573-7535**)

Guo, Q., Feng, W., Zhou, C., Huang, R., Wan, L., Wang, S.: Learning dynamic siamese network for visual object tracking. In: The IEEE International Conference on Computer Vision (ICCV), (2017)

Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 2, pages 1735–1742. IEEE, (2006)

He, P., Zhu, J., Zheng, Z., Lyu, M. R.: Drain: An online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS), pages 33–40, (2017). https://doi.org/10.1109/ICWS.2017.13

He, S., Zhu, J., He, P., Lyu, M. R.: Loghub: A large collection of system log datasets towards automated log analytics, (2020)

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

Jhuang, H., Gall, J., Zuffi, S., Schmid, C., Black, M. J.: Towards understanding action recognition. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), (2013)

LeCun, Yann, Bengio, Yoshua, Hinton, Geoffrey: Deep learning. Nature **521**(7553), 436–444 (2015)

LeCun, Yann et al.: Lenet-5, convolutional neural networks. 20 (5):14, (2015b), http://yann.lecun.com/exdb/lenet

Leppänen, Marko Mäkinen., Simo, Pagels, Max, Eloranta, Veli-Pekka, Itkonen, Juha, Mäntylä.: Mika V, Männistö, Tomi: The highways and country roads to continuous deployment. IEEE Software **32**(2), 64–72 (2015)

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, Talwalkar, Ameet: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. **18**(1), 6765–6816 (2017)

Lin, Q., Zhang, H., Lou, J., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pages 102–111, (2016)

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C.: Ssd: Single shot multibox detector. In: European conference on computer vision, pages 21–37. Springer, (2016)

Lou, J.-G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX Annual Technical Conference, pages 1–14, (2010)

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

McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint*arXiv:1802.03426, (2018)

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., et al.: Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, volume 7, pages 4739–4745, (2019)

Mikolov,Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, Dean, Jeff: Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., (2013). http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., Kao, O.: Self-attentive classification-based anomaly detection in unstructured logs. *arXiv preprint*arXiv:2008.09340, (2020)

Oliner, A., Stearley, J.: What supercomputers say: A study of five system logs. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), pages 575–584, (2007). https://doi.org/10.1109/DSN.2007.103

Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2016)

Reitermanova, Zuzana, et al.: Data splitting. WDS **10**, 31–36 (2010)

Sun, D., Wu, Z., Wang, Y., Lv, Q., Hu, B.: Risk prediction for imbalanced data in cyber security : A siamese network-based deep learning classification framework. In: 2019 International Joint Conference on Neural Networks (IJCNN), pages 1–8, (2019)

van der Laurens, Maaten, Geoffrey, Hinton: Visualizing data using t-sne. J. Mach. Learn. Res. **9**(Nov), 2579–2605 (2008)

Velez, Miguel, Jamshidi, Pooyan, Sattler, Florian, Siegmund, Norbert, Apel, Sven, Kästner, Christian: Configcrusher: towards white-box performance analysis for configurable systems. Autom. Softw. Eng. **27**(3), 265–300 (2020). https://doi.org/10.1007/s10515-020-00273-8

Wang, W., Yang, J., Xiao, J., Li, S., Zhou, D.: Face recognition based on deep learning. In: International Conference on Human Centered Computing, pages 812–820. Springer, (2014)

Wilkins, B., Watkins, C., Stathis, K.: Anomaly detection in video games. *arXiv preprint*arXiv:2005.10211, (2020)

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint*arXiv:1609.08144, (2016)

Xia, Xin, Lo, David, Shihab, Emad, Wang, Xinyu, Zhou, Bo.: Automatic, high accuracy prediction of reopened bugs. Autom. Softw. Eng. **22**(1), 75–109 (2015). https://doi.org/10.1007/s10515-014-0162-2. (**ISSN 1573-7535**)

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

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

Xiao, Yu., Joshi, Pallavi, Jianwu, Xu., Jin, Guoliang, Zhang, Hui, Jiang, Guofei: Cloudseer: workflow monitoring of cloud infrastructures via interleaved logs. ACM SIGARCH Comput. Archit. News **44**(2), 489–502 (2016)

Zhang, Xu, Xu, Yong, Lin, Qingwei, Qiao, Bo, Zhang, Hongyu, Dang, Yingnong, Xie, Chunyu, Yang, Xinsheng, Cheng, Qian, Li, Ze, et al.: Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 807–817, (2019)

Zhang, Y., Wang, L., Qi, J., Wang, D., Feng, M., Lu, H.: Structured siamese network for real-time visual tracking. In: Proceedings of the European conference on computer vision (ECCV), pages 351–366, (2018)

Zhu, Jieming, He, Shilin, Liu, Jinyang, He, Pinjia, Xie, Qi, Zheng, Zibin, Lyu, Michael R: Tools and benchmarks for automated log parsing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pages 121–130. IEEE, (2019)