Received January 22, 2022, accepted February 4, 2022, date of publication February 9, 2022, date of current version February 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150342

System Failure Detection Using Deep Learning Models Integrating Timestamps With Nonuniform Intervals

YIXIN HUANGFU[1], (Member, IEEE), SAEID HABIBI[1], (Member, IEEE), AND ALAN WASSYNG[2], (Member, IEEE)
[1]Department of Mechanical Engineering, McMaster University, Hamilton, ON L8S 4L8, Canada
[2]Department of Computing and Software, McMaster University, Hamilton, ON L8S 4K1, Canada

Corresponding author: Yixin Huangfu (huangfuy@mcmaster.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and in part by the Ontario Research Fund: Research Excellence (ORF-RE) Program.

ABSTRACT System logs play an important role in software development and system maintenance. Many system software programs continuously generate system logs during software runtimes for failure detection and diagnosis purposes. Currently, the analysis of system log data is mainly a manual process that highly depends on human knowledge and experience. This time-consuming task has become a problem because of the ever-increasing volume of log data. Existing studies have investigated machine learning and deep learning techniques to automate the failure detection task. This paper takes the deep learning approach and proposes two detection structures based on recurrent and convolutional neural networks. More importantly, this paper takes a step further by closely examining the timestamps of log data which existing studies have generally ignored. This study found that time information can be a distinguishing factor between regular and abnormal log sequences. Inspired by this observation, a novel method is proposed to integrate log timestamps in deep learning models using interpolation techniques. The evaluation results show that the log timestamps can significantly improve the performance of failure detection. Cross-comparison of the different models demonstrates that the proposed network structure can successfully utilize the timestamp information. The code is available on GitHub: https://github.com/hfyxin/Ts-models-log-data-analysis.git.

INDEX TERMS Data engineering, data mining, feature extraction, neural networks, pattern recognition, software maintenance.

I. INTRODUCTION

System logs are machine data constantly generated by large-scale software, such as online servers and operating systems. The events and variables recorded by the logs are the footprints of the software runtime. Depending on the system, the log messages generated in just a second range from a few to thousands of lines. This massive amount of data contains rich information for troubleshooting during software development or system maintenance. In software development, they can help developers identify software errors in the source code. In a web service application, log data can pinpoint failures that occur in the system.

Log data are traditionally analyzed by experienced software engineers, as it can be challenging to comprehend log messages without domain knowledge. This approach is reaching its limit with the development of large-scale and complex software [1]. With the growing complexity of software systems, the manual process requires more specialized personnel and maintenance costs. The massive volume of log data also makes analysis more time-consuming, reducing the recoverability and availability of the service, which can be crucial to a system that must be available all the time.

Recent research has focused on automating failure detection and recognition tasks using self-learning algorithms to analyze log data [2], [3]. This data-driven approach utilizes machine learning models to learn the feature patterns from anomaly logs. By fitting a large amount of training data, the model is able to process new log data and detect if a fault or failure condition occurs. The detection process is much more efficient than manual inspection and can save software

The associate editor coordinating the review of this manuscript and approving it for publication was Turgay Celik.

engineers precious time for system maintenance and defect triage.

The related works, as described in Section II, primarily emphasize statistical machine learning models [4]. They generally utilize a feature engineering approach to preprocess log data. Although performing reasonably well, these models are generally incapable of including time and sequential information of log messages, so there is room for improvement. On the other hand, some studies have investigated neural networks or deep learning approaches for failure detection [5], [6]. These models have an inherent advantage in handling sequential data and are capable of processing timestamp information. However, most of them opt to ignore log timestamps in the analysis.

Timestamps are an important piece of information available in all types of log data [7]. Most systems produce log data at an irregular rate, meaning that the log messages have nonuniform time intervals. The time elapsed between one log message and the previous varies, reflecting the unique temporal behavior of the software events. Timestamps contain temporal patterns that can contribute to anomaly detection, but none of the existing research has investigated their benefits.

In this paper, a novel deep learning approach for system failure detection is proposed using timestamps in log data. The proposed method integrates the nonuniform timestamps within the deep learning structure using interpolation techniques. This study is based on statistical observations from log sequences, which show the significance of log timestamps in improving fault detection.

The rest of this paper is structured as follows. Section II gives an overview of the existing research in the field. Section III reviews the HDFS dataset and presents the significance of timestamps. Section IV details the proposed deep neural network structure and the resample layer to integrate timestamps. Section V describes the experimental setup, and Section VI presents the findings from comparison tests. Section VII concludes this study.

II. RELATED WORK

Recent research has investigated automated system log analysis for detecting and classifying system failures. As logs evolved from simple *printf* commands to dedicated monitoring libraries, many analytic methods were developed to analyze the data. The rudimentary technique uses a keywords search for abnormal log lines, which has shown to be less effective in some applications [7], [8]. Rule-based automated detection [9] is an alternative, but it can be challenging for software developers to determine and maintain a comprehensive set of rules. Recent research mainly focuses on learning-based methods that automatically extract distinguishing information from data instead of largely relying on human expert input [1]. Automating log analysis consists of three stages: parsing, feature extraction, and classification explained as follows.

Parsing is a necessary first step to convert raw text log messages to usable structured fields. The need for parsing varies depending on the type of logs. Some logs follow a rigorous format and can readily be parsed with minimal effort [2], [3]. For these, a regular expression can successfully extract valuable features. Other logs tend to be harder to interpret, as some critical diagnostic fields are buried in syntactically complicated statements. These logs include HDFS, Microsoft online service system [11], and most operating system logs. For these, parsing has become a complex data mining problem [12], [13], and its correctness can affect the performance of the detection. Some utilize source code to generate tree-based syntax [14], while more research exploits the log data to extract log message templates [15], [16]. These syntactic templates or log signatures represent logging events, and variables within them are also extracted.

After parsing, log data are in sequential order and can have various lengths. Feature extraction is the process of structuring the parsed sequential log data into usable features for detection algorithms. This process can be done with hand-crafted rules based on domain knowledge. Such features include log message counts [17] and state ratio [14] for Hadoop, performance fingerprints for a datacenter [18], term-frequency inverse document frequency (TF-IDF) feature [19], [20], and many other statistical features [21]–[23]. These features are used in conjunction with machine learning models for detection or classification. Learning models can be formulated in a supervised manner, such as the logistic regression [18], decision tree [2], and support vector machines [3], [11], [21], as well as in an unsupervised manner, such as principal component analysis [14], invariant mining [17], and K-means clustering [24].

Another approach to the classification task is to utilize sequence information and apply workflow-based detection. Log messages that are related to specific identifiers are grouped to produce event sequences. These sequences represent system execution paths, which can be normal or abnormal. They can be stored as templates in a database [25], used to construct a finite state machine [26], or to generalize sequence patterns with clustering techniques [11]. New log sequences are compared against the processed ones to determine if a failure occurs.

Neural network models are new contenders in the log analysis field. The origin of neural networks dates back to the 1940s, but recent breakthroughs in computational capability have unleashed some of their potential [1]. The term deep learning refers to the recent form of neural network structures emphasizing the ability to train deeper networks. The most representative ones are Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). CNN is most popular in image recognition [27], [28], while RNN is preferred in natural language processing [29], [30]. The potential of deep learning models is not only limited to IT applications; they can also apply to many engineering disciplines such as mechanical system fault detection and state estimation [31], [32].

The structure of deep learning models in log analysis applications can be versatile. For example, RNN can process sequential inputs directly without the feature extraction process. In particular, by utilizing a many-to-one structure, an RNN network can infer the next possible log token based on previous ones [33]. Studies in [5], [6] used this approach to perform anomaly detection in real-time and include log timestamps in a separate RNN model. Another approach is to train a classification model to directly output a failure detection result based on log sequence input [34]. This approach allows the use of various network architectures in addition to RNN, such as sequential CNN [35], but these studies generally disregarded timestamps. Other variations of deep learning models were also investigated, such as the Autoencoder in [36].

III. LOG DATA PREPROCESSING AND OBSERVATION

This study uses the Hadoop Distributed File System (HDFS) log benchmark dataset originally introduced in [14]. The HDFS dataset is selected for its availability and popularity among related research. This section overviews the dataset and presents statistical patterns of log sequences.

The HDFS is a data storage and management system that runs on a cluster of computers. The log dataset is fully labeled and openly available from [16]. The original form of this dataset is a single 1.47 GB text file with 11.2 million log lines, recording 38.7 hours of HDFS system runtime. The log lines are essentially unstructured text. A breakdown of a sample log message is shown in Figure 1. Each log line contains formatted fields such as date and time, as well as a *statement* sentence that describes an action. The statement field in each log message contains valuable identifiers such as *block IDs* and type of operations (*event type*). Extracting this information and sorting them into categories is a challenging topic called log parsing. A few researchers have studied this topic and achieved good parsing accuracy [25], [26], [37]–[40]. In particular, the Drain algorithm [15] uses a tree structure to parse the event types in an unsupervised manner. This paper chooses the Drain algorithm to perform the parsing process for its accuracy and efficiency demonstrated in an evaluation study [16].

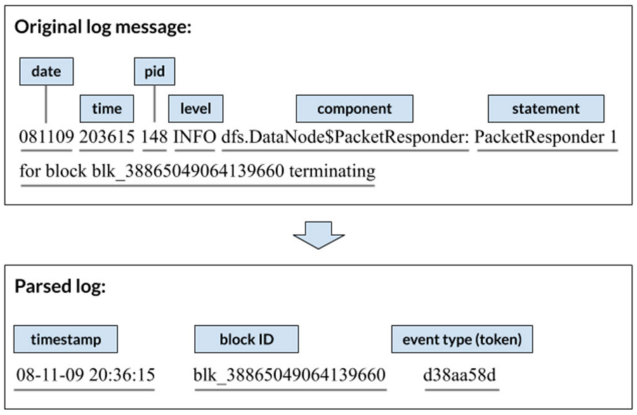
After parsing, each log line is represented as a tuple of *timestamp*, *block ID*, and *event type*, as shown at the bottom of Figure 1. The event type is commonly referred to as a *token*. In this example, the token "d38aa58d" represents the event described in the original log statement, and this event occurs at the data block indicated by the block ID. The failures in the HDFS system are primarily associated with data blocks, so logs related to the same block were grouped and treated as a whole. The grouped logs take the form of a token sequence as illustrated in Figure 2. Note that Figure 2 describes only one sequence. It has a label of either regular – 0 or abnormal – 1, corresponding to a block's health condition. The sequences and their labels form the samples for training a classification model.

After sequencing, the original 11.2-million-line log file is converted into a dataset containing 575,061 labeled samples like the one shown in Figure 2. These sequences consist of 48 unique tokens, representing 48 different log events. The dataset is highly skewed, with only 16,838 abnormal samples, less than 2.93% of the total dataset.

Examining the token sequence only, most of all 575,061 samples have sequence lengths between 10 and 40 tokens with an average of 19. The length histogram of all sequence samples is shown at the top of Figure 3. Some outliers with lengths up to 298 are not displayed in the graph. There are no evident distribution patterns that can be seen from this graph.

If considering timestamps, the duration of log sequences varies from 0.5 to 54,000 seconds, where a majority of them lie within 120 seconds. The histogram of sequence durations at the bottom of Figure 3 now shows a clear bimodal pattern. Most of the samples follow a Gaussian distribution with a mean of 40 seconds. A smaller amount of short-lived sequences cluster around the very left of the graph.

Figure 4 further examines the distribution difference between regular and abnormal samples. The duration histogram of regular samples at the top graph clearly shows two Gaussian patterns with means at 3 seconds and 40 seconds. In contrast, the abnormal samples on the bottom graph are generally shorter and fall within the 0 to 10 seconds range. Longer abnormal samples with a duration of more than 10 seconds do exist, but they are relatively few and barely distinguishable on this graph.

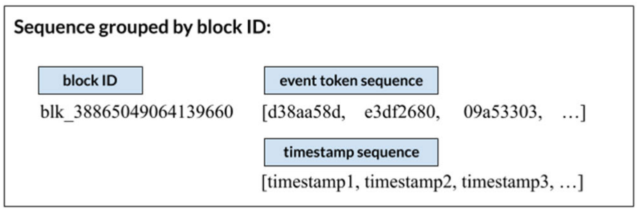


```
081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1
for block blk_38865049064139660 terminating
```

```
08-11-09 20:36:15    blk_38865049064139660    d38aa58d
```

FIGURE 1. An example of parsing HDFS logs.

The distribution difference between sample lengths and sample durations shows a correlation between time information and the temporal dynamics of log sequences. The different patterns between regular and abnormal samples further demonstrate that time information can potentially be utilized to detect a failure. The following section discusses methods to utilize time information.


Sequence grouped by block ID:

block ID
blk_38865049064139660

event token sequence
[d38aa58d, e3df2680, 09a53303, …]

timestamp sequence
[timestamp1, timestamp2, timestamp3, …]

FIGURE 2. An example of a tokenized log sequence.

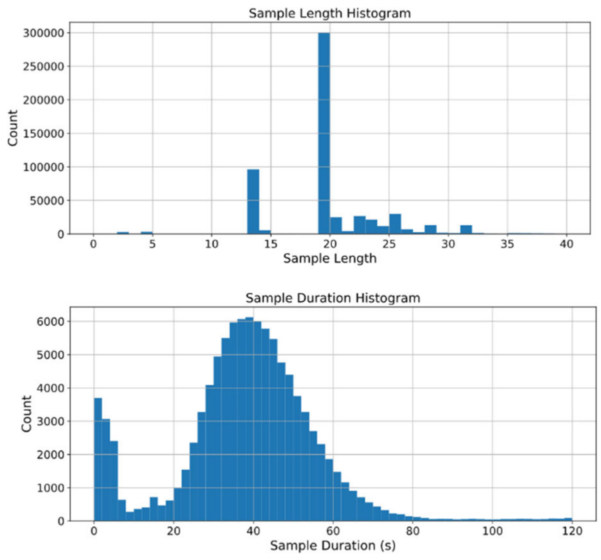


FIGURE 3. Histogram of sample length (top) and sample duration (bottom).

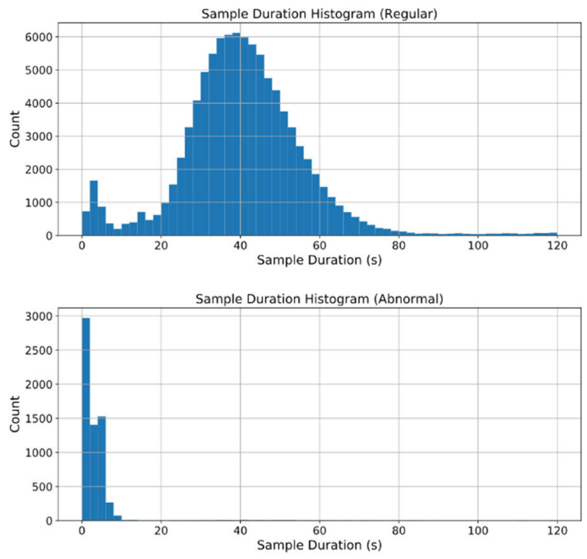


FIGURE 4. Histogram of sample durations by labels (top: regular samples, bottom: abnormal samples).

IV. METHODOLOGY

This study proposes an automated failure detection system as illustrated in Figure 5. As a learning system, the training process is shown on the top and the detection process is on the bottom. The original log messages go through a parsing process to obtain token sequences as described in Section III. The labeled token sequences are used to train a deep learning classification model in the training process. In the detection process, the trained model takes the token sequences as input and produces detection results. The classification model contains four main components: the embedding layer, the novel resample layer, the sequential backbone layer, and the full-connection layer. The novel resample layer enables timestamp integration into the model. It can be bypassed, and the model functions as a regular sequence classification model. This section presents these components in detail.

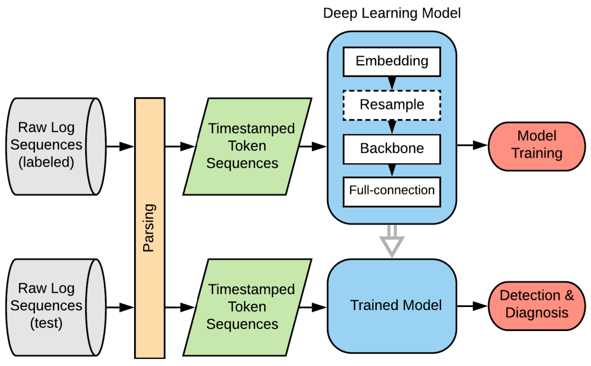


FIGURE 5. Overview of the failure detection model.

Subsection A starts with the detection model framework without timestamp integration (referred to as the *base models*). Subsection B describes a novel network structure to integrate timestamps by interpolation. Subsection C elaborates interpolation methods and selects feasible ones for implementation.

A. THE BASE MODELS

Two promising candidates for the deep learning model are RNN and CNN. The sequential nature of log data makes the RNN models [41] particularly suitable for this application. In an RNN, the outputs or hidden layer values of a network are fed back and merged with inputs for the next iteration. In this way, an RNN preserves the temporal information of a sequence from the first element to the last. In particular, the Long Short-Term Memory (LSTM) model [42], a variation of RNN, is used in this study as it is proven to be one of the best performing RNNs [1]. The uniqueness of LSTM is on the microscopic level: it formulates a series of gate functions within the hidden unit to throttle long-term dependencies. The high-level network constructions of a general RNN and an LSTM model are the same.

Another way to process sequential inputs is to use CNN. Since first introduced in 1998 [43], CNN quickly became the most popular algorithm for image classification tasks. The CNN utilizes a number of convolution kernels that scan a 2D image to search for distinguishable patterns. This study uses a variation of CNN where the kernels move along one direction (1D-CNN) to search for sequence patterns.

Figure 6 illustrates the base model structures using RNN and CNN as the backbone, respectively. The blue blocks represent network layers, and the green blocks represent input and intermediate data. The RNN base model is shown at the top. It takes token sequences previously shown in Figure 3 as input. The tokens first go through an *embedding* process to obtain the vectorized representation. This process converts individual log tokens into numerical vectors, an inspiration from language modeling research [29]. These embedding vectors capture the correlations among tokens in a high-dimensional space. The embedding process is effectively a fully connected network that maps a log token in one-hot vector form to an $N \times 1$ embedding vector. The parameters of the embedding process are obtained using the word2vec method [44] in an unsupervised manner using all available data except the data used for testing.

The embedding vectors in the form of an $N \times M$ matrix, where M is the length of the sequence, are then fed into the RNN/LSTM backbone. Here the RNN uses a many-to-one structure and outputs a 1D vector y. The top of Figure 6 illustrates a two-layer RNN structure. A full-connection neural network then processes the vector y to get the final output. The output layer has a softmax function to obtain a probability distribution among the two labels, i.e. 0 – regular, 1 – abnormal.

In the CNN base model illustrated at the bottom of Figure 6, the embedding process is the same as the RNN base model and produces an $N \times M$ matrix. Kernels of the CNN's first convolutional layer have a width of N and scan the input vectors along the timesteps. Multiple convolutions can be stacked together to increase non-linearity and enable the detection of more complex feature patterns. The output is flattened to a 1D vector and passed through a full-connection network, similar to the one in the RNN model but having different configurations.

B. TIMESTAMP INTEGRATION

The embedding vectors explained in the previous section represent sequential orders of log data, but do not include time information. Since the log data have nonuniform intervals, ignoring timestamps will likely result in a representation with altered temporal characteristics. Also, because of the significance of time duration – as explained in Section III – it is important to include timestamps into the detection model. This section proposes the timestamped model (Ts model), a deep learning structure that integrates timestamps. Ts models use interpolation to resample the log sequences to create evenly spaced data points, an insight from digital signal processing.

It is evident that interpolation methods only apply to sequences in numerical values. Since the log token sequences are categorical, the resample cannot directly apply to these sequences at the model's input. Fortunately for deep learning models, the embedding process represents log tokens as numerical data points in a vector. The embedding sequence can be viewed as a special signal consisting of these data points along the time dimension. Resampling these vector sequences and creating new datapoints by interpolation is therefore feasible and reasonable. In other words, resampling token sequences is irrational, but resampling can be applied to the embedding sequences that contain meaningful numerical values.

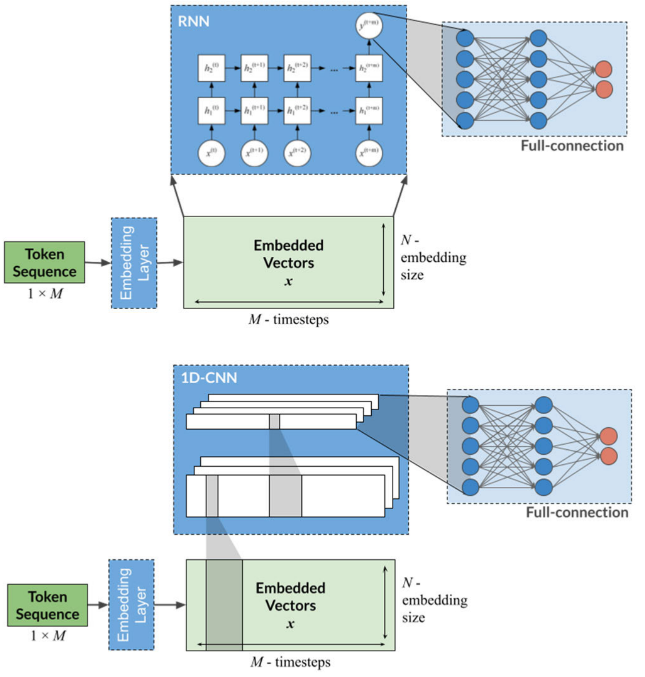


FIGURE 6. The structure of base models (top: the RNN base model, bottom: the CNN base model).

The new model structure incorporating the resample layer is shown in Figure 7. The Ts model's inputs are log token sequences with length M as well as the timestamps. The token sequence is first converted into embedding vectors, the same as the base model. Then, a resample layer applies interpolation to every row of the embedding matrix, giving an $N \times K$ output, that is the interpolated vectors. The value of K is determined by the configuration of the resample process, such as the sampling resolution and maximum duration. An RNN or CNN network then processes the interpolated vectors, similar to the settings in the base models. The output side of the Ts model has the same fully connected network as the base models as shown in Figure 6.

C. INTERPOLATION METHODS

This subsection takes one row of the embedding vector as an example to explain the difference and to choose among various interpolation methods. Interpolation refers to the up-sampling process of creating intermediate data points from a given time signal. For a one-dimensional input time series (t, x):

$$t[M] = [t_0, t_1, \ldots, t_{M-1}]$$
$$x[M] = [x_0, x_1, \ldots, x_{M-1}]$$

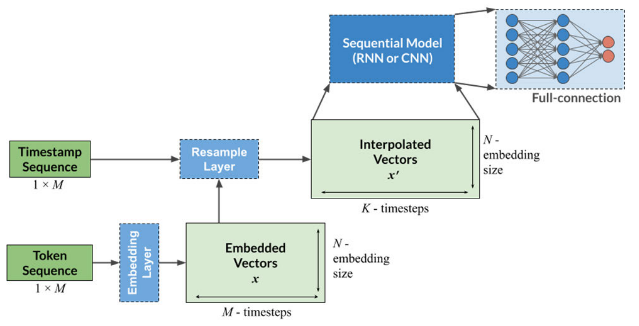


FIGURE 7. **The Ts model structure with timestamp integration.**

where t is the timestamp sequence, x is the data sequence, and M is the length of both sequences. An interpolation solves for a new series (t', x'):

$$t'[K] = \left[t'_0, t'_1, \ldots, t'_{K-1}\right]$$
$$x'[K] = \left[x'_0, x'_1, \ldots, x'_{K-1}\right]$$

where K is the length of the interpolated sequence.

The original sequence tuple (t, x) has nonuniform intervals in this application. The output tuple $\left(t', x'\right)$ preferably has equal interval T_s, a.k.a., the sampling rate. The interpolation is achieved by solving a function $P(t)$ that goes through all input data points (t_i, x_i). There are several interpolation methods for numeric sequences [45]:

1. Zero-Order Hold (ZOH), or piecewise-constant interpolation, is the simplest and fastest way to even out the timestamps. It takes the closest sequence data point as the output. The interpolation result is a discrete function expressed as follows:
$$P(t) = x_i, \quad if\ t_i < t < t_{i+1} \tag{1}$$
2. Linear interpolation is a local interpolation method that calculates new sample points using two adjacent data points. It produces a continuous function, although its first-order derivative is still discrete:
$$P(t) = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} t + \frac{x_i t_{i+1} - x_{i+1} t_i}{t_{i+1} - t_i}, \quad if\ t_i < t < t_{i+1} \tag{2}$$
3. For improved smoothness, cubic spline interpolation is another local interpolation option that fits a series of third-order polynomials (splines) $s_i(t)$ using adjacent data points, while ensuring the whole function has continuous second-order derivatives at all input data points:
$$P(t) = s_i(t), \quad if\ t_i < t < t_{i+1} \tag{3}$$
where the polynomial $s_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ satisfies:
$$\begin{aligned} s_i(t_{i-1}) &= x_{i-1} \\ s_i(t_i) &= x_i \\ s'_i(t_i) &= s'_{i+1}(t_i) \\ s''_i(t_i) &= s''_{i+1}(t_i) \end{aligned} \tag{4}$$
4. Lagrange interpolation achieves continuity for all orders of derivatives by fitting a polynomial function with an order of $M - 1$. One way to obtain the polynomial function is as follows:
$$P(t) = \sum_{i=0}^{M-1} x_i L_i(t) \tag{5}$$
where $L_i(t)$ are the Lagrange basis polynomials:
$$L_i(t) = \prod_{j=0, j\neq i}^{M-1} \frac{t - t_i}{t_i - t_j} \tag{6}$$

After obtaining the interpolated function $P(t)$, the resampled sequence $x'[K]$ is calculated by:

$$x'_i = P\left(t'_i\right), \quad i = 0, 1, \ldots, K-1 \tag{7}$$

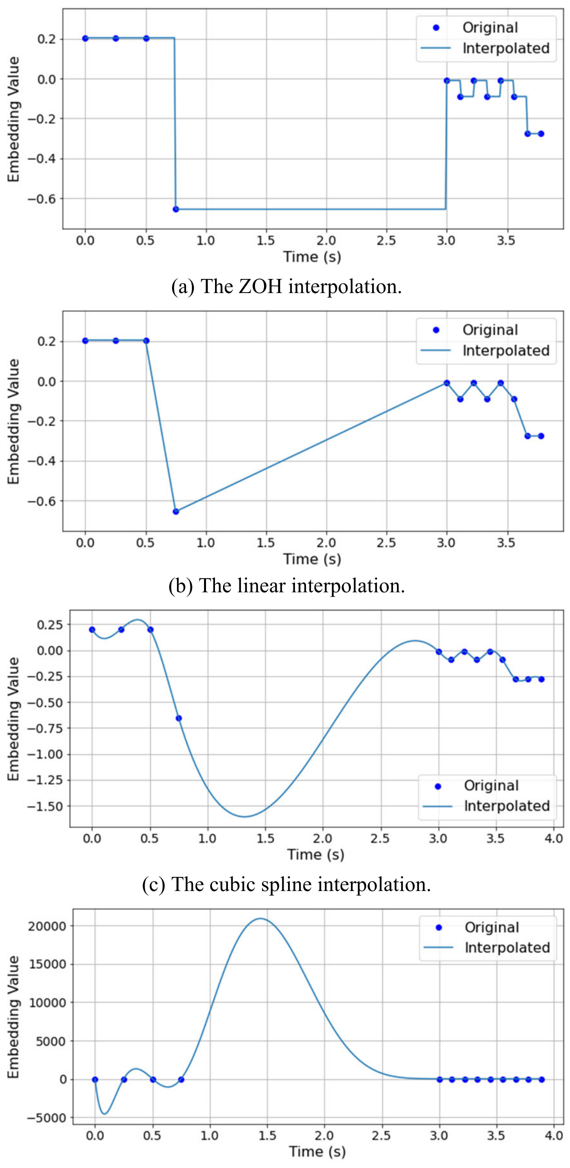
The set of graphs in Figure 8 shows the implementation of interpolating a short embedding sequence from the HDFS dataset. Figures 7(a) through (d) correspond to the results of using ZOH, linear, cubic spline, and Lagrange methods, respectively. The ZOH interpolation in Figure 8(a) produces a discrete function that contains mainly staircase patterns. Linear interpolation in Figure 8(b) gives a continuous function, showing improved smoothness over the ZOH method. Visually this reveals more varied features than Figure 8(a). The cubic spline interpolation in Figure 8(c) appears very smooth, and the values of splines are within a reasonable range. The Lagrange interpolation in Figure 8(d) shows similar smoothness as Figure 8(c) but does not seem to bring more distinct features. The Lagrange method creates unexpectedly large values (note the range of y-axis), which is reasonable mathematically but may be impractical.

Among the three methods, the ZOH and linear interpolation are selected for the resample layer of the Ts models. The reasons for not choosing the other two methods are as follows. The Lagrange method's unexpectedly large values are unfavorable as they may cause unstable neural network training. The curves produced by the cubic spline interpolation look promising, but the computation cost is a practical problem. It requires solving an $M \times M$ tridiagonal linear system, which is significantly more complex than the linear method, even with efficient algorithms [46]. The computational requirement could be an issue for the linear method, as concluded in Section VII, so the cubic spline method is not chosen.

Last but not least, the resample layer is implemented in the batch form such that each embedded matrix containing N sequences can be processed in one go. Coding in matrix form also enables parallel computation of the deep learning platform (TensorFlow) and speeds up the model execution.

V. IMPLEMENTATION DETAILS

The failure detection system and all deep learning models are implemented in Python using TensorFlow packages on Windows 10. Training the log embeddings is done separately using *gensim*, a language modeling package for Python.



(a) The ZOH interpolation.

(b) The linear interpolation.

(c) The cubic spline interpolation.

(d) The Lagrange interpolation.

FIGURE 8. The comparison of different interpolation methods using the same data sequence.

The hardware platform includes one Nvidia graphics card and 32GB memory.

A. EXPERIMENT SETUP

The system examined in this study is the HDFS introduced in Section III. HDFS runs on a central node (a server) that manages a cluster of data nodes (commodity computers). The data nodes' hardware is prone to failure, and detecting them in a timely manner ensures service availability. System logs that record the communication between nodes are often the key to diagnosing a node failure.

The automated failure detection system proposed in this study aims to provide the basis for an online service that would be capable of continuously monitoring the system log data. Failure detection can be achieved by using log messages generated within a time window, treated as a sequential sample. The detection system in Figure 1 takes this sequence sample as input and produces a detection result. In real-world settings, the proposed detection models can run within the HDFS server. Alternatively, dedicated computing hardware can process the data in parallel to the server.

In this study, the purpose is to evaluate different model structures and the effectiveness of timestamps. Therefore, the same pre-recorded log data is used for every test to ensure a fair comparison. Section III provides an overview of the dataset and its preprocessing. To simulate the online approach, a time window of maximum duration T_{max} is used as the input log sequence samples. Since the histograms in Section III reveal that the typical duration of a log sequence is less than 120 seconds, the default T_{max} is set to this value. Section VI evaluates and discusses the effects of varying T_{max}.

The selected samples are shuffled and divided into training, validation, and testing sets, with a ratio of 16:4:5. The regular/abnormal class ratio is the same across the three subsets. As mentioned in Section III, the HDFS dataset is highly skewed. If training using the original class ratio where regular samples are significantly more than abnormal ones, the model will bias towards more regular classification. In order to achieve optimal results for both classes, the training dataset needs to be balanced. This study applies the oversampling technique to abnormal samples to ensure that the model sees an equal amount of positive and negative samples during training. The validation and testing sets keep the original class ratio.

B. MODEL CONFIGURATIONS AND HYPER-PARAMETERS

Six models are configured using the structures presented in Figures 6 and 7. The following abbreviations refer to these six models:

1. RNN: The base model using LSTM layers.
2. CNN: The base model using CNN layers.
3. Ts-RNN-ZOH: The timestamp model using ZOH interpolation and LSTM layers.
4. Ts-CNN-ZOH: The timestamp model using ZOH interpolation and CNN layers.
5. Ts-RNN-Lin: The timestamp model using linear interpolation and LSTM layers.
6. Ts-CNN-Lin: The timestamp model using linear interpolation and CNN layers.

Table 1 shows the model configurations. Ts models using different interpolation methods are combined since their structures are exactly the same at a high level. Each table from top to bottom lists the layers for a forward pass of the network. The original RNN and CNN models both have one input sequence $\boldsymbol{x}$ with a length of 250. The Ts models have one additional input – timestamp sequence $\boldsymbol{ts}$ – and have one additional resample layer than their base models. The output is a value representing the probability of failure. The total weights at the bottom show the number of trainable/total parameters in each model. Ts models have the same amount of weights as the original model since the additional resample layer does not add new weights.

Each row in the table represents a layer in the network structure, corresponding to a block in Figures 6 and 8. All models start with an embedding layer that converts the log sequences in token forms into numerical vectors. The Ts models include an additional resample layer between the embedding layer and the RNN/CNN layers.

The RNN models utilize one LSTM layer coupled with two full-connection layers to perform the classification. In comparison, the CNN models use one convolutional layer, a max-pooling layer, and three full-connection layers. The configurations of these layers are the result of hyperparameter tuning, an iterative process of finding optimal model settings. The hyperparameter tuning process uses the validation set to compare the performance of different configurations.

Note that the network is not particularly "deep," indicating that the patterns within this log dataset are relatively simple. Stacking multiple CNN layers or LSTM layers is feasible; however, the tuning process found that the models tend to overfit and produce no significant improvements with more layers.

VI. EVALUATION

Evaluation tests are designed to address the following questions: 1) whether the timestamp integration improves over the base models and how much is the improvement, 2) how the time duration of log sequences affects the models' performance and, 3) how repeatable the results are in terms of model training.

This section is structured as follows. Subsection A evaluates all proposed models using log samples with a duration of 120s or less. Then Subsection B includes longer samples. The tests in Subsection C demonstrate the models' repeatability in terms of training. Subsection D discusses the results and compares them with existing studies.

A. RESULTS WITH DEFAULT SAMPLE DURATION

In this test, all six models are trained and evaluated against the same test dataset with a duration threshold $T_{max} = 120$. For each sample, the models produce an output score between 0 and 1, representing the probability of a failure. The default threshold of 0.5 is used to determine whether the detection is normal – 0 or abnormal – 1.

Table 2 shows the performance of the six models at the default threshold in terms of accuracy, precision, recall, F1 score, and Matthews Correlation Coefficient (MCC). In particular, MCC is included for a reliable evaluation due to the data being highly skewed. MCC has a range of $[-1, 1]$ and all other metrics have $[0, 1]$. For all metrics, the higher value indicates better performance. Note that the values in Table 2 are not definitive since the model's final weights are subject to variance due to random factors during the training process, such as initialization and dropout. The test results displayed in this table are the ones most reproducible.

TABLE 1. The model configurations.

RNN

Layer	*Configuration*	*Output shape*
Input		(250)
Embedding	16 units	(250, 16)
LSTM layer	64 units, tanh activation	(64)
Full-connection	32 units, tanh activation	(32)
Output layer	1 unit, sigmoid activation	(1)
Total weights:	22,849 / 23,633	

Ts-RNN (ZOH and Linear)

Layer	*Configuration*	*Output shape*
Input		(250), (250)
Embedding	16 units	(250, 16), (250)
Resample	Resolution = 0.1s	(1001, 16)
LSTM layer	64 units, tanh activation	(64)
Full-connection	32 units, tanh activation	(32)
Output layer	1 unit, sigmoid activation	(1)
Total weights:	22,849 / 23,633	

CNN

Layer	*Configuration*	*Output shape*
Input		(250)
Embedding	16 units	(250, 16)
1D convolution	32 kernels, size=4, stride=1	(81, 32)
Global max pooling		(32)
Full-connection	32 units, ReLU activation	(32)
Full-connection	32 units, ReLU activation	(32)
Output layer	1 unit, sigmoid activation	(1)
Total weights:	6,273 / 7,057	

Ts-CNN (ZOH and Linear)

Layer	*Configuration*	*Output shape*
Input		(250), (250)
Embedding	16 units	(250, 16), (250)
Resample	Resolution = 0.1s	(1001, 16)
1D convolution	32 kernels, size=8, stride=3	(332, 32)
Global max pooling		(32)
Full-connection	32 units, ReLU activation	(32)
Full-connection	32 units, ReLU activation	(32)
Output layer	1 unit, sigmoid activation	(1)
Total weights:	6,273 / 7,057	

According to the table, all models can achieve high accuracy close to 100%. The F1 score shows a slightly better distinction: the RNN, CNN, and Ts-CNN are the best performing among all models. The timestamps integration did not show a significant effect in this test. It may be due to the fact that overall performance is already at a very high level. The difference among models is insignificant to draw meaningful conclusions.

B. RESULTS WITH LONGER SEQUENCES

This test chooses a list of T_{max} values ranging from 120 seconds to 1920 seconds in order to examine how the

TABLE 2. Model performance evaluated on the test set.

Models	Metrics				
	Accuracy	*Precision*	*Recall*	*F1 score*	*MCC*
RNN	0.9996	0.9946	0.9992	**0.9969**	0.9967
CNN	0.9996	0.9946	0.9984	**0.9965**	0.9963
Ts-RNN-ZOH	0.9970	0.9781	0.9743	0.9762	0.9752
Ts-CNN-ZOH	0.9987	0.9794	1.0000	0.9896	0.9963
Ts-RNN-Lin	0.9971	0.9936	0.9603	0.9766	0.9746
Ts-CNN-Lin	0.9996	0.9946	0.9984	**0.9965**	0.9890

TABLE 3. F1 scores with different T_{max}.

T_{max}	Metrics – F1 Score					
	RNN	*CNN*	*Ts-RNN-ZOH*	*Ts-CNN-ZOH*	*Ts-RNN-Lin*	*Ts-CNN-Lin*
120s	0.9969	0.9965	0.9762	0.9896	0.9766	**0.9965**
240s	0.9780	0.9890	0.9754	**0.9931**	0.9514	**0.9931**
480s	0.9551	0.9859	0.9542	**0.9902**	0.9202	**0.9905**
960s	0.9364	0.9632	0.9421	**0.9888**	0.9554	**0.9884**
1920s	0.9169	0.9519	0.9200	**0.9840**	0.9279	**0.9840**

duration of sequence samples affects model performance. Tables 3 and 4 shows the F1 score and MCC, which are the more balanced metrics. All six models see a decrease in performance as the sample duration increases, meaning that longer sequences are more difficult to analyze. In the meantime, two Ts-CNN models – Ts-CNN-ZOH and Ts-CNN-Lin – have the least decrement, producing the highest F1 score and MCC among the six for longer sequence durations. In each row, the Ts models perform differently from their base models – increase for most cases – meaning that the timestamp integration is at work. The contribution is positive and distinct in the case of CNN and Ts-CNN models, while less clear for the RNN and Ts-RNN models. Both F1 score and MCC reveal the same observation.

Depending on the choice of base models, ZOH and linear interpolation have different effects on the performance. Ts-RNN-ZOH gives an overall better performance than the Ts-RNN-Lin. On the other hand, Ts-CNN with linear interpolation is a better choice than the ZOH interpolation, specifically because of the test at 120s. Overall, the Ts-CNN-Lin is the best performing model across all tests.

C. REPRODUCIBILITY

Three of the best-performing model configurations in Table 3, the CNN, Ts-CNN-ZOH, and Ts-CNN-Lin, are re-trained and evaluated ten times using the same dataset. The purpose is to examine the performance variation during the training. Each time the model is re-initialized with random weights. Figure 9 shows their F1 scores in box plots. The base CNN and Ts-CNN-Lin are placed side by side on the top graph; the Ts-CNN-ZOH and Ts-CNN-Lin are on the bottom. In these graphs, the boxes represent the interquartile range, including the data points whose values are between 75th and 25th percentiles. The whiskers show the upper and lower 25th percentiles, except for outliers indicated by individual dots. The median performance is highlighted with a black cross and connected with dashed lines.

TABLE 4. MCC with different T_{max}.

T_{max}	Metrics – MCC					
	RNN	*CNN*	*Ts-RNN-ZOH*	*Ts-CNN-ZOH*	*Ts-RNN-Lin*	*Ts-CNN-Lin*
120s	0.9967	0.9963	0.9746	0.9889	0.9752	**0.9963**
240s	0.9769	0.9862	0.9738	**0.9927**	0.9483	**0.9927**
480s	0.9536	0.9851	0.9524	**0.9896**	0.9154	**0.9900**
960s	0.9353	0.9618	0.9408	**0.9882**	0.9540	**0.9879**
1920s	0.9168	0.9505	0.9194	**0.9834**	0.9245	**0.9834**

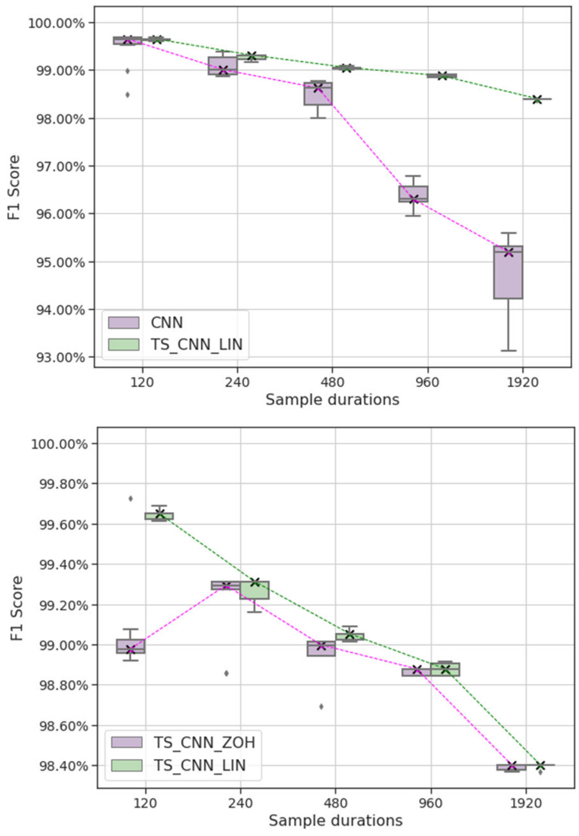


FIGURE 9. Model performance of repeated tests (top: CNN vs. Ts-CNN-Lin, bottom: Ts-CNN-ZOH vs. Ts-CNN-Lin).

From Figure 9, the top graph clearly demonstrates the improvement brought by timestamp integration. Ts-CNN-Lin shows higher median, better best-case and worst-case performances than the base CNN model under all duration settings. Note that the variation caused by training initialization is also suppressed with the Ts models. In the bottom graph, the linear interpolation has an overall advantage over the ZOH interpolation, though the difference is marginal other than for the 120s case. It can be concluded that the timestamps contain valuable information that contributes to failure detection. This contribution can be successfully captured by Ts-CNN models, especially with linear interpolation.

TABLE 5. Several existing models' performance.

Models	Metrics		
	Precision	*Recall*	*F1 score*
Logistic [4]	0.95	1.00	0.98
SVM [4]	0.95	1.00	0.98
CNN in [35]	0.977	0.993	0.985
DeepLog [6]	0.95	0.96	0.96
Ts-CNN-Lin *(120s case)*	0.9938	0.9992	0.9965
Ts-CNN-Lin *(1920s case)*	0.9692	0.9993	0.9840

D. COMPARISON WITH SELECT NUMBER OF PUBLISHED METHODS

There are a few previous publications on system failure detection using the HDFS dataset [4], [6], [35]. The performances of four of the most recent machine learning methods are compared in Table 5. The numbers are directly referenced from the papers, so they have different decimal points. The first two methods use feature extraction with machine learning [4] and a fixed window size of 1 hour. The third model [35] uses a different CNN strategy with more convolutional layers than the CNN introduced in Section IV.A. The last model, DeepLog [6], is a real-time detector based on LSTM.

From this table, an F1 score of 0.98 would be a reasonable baseline to choose with 0.985 being the highest amongst the four. The Ts-CNN-ZOH and Ts-CNN-Lin models presented in this paper are able to achieve the highest-level performance. In particular, the CNN in [35] achieved its high performance through refining its network architecture, while Ts-CNN models did so only by including the timestamp information. Moreover, at a sample duration of 120s, the Ts-CNN-Lin models can exceed an F1 score of 0.995, whereas existing studies did not investigate the effect of log durations. Also note that the Ts-CNN results included repeatability tests and emphasized the median performance, whereas the above studies did not report on test settings. In conclusion, the Ts-CNN models presented in this paper can achieve and improve upon their performance.

The comparison between the Ts models and the regular models in this section shows that the inclusion of timestamps can significantly improve the performance of an existing model. In particular, Ts-CNN-Lin, the model using linear interpolation, demonstrates a higher level of F1 scores and a lower level of variation in repeated tests. None of the existing studies have systematically investigated the benefits of timestamp integration. The DeepLog study attempted to include the timestamps [6]. Their approach was to handle timestamps in an independent process separated from the log token model. The paper did not quantify the timestamp contribution to the failure detection performance.

VII. CONCLUSION

This study investigated the effect of timestamps in log data and their contribution to the system failure detection task. Log data generally have nonuniform time intervals, but few existing studies have included them in automated analysis. For the first time, statistical observations reveal that the time information affects the log sequences' distribution characteristics. In particular, regular and abnormal samples demonstrate a clear distribution difference.

To further investigate the effect of timestamps, the paper first proposes and implements a deep-learning-based failure detection framework, including both CNN and LSTM variations. Then, the novel Ts models consider the nonuniform time intervals by applying interpolation at the embedding level. Various interpolation methods are discussed, and two of them, the ZOH and linear methods, are selected for implementation and evaluation.

The evaluation results prove that the timestamps provide several benefits. First, Ts models show a general improvement over the base models in terms of accuracy and F1 scores. Second, Ts models demonstrate smaller performance variance in terms of training repeated tests. Last, deep learning models generally show reduced performance when processing longer log samples, and such tendencies can be curbed by integrating timestamps. In particular, the proposed Ts-CNN-Lin model showed the best performance among all models tested for all metrics, achieving top-level performance among several machine learning and deep learning studies. These results confirm that the timestamps contain useful information that helps to improve failure detection, and the proposed Ts models can successfully capture such information. This finding has the potential to reduce service downtime and improve system maintainability.

There are limitations to the proposed methods. The first limitation is the computational requirements. The model has a fixed input size once it is created, meaning that it may not process an arbitrarily long log sequence in one pass. If analysis requires longer log durations, the model needs to size up and be retrained, and computation time and memory usage will increase. The hardware is likely to become a bottleneck. Cubic spline interpolation was not implemented due to this computational capability concern. Another limitation is that deep learning approaches generally require dedicated hardware with parallel computing capability, e.g., a GPU with parallel computing capability. Otherwise, the model may not run efficiently and will take more time to process.

ACKNOWLEDGMENT

Yixin Huangfu would like to thank Michael Kelly for proofreading this article.

REFERENCES

[1] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," 2020, *arXiv:2009.07237*.

[2] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. Int. Conf. Autonomic Comput.*, New York, NY, USA, 2004, pp. 36–43.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, ''Isolation forest,'' in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.

[4] S. He, J. Zhu, P. He, and M. R. Lyu, ''Experience report: System log analysis for anomaly detection,'' in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 207–218.

[5] A. Das, F. Mueller, C. Siegel, and A. Vishnu, ''Desh: Deep learning for system health prediction of lead times to failure in HPC,'' in *Proc. 27th Int. Symp. High-Perform. Parallel Distrib. Comput.*, Jun. 2018, pp. 40–51.

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, ''DeepLog: Anomaly detection and diagnosis from system logs through deep learning,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 1285–1298.

[7] A. Oliner and J. Stearley, ''What supercomputers say: A study of five system logs,'' in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 575–584.

[8] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou, ''Understanding customer problem troubleshooting from storage system logs,'' in *Proc. 7th Conf. File Storage Technol.*, New York, NY, USA, Feb. 2009, pp. 43–56.

[9] J. E. Prewett, ''Analyzing cluster log files using logsurfer,'' presented at the 4th Annu. Conf. Linux Clusters, 2003.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[11] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, ''Log clustering based problem identification for online service systems,'' in *Proc. 38th Int. Conf. Softw. Eng. Companion*, May 2016, pp. 102–111.

[12] G. Dong and J. Pei, *Sequence Data Mining*. New York, NY, USA: Springer, 2007.

[13] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Amsterdam, The Netherlands: Elsevier, 2012.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, ''Detecting large-scale system problems by mining console logs,'' in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ. (SOSP)*, New York, NY, USA, 2009, pp. 117–132.

[15] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, ''Drain: An online log parsing approach with fixed depth tree,'' in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.

[16] J. Zhu, S. He, J. Liu, P. He, and Q. Xie, ''Tools and benchmarks for automated log parsing,'' in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2019, pp. 121–130.

[17] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, ''Mining invariants from console logs for system problem detection,'' in *Proc. USENIX Annu. Tech. Conf.*, 2010, pp. 1–14.

[18] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, ''Fingerprinting the datacenter: Automated classification of performance crises,'' in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 111–124.

[19] K. S. Jones, ''A statistical interpretation of term specificity and its application in retrieval,'' *J. Document.*, vol. 28, no. 1, pp. 11–21, Jan. 1972, doi: 10.1108/eb026526.

[20] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, ''Automated IT system failure prediction: A deep learning approach,'' in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 1291–1300.

[21] Y. Zhang and A. Sivasubramaniam, ''Failure prediction in IBM BlueGene/L event logs,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Omaha, NE, USA, Apr. 2008, pp. 583–588.

[22] I. Beschastnikh, Y. Brun, M. D. Ernst, A. Krishnamurthy, and T. E. Anderson, ''Mining temporal invariants from partially ordered logs,'' in *Proc. Manag. Large-Scale Syst. Via Anal. Syst. Logs Appl. Mach. Learn. Techn. (SLAML)*, New York, NY, USA, 2011.

[23] K. Nagaraj, C. Killian, and J. Neville, ''Structured comparative analysis of systems logs to diagnose performance problems,'' presented as the 9th USENIX Symp. Netw. Syst. Design Implement., 2012, pp. 353–366.

[24] M. C. Dani, H. Doreau, and S. Alt, ''K-means application for anomaly detection and log classification in HPC,'' in *Advances in Artificial Intelligence: From Theory to Practice*. Cham, Switzerland: Springer, 2017, pp. 201–210.

[25] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, ''Abstracting execution logs to execution events for enterprise applications (Short Paper),'' in *Proc. 8th Int. Conf. Quality Softw.*, Aug. 2008, pp. 181–186.

[26] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, ''Execution anomaly detection in distributed systems through unstructured log analysis,'' in *Proc. 9th IEEE Int. Conf. Data Mining*, Dec. 2009, pp. 149–158.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ''ImageNet classification with deep convolutional neural networks,'' in *Proc. Adv. Neural Inf. Process. Syst.*, F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105.

[28] O. Russakovsky, J. Deng, H. Su, and J. Krause, ''ImageNet large scale visual recognition challenge,'' *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[29] M. Sundermeyer, R. Schlüter, and H. Ney, ''LSTM neural networks for language modeling,'' in *Proc. 13th Annu. Conf. Int. Speech Commun. Assoc.*, 2012, pp. 194–197.

[30] A. Graves, N. Jaitly, and A.-R. Mohamed, ''Hybrid speech recognition with deep bidirectional LSTM,'' in *Proc. IEEE Workshop Autom. Speech Recognit. Understand.*, Dec. 2013, pp. 273–278.

[31] R. Ahmed, M. El Sayed, S. A. Gadsden, J. Tjong, and S. Habibi, ''Automotive internal-combustion-engine fault detection and classification using artificial neural network techniques,'' *IEEE Trans. Veh. Technol.*, vol. 64, no. 1, pp. 21–33, Jan. 2015.

[32] E. Chemali, P. J. Kollmeyer, M. Preindl, R. Ahmed, and A. Emadi, ''Long short-term memory networks for accurate state-of-charge estimation of Li-ion batteries,'' *IEEE Trans. Ind. Electron.*, vol. 65, no. 8, pp. 6730–6739, Aug. 2018.

[33] Y. Zuo, Y. Wu, G. Min, C. Huang, and K. Pei, ''An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis,'' *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 2, pp. 548–561, Jun. 2020.

[34] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, ''Robust log-based anomaly detection on unstable log data,'' in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 807–817.

[35] S. Lu, X. Wei, Y. Li, and L. Wang, ''Detecting anomaly in big data system logs using convolutional neural network,'' in *Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervasive Intell. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 151–158.

[36] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, ''Anomaly detection using autoencoders in high performance computing systems,'' in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, Jul. 2019, pp. 9428–9433.

[37] L. Tang, T. Li, and C.-S. Perng, ''LogSig: Generating system events from raw textual logs,'' in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, New York, NY, USA, 2011, pp. 785–794.

[38] M. Du and F. Li, ''Spell: Streaming parsing of system event logs,'' in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 859–864.

[39] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, ''LogMine: Fast pattern recognition for log analytics,'' in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, 2016, pp. 1573–1582.

[40] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, ''An evaluation study on log parsing and its use in log mining,'' in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016, pp. 654–661.

[41] S. S. Haykin, *Neural Networks and Learning Machines*, vol. 3. Upper Saddle River, NJ, USA: Pearson, 2009.

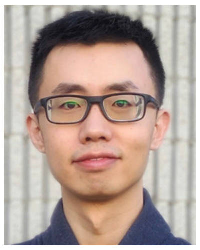
[42] S. Hochreiter and J. Schmidhuber, ''Long short-term memory,'' *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, ''Gradient-based learning applied to document recognition,'' *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean, ''Efficient estimation of word representations in vector space,'' 2013, *arXiv:1301.3781*.

[45] P. Prandoni and M. Vetterli, *Signal Processing for Communications*, 1st ed. Lausanne, Switzerland: EPFL Press, 2008.

[46] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, 2nd ed. Berlin, Germany: Springer-Verlag, 2007.



YIXIN HUANGFU (Member, IEEE) received the B.Sc. degree in automotive service engineering from the Wuhan University of Technology, Wuhan, China, in 2011, and the M.Sc. degree in mechanical engineering from the Beijing Institute of Technology, Beijing, China, in 2014. He is currently pursuing the Ph.D. degree in mechanical engineering with McMaster University, Hamilton, ON, Canada, under the supervision of Dr. Saeid Habibi.

From 2014 to 2017, he was a Vehicle Control Engineer with Ford Motor Research and Engineering Company Ltd., Nanjing, China. One of his papers, "A Novel Method for Approximating Object Location Error in Bounding Box Detection Algorithms Using a Monocular Camera," is published on IEEE Transactions on Vehicular Technology, in 2021. His research interests include environment perception systems on autonomous vehicles, image recognition and classification with neural networks, sequential data analysis, and data-driven fault diagnosis systems.

SAEID HABIBI (Member, IEEE) is a Professor and the former Chair of the Department of Mechanical Engineering, McMaster University, and holds the Senior Industrial Research Chair position at Hybrid Technologies sponsored by NSERC and Ford, Canada. His research interests include intelligent control, state and parameter estimation, fault diagnosis and prediction, variable structure systems (VSS), actuation systems, mechatronics, and fluid power. Application areas include automotive, aerospace, water distribution, and robotics. He developed the smooth variable structure filter (SVSF) theory, which is a predictor-corrector model-based state estimation strategy that guarantees stability and allows extraction of a higher degree of information from measured signals through secondary indicators of performance. These characteristics make SVSF exceptionally suitable for advanced control as well as vehicle tracking, prognostics, and health monitoring in automotive systems.

ALAN WASSYNG (Member, IEEE) is a Professor with the Department of Computing and Software, McMaster University, Canada. He is the former Director of the McMaster Centre for Software Certification and one of its founders. He has been involved in development and certification of safety-critical software intensive systems for the past 30 years.