Computers, Materials & Continua CMC, vol.64, no.2, pp.729-752, 2020

Performance Anomaly Detection in Web Services: An RNN-Based Approach Using Dynamic Quality of Service Features

Muhammad Hasnain[1], Seung Ryul Jeong[2, *], Muhammad Fermi Pasha[3] and Imran Ghani[4]

Abstract: Performance anomaly detection is the process of identifying occurrences that do not conform to expected behavior or correlate with other incidents or events in time series data. Anomaly detection has been applied to areas such as fraud detection, intrusion detection systems, and network systems. In this paper, we propose an anomaly detection framework that uses dynamic features of quality of service that are collected in a simulated setup. Three variants of recurrent neural networks-SimpleRNN, long short term memory, and gated recurrent unit are evaluated. The results reveal that the proposed method effectively detects anomalies in web services with high accuracy. The performance of the proposed anomaly detection framework is superior to that of existing approaches using maximum accuracy and detection rate metrics.

Keywords: Point anomaly, anomaly detection, recurrent neural networks, web services, simulated data.

1 Introduction

In the modern era of web technology, web services occasionally exhibit undesirable and unexpected behavior. This behavior is referred to as an anomaly, which is a bottleneck in the underlying web services [Ibidunmoye, Hernández-Rodriguez and Elmroth (2015)]. If unnoticed, performance anomalies can last from seconds to days and can arise due to a multitude of factors. For example, increasing workload, bugs, and hardware failure are known causes of performance anomalies [Wang, Wei, Zhang et al. (2014)]. Furthermore, low bandwidth and low scalability of a web server can also result in performance anomalies [Naseer and Saleem (2018); Wang, Du, Lin et al.

[1] School of Information Technology, Monash University, Subang Jaya, 47500, Malaysia.

[2] Graduate School of Business IT, Kookmin University, Seoul, Korea.

[3] School of Information Technology, Monash University, Subang Jaya, 47500, Malaysia.

[4] Department of Mathematical and Computer Sciences, Indiana University of Pennsylvania, Indiana, USA.

[*] Corresponding Author: Seung Ryul Jeong. Email: srjeong@kookmin.ac.kr.

Received: 02 March 2020; Accepted: 09 April 2020.

(2019)]. In this study, we address the challenge of detecting performance anomalies in web services.

Performance anomalies are a major problem that threatens the quality of web services [Zhang, Meng, Chen et al. (2016); Hababeh, Thabain and Alouneh (2019)]. Performance degradation can result in higher monetary costs, and service providers thus require an automated means of detecting performance problems in their web services. For large-scale and integrated web services, user interaction with web servers changes frequently and thereby affects resource requirements and workload patterns. For instance, web services are prone to various performance problems that involve the central processing unit (CPU) and memory resources [Kardani-Moghaddam, Buyya and Ramamohanarao (2019)].

Performance anomaly detection at the service level is addressed from a gray box or white box perspective, in which some knowledge regarding the source code, flow of users' transactions, and distributed components is known a priori. This approach is not feasible, however, because the source code of web services is not provided without the consent of the service providers. In addition, integrated services belong to various services providers [Ibidunmoye, Rezaie and Elmroth (2017)], and web services providers do not necessarily ship source code along with other software items. This limitation has led to new research within the academic and software development community, and researchers are now more interested in developing innovative performance anomaly detection approaches that are versatile and scalable.

Bigonha et al. [Bigonha, Ferreira and Souza (2019)] proposed using software object-oriented (OO) metrics to assess software quality. Software faults and bad smells affect the quality of software. Software code is necessary for evaluating OO metrics for smell detection and fault prediction. The proposed approach using a threshold of OO metrics is effective for general systems in which code is available, and metrics' thresholds are known in the literature. This approach is more relevant to code anomalies and cannot be applied to software systems or web services that do not provide code access. To overcome the limitations of existing approaches, other software metrics, such as performance metrics, can be used to perform anomaly detection in web services.

For example, quality of service (QoS) assessment is an effective tool for investigating performance anomalies. QoS is the representation of quality instead of the functional features of web services. As a conceptual term, QoS includes several concrete attributes. These are categorized into two broad classes: static QoS attributes and dynamic QoS attributes. Syu et al. [Syu, Kuo and Fanjiang (2017)] discussed static QoS values, which are provided by service providers. However, static QoS values are unreliable and often ineffective because they are infrequently updated. In contrast, the values of dynamic QoS metrics, such as response time and throughput, change significantly over time due to the distributed, autonomic, and heterogeneous features of web services.

Web services are owned and provided by different services providers (SPs), and run on various platforms. In addition, the dynamics of the environment, including unexpected connection latency and internet congestion, may lead to fluctuations in the response time and throughput values. Therefore, it is unwise to rely only on static values of quality attributes to perform web services operations, such as selection,

recommendation, and ranking. Instead, dynamic values acquired using appropriate means can provide accurate and reliable QoS values. The majority of existing studies propose using various time-series methods to address prediction problems. Time series forecasting is a well-developed and widely examined research field, and its forecasting methods can be used effectively. Several forecasting methods have been developed and applied to resolve real-world issues, and identifying which forecasting methods are more suitable for the dynamic QoS prediction of web services is an important task.

Owing to differences in QoS values between users' obtained QoS values and the values stated in the service level agreement (SLA), researchers have proposed a technique to optimize the SP side by considering user satisfaction. Several researchers have stated that the declaration of users' workload is an essential precondition for SLA negotiation [Ranaldo and Zimeo (2016)]. Therefore, underestimation of the users' workload can influence the QoS attributes because the actual workload may surpass the estimated load.

Because this study does not concern short-duration anomaly information, the anomaly detection method considers a week's performance of web services. Short-duration anomaly detection is suitable for measuring traffic anomalies in a network [Lakhina, Crovella and Diot (2004)]. Based on the predicted performance and actual metric values, we propose determining performance anomalies using total (actual+predicted) QoS metric values, which can effectively explain and map total QoS values to the occurrence of near-future performance anomalies in web services. This can result in quality improvement of web services because service managers or providers can avoid future anomalies.

In our proposed anomaly detection framework, two sets of anomaly detections processes are combined in the post-processing phase to estimate a long pattern of point anomaly detection. The first set of anomalies from actual data may consist of many weeks' data, while the second set of anomalies is obtained from the training and testing of the actual data. The resulting set of anomalies can be used to determine web service outages, which may be seriously taken to fix the issue in web services.

The contributions of our proposed work are as follows:

1. We propose a point anomaly detection framework that uses dynamic features of web services. This is a hybrid framework that combines performance prediction and anomaly detection from QoS features of web services.

2. We evaluate the proposed point anomaly detection framework and perform simulated data collection of dynamic features of web services.

3. We compare the performance of the proposed approach with state-of-the-art anomaly detection approaches.

The remainder of this paper is structured as follows.

In Section 2, we present background on anomaly detection approaches, while in Section 3, we describe the proposed anomaly detection approach. Next, in Section 4, we provide the results of a comparison between the proposed method and existing anomaly detection approaches. The conclusion and ideas for future work are presented in Section 5.

2 Background

The anomaly detection problem is the problem of detecting patterns that do not conform to the expected behavior in a dataset of web services. In addition, Sauvanaud et al. consider erroneous behavior of web services and SLA to be performance anomalies [Sauvanaud, Kaâniche, Kanoun et al. (2018)]. These anomalies arise from dynamic workloads and the configuration of web services. Anomaly descriptions are built upon a relationship between the behavior of web services and the performance fitness level. Various types of anomalies exist and have been discussed in the literature. For example, a point anomaly represents a point at which a web service deviates from the range of expected values (e.g., memory usage deviation from the mean value and latency spikes [Tsuda, Samejima, Akiyoshi et al. (2014); Li, Yuan, Shen et al. (2019)]. The second type of anomaly is a collective anomaly, which is a homogenous deviation of a group of data points from normal regions of the remaining data. The third type of anomaly is a contextual anomaly, which arises from specific conditions such as workload (e.g., low, moderate, and high levels). The fourth type of anomaly is known as a pattern anomaly, which arises from high dimensional data that appear in any dimension [Kim and Cho (2018)].

Web services anomaly detection is generally aimed at discovering errors by analyzing performance-related data. The goal is to verify whether a web service behaves according to expectations [Cotroneo, Natella and Rosiello (2017)]. Most existing anomaly detection approaches are focused on intrusion or misuse detection. However, our research contributes to detecting performance anomalies and their impact on the performance of web services.

In a similar work, Wang et al. [Wang, Wei, Zhang et al. (2014)] proposed a payload anomaly detection approach for web applications. The authors aimed to improve the reliability of web applications by detecting behavior that did not conform to the normal behavior of web applications. Anomalies occur when resource utilization exceeds a specified amount, or when customers rush to perform online purchase transactions on special events, including promotions and holidays such as Christmas. Workload patterns, which are access patterns and request volume, can be used to detect contextual anomalies.

Similar to our proposed work, Jin et al. [Jin, Cui, Li et al. (2018)] employed the multiple-classifier payload-based anomaly detector (McPAD) model to extract features from network traffic of web applications. Based on the extracted features, anomalies are detected using a feature clustering algorithm. The proposed McPAD model has limitations in anomaly detection based on prior anomaly probabilities because certain words (*SELECT*, *XSS*, and *UNION*) used for web attacks are given less priority than other words. Jin et al. [Jin, Cui, Li et al. (2018)] extended this model and built a prior knowledge concerning the collection of common words that are widely used during anomalies.

Rodriguez et al. [Rodriguez, Kotagiri and Buyya (2018)] discussed performance anomalies in the context of the increasing size of scientific applications and their performance. Both resource contention and failures have been considered leading causes of delay in the workflow runtime. To address this delay, the researchers proposed a hierarchical temporal memory (HTM) metric-based infrastructure model for the early detection of anomalies. Based on timestamp t, task start time e, and measure of resource consumption c, the proposed HTM model provides an anomaly output score. The

proposed HTM model has been evaluated in biological applications, such as BLAST and 1000 Genome. The results demonstrated that the performance of the HTM model is superior to that of other approaches in anomaly detection. The application of the HTM model can also be applied to web service information. Therefore, the current study is aimed at the detection of anomalies while web services receive dynamically changing workloads from users.

In a recently published work, Jayathilaka et al. [Jayathilaka, Krintz and Wolski (2019)] detected performance anomalies by observing the workload data in cloud services. Sudden changes in users' requests and slow database queries were identified as the leading causes of performance anomalies. In addition, Ghaith et al. [Ghaith, Wang, Perry et al. (2016)] proposed a performance testing method for detecting regression anomalies caused by software updates. Existing methods perform various performance regression testing in the context of applied workloads. This can become a lengthy process that requires extra effort for performance testing. The authors' proposed approach has been evaluated on two systems to identify performance regression anomalies. Real anomalies are isolated from other anomalies that emerge from workload changes.

Kim et al. [Kim and Cho (2018)] proposed the C-LSTM model to detect anomalies in traffic data of online web systems. C-LSTM combines a convolutional neural network (CNN), deep neural network (DNN), and LSTM. The proposed method extracted spatial and temporal information from complex raw data. The CNN layer was used to reduce the frequency variation in the spatial data. The main limitation of this approach is that many point anomalies, contextual anomalies, and collective anomalies were not detected. The second problem that remains unaddressed by this approach is the large delay in detecting anomalies in real data. In a more recent work, Li et al. [Li and Niggeman (2020)] proposed a geometric method for anomaly detection in complex industrial automation systems. The proposed method is effective for one-class classification by developing a boundary, such as non-convex hulls. Convex-hull-based methods provide an intuitive solution to the problem of one-class classification for convex data; however, they fail to classify non-convex data collected from cyber-physical production systems (CPPSs). Data points outside the boundary are considered anomalous data points. This method is effective because no prior knowledge is necessary; boundaries are compact and efficient representations, and generalization can be adjusted. However, a decision boundary drawn from the exact border points by the normal behavior of data may create the overfitting problem of the proposed method, which should be addressed in future work.

A point anomaly is one of the parameters of the univariate category of anomalies that occur in several types of data. Other parameters include contextual and collective anomalies. In contrast to the latter two anomalies, a point anomaly is easier to detect because it corresponds to an excessive value from individual samples [Pilastre, Boussouf, D'Escrivan et al. (2020)]. We handle the simulated data of web service performance metrics in our proposed framework to detect anomalous points, and consider a point anomaly a more appropriate parameter. Prior to our proposed work, Canizo et al. [Canizo, Triguero, Conde et al. (2019)] investigated anomaly detection in time series data of an industrial case study. The multi-head CNN method proposed in their study detected three types of anomalies. A point anomaly is one type that considers peak points in the

sensor data. In their study, Canizo et al. referred to peak points that descend from normal points as anomalous points. The main difference between our proposed method and the multi-head CNN method is that our approach considers all points beyond the normal points (i.e., either higher or lower than the normal points). In contrast, the study by Canizo et al. was limited in that it only considered points descending beyond the normal points to be anomalous, while considering ascending points to be non-anomalous points. Jia et al. [Jia, Chen, Gao et al. (2019)] also argued that points or instances near the center point have a small anomalous effect, while those far from the center point have a large impact. This implies that a point that is far away from the center is an anomalous point.

Wang et al. [Wang, Jing, Qi et al. (2019)] proposed the adaptive label screening and relearning (ALSR) approach, which is aimed at detecting differences between anomalous points. Because earlier approaches focus more on individual points rather than events, the labels of continuous anomalies and their intervals are not appropriately handled, and the predicted performance is not precise in real-world situations. To overcome this problem, the ALSR approach uses continuous label screening and intervals of anomalies for finer granularity. The ALSR approach is effective with mainstream performance indicators. However, the performance of this approach is affected by the diversity of performance indicators.

Ding et al. [Ding, Ma, Gao et al. (2019)] investigated anomaly detection in system design, as high-quality anomaly detection can ensure the high performance of various applications, such as disaster prevention, system monitoring, and intrusion detection. To overcome the above-stated anomaly detection issue, the researchers proposed an anomaly detection approach of time series information by combining LSTM and the Gaussian matrix model (GMM). The former model best evaluates real-time anomalies in time series information, while the latter model detects possible point anomalies. The method proposed by Ding et al. performs better than state-of-the-art methods in univariate time series with the best convergence; however, it displays performance degradation when applied to multivariate (high-dimensional) time series data for anomaly detection.

In summary, existing research on anomaly detection identifies point anomalies, contextual anomalies, collection anomalies, and pattern anomalies in different types of software systems. For each type of anomaly detection, several methods have been proposed. The proposed anomaly detection methods focus on the univariate and multivariate features of datasets. However, research is necessary for handling anomaly detection in multivariate datasets because complexity increases with a high-dimensional dataset. None of the aforementioned studies considers performance anomaly detection in a dynamic QoS dataset collected in a simulated environment. However, the existing approaches inspire us to locate the performance points that surpass the normal points.

3 Proposed anomaly detection approach

In this section, we describe the proposed anomaly detection framework, which is illustrated in Fig. 1.

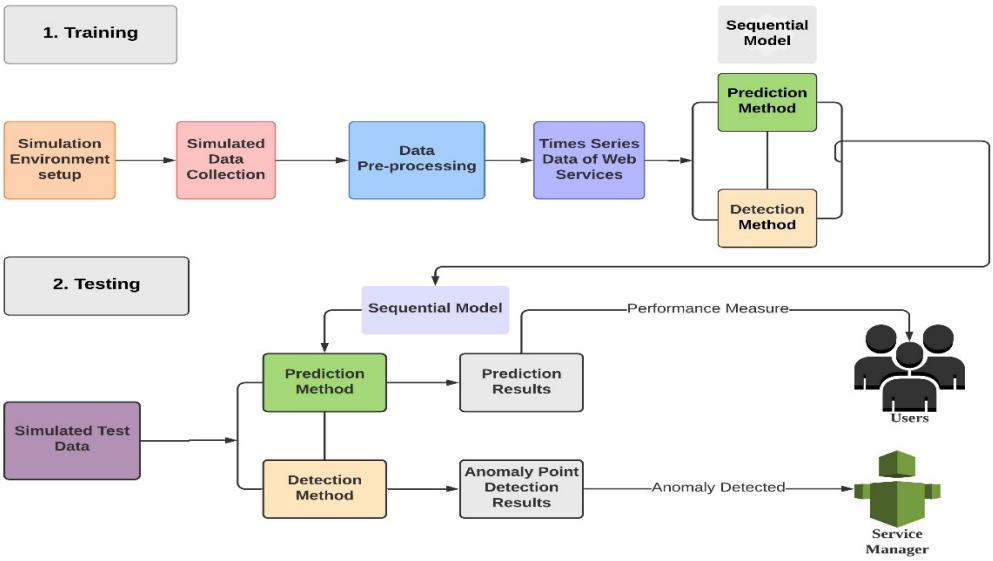

Figure 1: Proposed anomaly detection framework

To predict performance anomalies in web services, variants of the recurrent neural network (RNN) model, such as GRU, LSTM, and SimpleRNN, have been proposed. Before this, the aforementioned sequential models including GRU, LSTM, and SimpleRNN were successfully applied to estimate the performance of web services from time series data [Hasnain, Fermi, Lim et al. (2019)]. The proposed GRU model is the most effective of the approaches when there are large fluctuations in the performance of web services. The problem of performance prediction and anomaly detection has long been unresolved; therefore, we implemented deep learning methods to detect point anomalies in web services. The proposed anomaly detection framework is based on two phases and several sub-phases, as illustrated in Fig. 1.

We have proposed to use training and testing phases, as shown in Fig. 1. The training phase of the proposed anomaly detection framework involves further steps, including simulated dataset collection, data pre-processing, and time-series data. Once these steps are completed, we train the sequential model (GRU, LSTM, and SimpleRNN) by using the time series data of web services. Upon the completion of the training of sequential models, we evaluate the sequential model at the testing phase of the proposed framework. An appropriate portion of the same simulated dataset is used to test the sequential models. Performance prediction results, as well as anomaly detection results, are reported in this phase of the proposed framework. Both web services users and web service managers get alarmed when performance anomalies are detected.

3.1 Overview of anomaly detection approach

In this subsection, we present an overview of the anomaly detection framework. Anomaly detection approaches require benchmark datasets to evaluate and compare their performance with other detection methods. We used a benchmark QoS dataset with a large number of performance instances of web services [Zheng, Zhang and Lyu (2010,

2012)]. The dataset contained outdated information regarding throughput and response time quality metrics. Moreover, aforementioned dataset does not contain the workload information regarding web services users. In this paper, we aim to bridge this gap by describing time series data collection in a simulated environment and analyze the data to identify performance problems in web services.

Simulation dataset collection is the first step in the training phase of the proposed framework. The procedure for dataset collection is as follows. First, we created a simulation environment to collect quality features of web services. The purpose was to collect the time series data of web services for different workloads. The WSDream dataset has been widely used in published works [Ma, Wang, Hung et al. (2015); Raj, Mahajan, Singh et al. (2019)]. Because this dataset consists of historical information about web service quality attributes, we were unable to use the collected information because it was outdated and less feasible for the proposed framework. Instead, we used concurrent workloads from 100 users and 200 users during the simulation with the help of Apache Jmeter 5.1.1. Our previous work outlines the proposed strategy of workloads (concurrent users) and other requirements [Hasnain, Pasha and Ghani (2020)]. Once the Apache Jmeter node was established on the computer system, we collected web service information. From the simulation environment, we collected time series data for 40 days and stored the data in a CSV file for further data preprocessing. We then fed the preprocessed data into our RNN methods for training. It is worth noting that no anomalous signal filtering process was used during the training phase in this method. We simultaneously trained the prediction and detection modules of the selected RNN methods. The proposed framework detects anomalous signals (performance) in advance, which allows the web service manager to examine the abnormal data points and modify the actions of web services developers and testers to correct them.

3.2 Prediction

The proposed prediction method module is intended to forecast the performance of web services as well as abnormal signals of web services. The prediction of abnormal and normal sequences of information has not been performed in previous studies. Therefore, our proposed anomaly detection framework is the first to both evaluate the performance of web service information from time series simulated data and detect unusual behavior of web services. This enables web service managers to obtain forecasted information on performance and abnormalities in web services. Because web service managers receive information in advance about anomalous behavior of the web service, they can correct this behavior to avoid performance degradation.

Suppose that we have sequential simulated data with an input $d_{0:t}$ [a_0, a_1…at] in web services such as throughput, where *a* represents univariate data and *t* represents the time step. We perform data transmission proposed in a paper by Wang et al. [Wang, Li, Fu et al. (2019)], in which the authors predicted wind power to address fluctuations in its features.

3.2.1 Gated recurrent unit

The GRU with the forward pass is expressed as follows:

$$z_t = \sigma\left(W_{xz}x_t + U_{hz}h_{t-1} + b_z\right) \quad (1)$$

$$r_t = \sigma\,(W_{xr}x_t + U_{hr}h_{t-1} + b_r) \quad (2)$$

$$h'_t = \tanh\,(W_{xc}x_t + U_{hc}(r_t \odot h_{t-1}) + b_c) \quad (3)$$

$$h_t = (z_t)\odot(h_{t-1} + (1 - z_t)\odot h'_t) \quad (4)$$

$$\tanh(t) = (e^t - e^{-t})/(e^t + e^{-t}) \quad (5)$$

$$\sigma(t) = 1/(1 + e^{-t}), \quad (6)$$

where z_t represents the update gate that aims to ensure that a part of current hidden state h_t updates, and h_{t-1} denotes the last time output. In addition, σ and tanh are two different activation functions that are used to represent the gated mechanism and normalize the input information, respectively, and ⊙ represents the dot products. W_{xz}, W_{xc} , and W_{xr} are weight vectors pertaining to the current input.

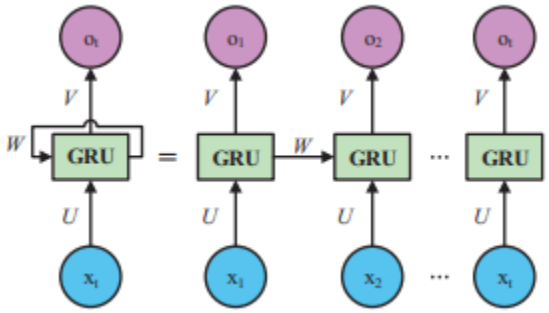

Figure 2: Overview of gated recurrent unit (GRU) architecture

For the representation of the GRU model as an RNN model, a neuron is designated as a GRU. The new architecture of the RNN-based GRU model is presented in Fig. 2. Simple RNN parameters are used to train the method, and can be calculated by following the backpropagation network as follows:

$$o_t = Vf(U_{Xt} + Wf\big(U_{Xt-1} + Wf(U_{Xt-2} + Wf(U_{Xt-3} + \cdots))\big)\big) \quad (7)$$

Here, o_t denotes the output of one neural cell. U, V, and W denote the weight matrices of x, h, and the output layer, respectively, and $f(\cdot)$ is the representation of the activation function.

A standard GRU model is capable of handling the time series (sequential) data efficiently and eases the vanishing problem of RNNs. Also, the gating structure of the GRU model can lead to the omission of important contents in the time series data. As shown in Fig. 2, there is the input layer, which is composed of many neurons. Above the input layer, there is a middle layer, which is also known as hidden layer. Each neuron in the output layer corresponds to the output space, as shown in Fig. 2. The middle layer, which is the one where the primary function of the GRU model takes place. So any change in the cell status depends on the working of reset and update gates.

3.2.2 Long short term memory

LSTM is another derivation of the RNN that is widely used to address sequential prediction problems. Like the GRU model, LSTM also aims to overcome the gradient vanishing problem that is observed in RNNs [Kim and Chung (2019)]. Due to this

problem, the gradient is further reduced when it returns to the early layers. As a result, learning in the earlier layers is inadequate, and the performance of the neural network decreases [Liu, Li, Chen et al. (2019)]. To overcome this limitation, the LSTM model is designed alongside three gates, including the input gate, output gate, and forget gate. These gates aid in remembering the results of the input sequences that are computed much earlier. The importance of LSTM in prediction is due to its cell state, which updates after every time step [Thara, PremaSudha and Xiong (2019)]. The cell state at moment *t* is recorded as c_t, which can be regarded as the memory unit of LSTM. Reading as well as memory modification can be achieved by controlling the input, output, and forget gates. These gates are generally described with tanh or sigmoid functions. The structure of the LSTM network is illustrated in Fig. 3.

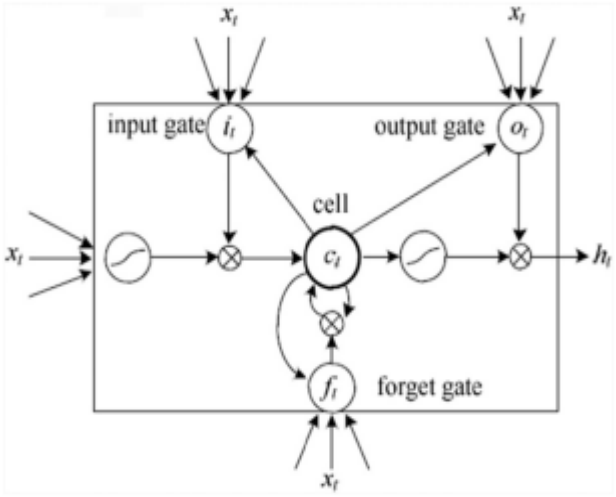

Figure 3: Overview of long short term memory (LSTM) unit

Fig. 3 provides an illustration of the structure and workflow of the LSTM unit. At each time, an LSTM unit obtains an input from the current state known as x_t and hidden state known as h_{t-1} of the LSTM using three gates. In addition, each door obtains the internal information input that is the state of memory unit c_{t-1}. Each gate operates on the data obtained from various sources, and the logic function defines whether the gate is active after it receives the inputs. A nonlinear function transforms the input obtained at the input gate. It also superimposes the memory cell's state that is processed by the forget gate to construct a new memory cell state, c_t. Finally, memory cell state c_t leads the LSTM unit to output h_t from operation of the nonlinear function as well as dynamic control by the output gate. Calculation of the above is as follows:

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \tag{8}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{9}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{10}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{11}$$

$$h_t = o_t \tanh(c_t), \tag{12}$$

where W_{xf}, W_{xi}, W_{xc}, and W_{xo} represent the weight matrices that connect the input data signal x_t; W_{ci}, W_{cf}, and W_{co} represent the diagonal matrices that combine the output vectors c_t with the gate functions of the neuron activation functions; W_{hc}, W_{ih}, W_{hf}, and W_{ho} represent the weight matrices that combine the hidden layer output signal h_t; σ represents the activation function that is usually tanh or a sigmoid function; and b_c, b_f, b_i, and b_o represent the offset vectors.

3.2.3 SimpleRNN

For the performance prediction of web services, SimpleRNN has the same structure as GRU and LSTM for implementation in the Keras framework. To execute SimpleRNN, GRU, or LSTM, layers are replaced with SimpleRNN layers (Zhang et al. [Zhang, Xiong, He et al. (2018)]. A simple RNN architecture is presented in Fig. 4.

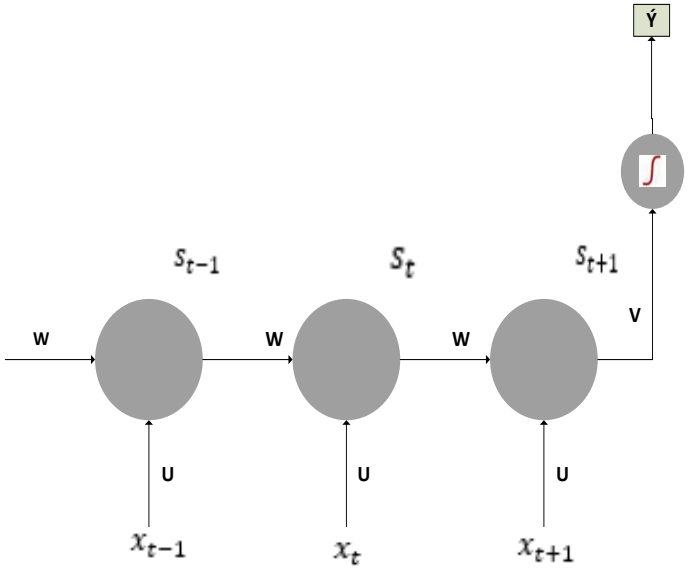

Figure 4: SimpleRNN network

Fig. 4 illustrates a simple architecture of the SimpleRNN model adopted from an existing study [Reddy and Delen (2018)]. Each node in this figure represents a layer of network units at each time step. Three meaningful connections, such as an input to a hidden layer, a hidden layer to a hidden layer, and a hidden layer to an output layer, are displayed in Fig. 4. Here, the U, W, and V matrices represent the weighted connections from an input to a hidden layer, a hidden layer to a hidden layer, and a hidden layer to an output layer, respectively. A scalar Y value is produced by passing the final weight matrix through the sigmoid function that is classified as a binary variable, and it is referred to as Ý. For comparison of Y-Predicted and the Y-actual, a loss function is used. It is worth noting that the same weights are used at each time step.

In addition to SimpleRNN, two popular enhancements to SimpleRNN have been developed. GRU and LSTM are extensions of SimpleRNN that can store prior

information of extracted features in memory. In this study, we use GRU and LSTM along with SimpleRNN to compare their performance on a simulated dataset of web services.

3.3 Detection method

The GRU method was initially proposed by Cho et al. [Cho, Merriënboer and Gulcehre (2014)], and has a more straightforward architecture than the LSTM method. However, SimpleRNN and LSTM are still effective in performance prediction and anomaly detection for fluctuating behavior of web services [Hasnain, Fermi, Lim et al. (2019)]. The GRU method is used to avoid the limitations of the SimpleRNN and LSTM methods. A GRU method with its update and reset gate vectors is capable of determining what information that is part of the input should be memorized, and what information should be forgotten [Kong, Tang, Deng et al. (2020)]. In addition, a GRU method can keep performance stable due to fewer parameters that can suppress overfitting. A GRU method contains an information modulation process, which is similar to LSTM; however, a GRU method does not contain a separate memory cell. The detection method module, as illustrated in Fig. 1, is intended to receive input signals (throughput value) and determine whether any of the signals are anomalous.

The presence of anomalies leads to inconsistency in signal reconstruction because RNN methods exploit their reconstruction Therefore, an auto-encoder method is trained on the data, and a threshold value above the reconstruction is known as an anomaly [Wielgosz, Mertik, Skoczeń et al. (2018); Habler and Shabtai (2018)]. Previous papers have dealt with different types of signals, and the approach based on the SimpleRNN, LSTM, and GRU methods can be effectively used in the domain of web services anomaly detection.

3.4 Anomaly threshold frequency

An automated threshold frequency was determined by considering the simulated data of the quality metrics of web services, as presented in Tab. 1.

Table 1: Threshold statistics for two different workloads

Statistics			
		100 Users	200 Users
N	Valid	40	40
	Missing	0	0
Mean		**11.955**	**19.015**
Median		12.150	18.350
Std. Deviation		2.1957	4.0623
Skewness		-0.463	-0.056
Std. Error of Skewness		0.374	0.374
Kurtosis		0.799	-1.225
Std. Error of Kurtosis		0.733	0.733
Minimum		5.9	11.6
Maximum		15.7	25.3

Tab. 1 provides an illustration of the statistics extracted on the collected simulated dataset over 40 days. We verified all errors in the collected data and performed preprocessing to clean the data. The statistics in Tab. 1 indicate that no values were missing, and the mean and standard deviation (SD) values for two workloads were calculated.

Prior to the execution of the RNN anomaly detection methods, we determined the threshold values for the simulated workloads. Therefore, a threshold value is arbitrary because it is difficult to discern, agreeing for the detection of all anomalies, and would be adequate relying on the results achieved from the simulated data. A possible combination of threshold frequency might be the mean and SD, which can be used to filter out the anomalous points from time series information of web services. This representation allows us to compute the mean and SD values of any type of data equivalent with a number of week's information. However, the mean value and SD value can be updated to increase the size of the data for many weeks and keep the proposed method up to date while changes occur in time series information [Guigou, Collet and Parrend (2019)].

This method is robust and lightweight because the mean and SD values of time series information over a long period of time can be easily maintained. Therefore, this method can handle changes in time series information in web services. First, a threshold was applied to simulated data to mark the information contained in the training dataset [Tian, Azarian and Pecht (2014)]. This was because the training data contained both anomalous and normal values of web services. For instance, we used 11.955 and 19.015 as thresholds for a 100-user workload and 200-user workload, respectively. We proposed adding ±1.00 to the threshold value to represent the normal behavior of web services. Therefore, any predicted value that exceeded 11.955±1.00 was represented as an abnormal value and was also considered a performance anomaly in the case of the 100-user workload. Similarly, a value that was above and below 19.015±1.00 was considered an anomaly in the case of the 200-user workload. Traditional methods use mean and SD with the mean plus two SDs [Zuo, Wang and Chen (2015)]. We reduced the mean positive and negative values to 1 because web service users are not satisfied with web services with high performance fluctuations.

4 Experiments

4.1 Dataset

To test our proposed performance prediction and anomaly detection, we used the simulated dataset of web services with throughput as a quality attribute. The dataset consisted of both abnormal and normal behavior of web services in terms of the quality attributes. The value of the quality attribute was obtained from the installed software, which enabled the simulation environment in the lab. Each point in the dataset was obtained using a uniform environment for 40 consecutive days. Web service (WS1) data collected from the simulation environment with two different workloads is plotted in Fig. 5.

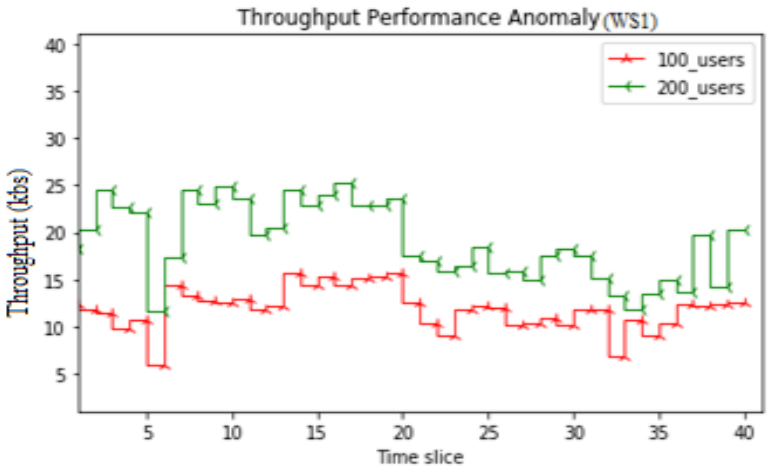

Figure 5: Example of simulation information in which two workloads are represented by different colors in terms of the throughput of web service 1 (WS1)

Fig. 5 provides a representation of the actual data obtained from simulation using two workloads of 100 users and 200 users, respectively. To visualize normal states in the actual simulated dataset, we changed the number of users. When the number of users was changed from 100 to 200, the average throughput value decreased, and consequently, web service performance degraded. Thus, for the 200-user workload, the throughput of the web service demonstrated more fluctuations than for the 100-user workload.

4.2 Evaluation metric

To evaluate the performance of our proposed GRU method and baseline (SimpleRNN and LSTM) methods, the mean absolute error (MAE) was employed as the criteria. Within the MAE (%), a smaller MAE represents superior performance of the method. The MAE equation is expressed as follows:

$$(\text{MAE}) = \frac{1}{n}\sum_{i=1}^{n} \text{abs}\left(y_i - \lambda(x_i)\right), \tag{13}$$

where y_i and n represent the predicted value and a total number of predicted values at any given time, respectively. In conjunction with MAE, the mean absolute percent error (MAPE) metric can be used to demonstrate the proportional relationship between the predicted value and actual value. In addition to these metrics, the mean square error (MSE) and root mean square error (RMSE) metrics are sensitive to outliers that can limit their efficacy [Jackson, Roberts, Nelsen et al. (2019)]. Thus, we selected the simplest metric, MAE that is widely used to measure absolute error. In addition, the MAE metric avoids the problem of errors canceling each other out, and can accurately reflect actual error prediction.

4.3 Results and analysis

In this subsection, we present the performance prediction and anomaly detection results and their analysis.

4.3.1 Performance prediction results

Here, we are interested in predicting the performance of a web service before the detection of anomalies. GRU, SimpleRNN, and LSTM were compiled, and the comparison results are provided in Figs. 6 and 8 for two different workloads.

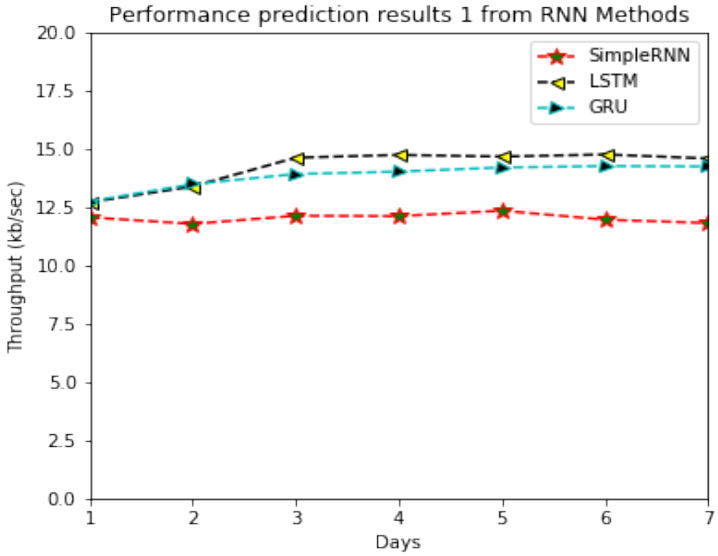

Figure 6: Performance prediction results from 100-user workload data

Fig. 6 illustrates the results from 100-user workload simulated data that were calculated to predict the performance of a web service for the next seven timestamps (days).

Table 2: Parameters and epochs

Prediction Method	Number of total params	Number of trainable params	Number of epochs	Neuron at dense layer
SimpleRNN	30,401	30,401	800	100
LSTM	121,301	121,301	800	100
GRU	91,001	91,001	800	100

Each performance prediction method was compiled on 800 epochs with 100 neurons in the dense layers. In our experiments on simulated data of 100-user and 200-user workloads, the total trainable parameters, biases, and weights combined were 30,401, 121,301, and 91,001 for the SimpleRNN, LSTM, and GRU methods, respectively. The total number of parameters was equal to the sum of the trainable parameters for each method. Our selected prediction methods (SimpleRNN, LSTM, and GRU) displayed better prediction accuracy at 800 epochs, as illustrated in Tab. 4.

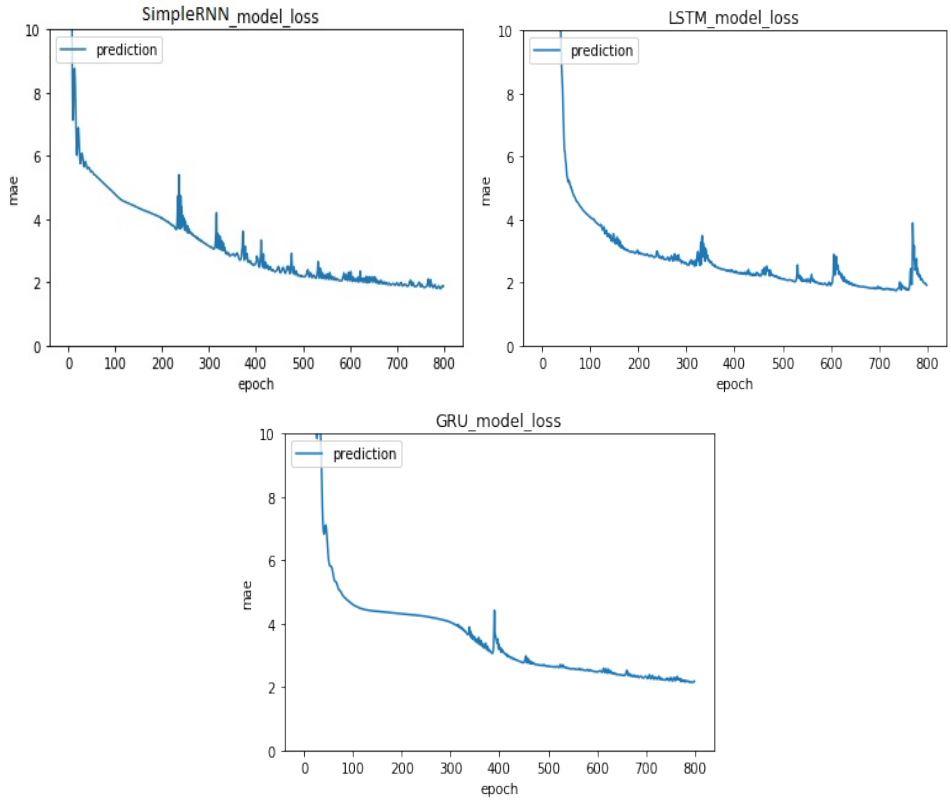

Figure 7: SimpleRNN, LSTM, and GRU methods error loss on prediction from simulation data of 100-users. Line in plot (a) represents the SimpleRNN method loss during the performance prediction. Line in plot (b) represents the LSTM method loss during the performance prediction, while line in plot (c) represents the GRU method loss of 6.0×10^2

The number of epochs influences the performance of a prediction method. Each method has a fixed architecture, and the loss function varies with the prediction process. As illustrated in Fig. 7(a), prediction results are apparent with errors although the loss decreases without stabilizing. In comparison with the SimpleRNN and GRU prediction methods, the LSTM method is less stable as it reaches the end of the epochs. Therefore, both SimpleRNN and LSTM are less stable and have a larger number of prediction errors. In contrast, the GRU method, as illustrated in Fig. 7(c), is more stable with a fixed number of epochs. This indicates that the GRU model is more effective in predicting the performance of web services.

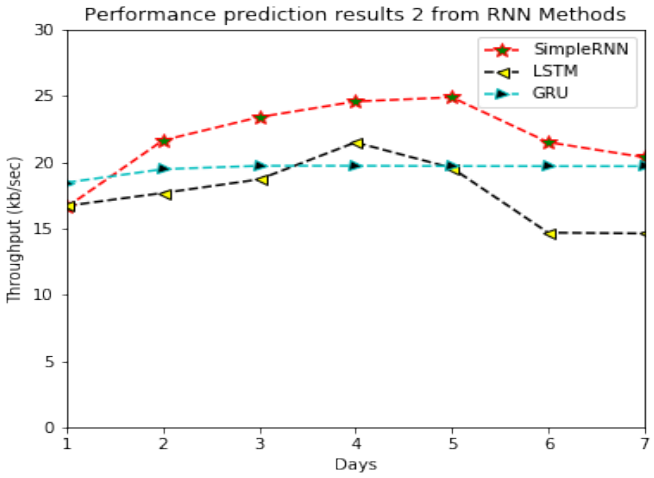

Figure 8: Performance prediction results from 200-user workload data

Fig. 8 presents the prediction results of 200-user workload data using RNN methods. In comparison to the SimpleRNN and LSTM methods, GRU exhibits performance prediction that is similar to neighboring results for the next seven timestamps (days).

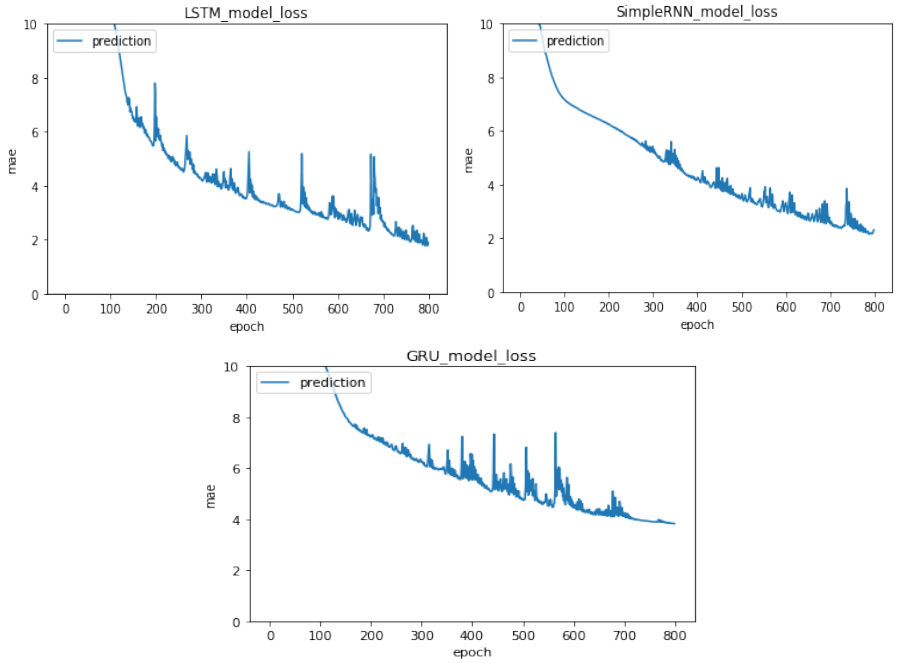

Figure 9: SimpleRNN, LSTM, and GRU methods error loss on prediction from simulation data of 200-users. Line in plot (a) represents the SimpleRNN method loss during the performance prediction. Line in plot (b) represents the LSTM method loss during the performance prediction, while line in plot (c) represents the GRU method loss of 6.0×10^{2}

Fig. 9 demonstrates that the error loss of the three performance prediction models decreases with an increase in the number of epochs from 200-user simulated data. However, the GRU method becomes more stable than the SimpleRNN and LSTM methods, and therefore, performance prediction by GRU is more effective than by the latter two methods.

4.3.2 Anomaly detection results

To determine whether an anomaly has occurred as a detection, a threshold is used in our proposed detection method. Anomalous points, along with their frequency for each detection method, are provided in Tab. 3. By using these anomalous points, we calculated the overall percent of anomalies. We used our proposed threshold values for each detection method to determine the total anomalous points for the data of two different workloads.

We observed a total of 47 points. Of these, 40 points represent the actual data, and the remaining seven points are the predicted points. The upper bound and lower bound define the space in which normal performance information lies. Our proposed anomaly detection framework sets the values of the upper bounds and lower bounds in Section 3.4 for two different workloads. An anomalous threshold value (11.955±1.00) is used to observe the upper bound (12.955) and lower bound (10.955) for the 100-user workload. Similarly, anomaly threshold frequency (19.015±1.00) statistics were further elaborated by specifying the upper bound (20.015) and lower bound (18.015) values for the 200-user workload. Points emerging far from the upper bound and lower bound are considered abnormal or anomalous points. The false alarm rate can be determined by calculating data points beyond the upper bound and lower bound to avoid the risk of performance degradation. Although mitigating the effects of the detected point anomalies is outside of the scope of this study, it is advised to keep web services in the region between the two bounds.

Based on the actual and predicted performance points, we employed anomaly detection to calculate the anomalous points for both actual and predicted points. Based on the proposed criteria of anomalous point detection, we used the statistics provided in Tab. 1. The proposed measures are helpful for determining anomalous points, which are beyond 11.955±1.00 and 19.015±1.00 for two different workloads. Using these statistics, we determined point anomalies from two different workloads, as presented in Tab. 3.

Table 3: Point anomalies detected by RNN methods

Detection method	Workload	Actual points	Predicted points	Anomalous points in the actual data	Anomalous points in predicted data	Total anomalous points	Percent of anomalies (%)
Simple RNN	100-user	40	07	20	01	21	44.68
	200-user	**40**	**07**	**35**	**07**	**42**	**89.36**
LSTM	100-user	40	07	20	06	26	55.32
	200-user	40	07	35	05	40	85.11
GRU	100-user	40	07	20	06	26	55.32
	200-user	40	07	35	00	35	74.47

Tab. 3 presents detailed information about point anomalies and their detection in the actual data and predicted data by the proposed anomaly detection framework. We collected simulated data of 40 points (timestamps), and out of these points, we detected 20 and 35 anomalous points in the 100-user and 200-user workloads, respectively. A 100-user workload is more favorable for a web service manager, as it has a smaller number of points beyond the threshold performance values. Upon increasing the workload, the actual data collected from the simulation demonstrated a large number of points beyond the upper and lower bounds of the threshold, as indicated in Tab. 3.

We observed variation in the predicted anomalous points for three prediction methods. All methods were implemented in the Colab environment with Keras layers. The GRU method and other two methods were composed of one method layer and one dense layer. The number of epochs was set to 800 because the GRU method displayed better performance for this number of epochs. To compare the efficiency of the different prediction methods, we performed anomaly detection. LSTM and GRU were more effective than the SimpleRNN method in the detection of point anomalies for the 100-user workload. However, SimpleRNN was more effective than GRU and LSTM in the detection of point anomalies for the 200-user workload. This indicates that SimpleRNN is effective in detecting abnormal points in a simulated dataset.

4.3.3 Comparison of performance of RNN methods

To verify the effectiveness of the proposed RNN methods, we conducted experiments on time series simulated data from two different workloads. As stated earlier in Section 4.3, for all simulations (including data collection, prediction, and detection), we used the same settings for software and hardware. However, for the performance prediction of web services, we adjusted the parameters of the RNN methods, including the SimpleRNN, LSTM, and GRU methods. We used the MAE metric to evaluate the prediction performance of web services. The comparison results are provided in Tab. 4.

Table 4: Performance accuracy comparison

Workload	**Feature**	**MAE%**		
	Throughput	Simple RNN	LSTM	GRU
100-user		0.96	0.93	1.02
200-user		1.08	0.93	1.56

Tab. 4 presents a comparison of the performance of the three RNN methods. The SimpleRNN and LSTM methods were used as baseline methods, while the GRU method was used as the main contender. The results in Tab. 4 indicate that LSTM is superior to the SimpleRNN and GRU methods, as it achieves a lower MAE% for both 100-user and 200-user workloads. The comparison results demonstrate that all three prediction methods have a high prediction accuracy in web services performance prediction and anomaly detection. A smaller value of MAE (%) for a method indicates better prediction quality.

4.3.4 Comparison between proposed point anomaly detection framework and other approaches

Existing point anomaly detection approaches have been evaluated on datasets other than web service dynamic quality features, and our proposed framework is the first to employ the dynamic features of web services. To demonstrate the effectiveness of the proposed framework, we performed evaluation using the detection rate (DR) and maximum accuracy metrics.

Table 5: Performance comparison of our proposed framework

Study	Approach	Detection rate	Maximum accuracy (%)
[Hundman, Constantinou, Laporte et al. (2018)]	LSTM	59%	--
[Poornima and Paramasivan (2020)]	Principal component analysis (PCA)	--	91.00
	Support vector machine (SVM)	--	89.00
	Decision Tree	--	82.00
[Garg, Kaur, Batra et al. (2020)]	Boruta Firefly Aided Partitioning DBSCAN (BFA- DBSCAN)	--	98.90
Our study	SimpleRNN	**89.36**	99.04
	LSTM	85.11	**99.07**
	GRU	74.47	98.98

Tab. 5 presents the performance evaluation results of the proposed framework and state-of-the-art anomaly detection approaches. The performance comparison results in Tab. 5 indicate that our proposed framework achieved a higher DR than existing approaches. Moreover, maximum accuracy values such as 99.07, 99.04, and 98.98 indicate that the proposed framework is superior in point anomaly detection in web services.

5 Conclusion and future work

In this paper, we proposed an application of RNN prediction and anomaly detection methods. Our target was the GRU method along with SimpleRNN and LSTM variants to analyze simulated data of throughput as a quality metric of web services. We used a simulated dataset of web services that contained throughput quality data from 100 and 200 users for 40 days. We preprocessed the data to prepare it for RNN variant structures. Three RNN variants were used for performance prediction and anomaly detection. In this paper, we present a performance prediction and anomaly detection framework that employs threshold frequency. This anomaly detection framework, which is inspired by the paradigm of deep learning methods, forecasts the seven-step performance of web services. Then, anomalous points are determined from the behavior of web services. We performed a theoretical performance comparison of the proposed GRU method with other RNN variants. Our proposed GRU method and other methods including LSTM and

SimpleRNN were able to effectively predict the performance of web services. Based on the simulated information and predicted performance, the proposed framework can be used to detect anomalies in web service quality metrics information.

In future work, we plan to extend our proposed anomaly detection framework from the perspective of design and application. Regarding design, we plan to use deep learning for the detection of high dimensionality in the sample data, which has been rarely undertaken in the literature. Second, we plan to extend the proposed framework in its applications for multivariate datasets with the aim of improving the quality of the framework.

Acknowledgement: The authors would like to thank School of Information Technology Malaysia, Monash University for providing research facilities.

Funding Statement: The authors receive no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Bigonha, M. A.; Ferreira, K.; Souza, P.; Sousa, B.; Januário, M. et al. (2019): The usefulness of software metric thresholds for detection of bad smells and fault prediction. *Information and Software Technology*, vol. 115, pp. 79-92.

Canizo, M.; Triguero, I.; Conde, A.; Onieva, E. (2019): Multi-head CNN-RNN for multi-time series anomaly detection: an industrial case study. *Neurocomputing*, vol. 363, pp. 246-260.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F. et al. (2014): Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724-1734.

Cotroneo, D.; Natella, R.; Rosiello, S. (2017): A fault correlation approach to detect performance anomalies in virtual network function chains. *IEEE 28th International Symposium on Software Reliability Engineering*, pp. 90-100.

Ding, N.; Ma, H.; Gao, H.; Ma, Y.; Tan, G. (2019): Real-time anomaly detection based on long short-term memory and gaussian mixture model. *Computers & Electrical Engineering*, vol. 79, pp. 1-11.

Garg, S.; Kaur, K.; Batra, S.; Kaddoum, G.; Kumar, N. et al. (2020): A multi-stage anomaly detection scheme for augmenting the security in IoT-enabled applications. *Future Generation Computer Systems*, vol. 104, pp. 105-118.

Ghaith, S.; Wang, M.; Perry, P.; Jiang, Z. M.; O'Sullivan, P. et al. (2016): Anomaly detection in performance regression testing by transaction profile estimation. *Software Testing, Verification and Reliability*, vol. 26, no. 1, pp. 4-39.

Guigou, F.; Collet, P.; Parrend, P. (2019): SCHED A: lightweight euclidean-like heuristics for anomaly detection in periodic time series. *Applied Soft Computing*, vol. 82, pp. 1-14.

Hababeh, I.; Thabain, A.; Alouneh, S. (2019): An effective multivariate control framework for monitoring cloud systems performance. *KSII Transactions on Internet & Information Systems*, vol. 13, no. 1, pp. 86-109.

Habler, E.; Shabtai, A. (2018): Using LSTM encoder-decoder algorithm for detecting anomalous ADS-B messages. *Computers & Security*, vol. 78, pp. 155-173.

Hasnain, M.; Pasha, M. F.; Ghani, I. (2020): Drupal core 8 caching mechanism for scalability improvement of web services. *Software Impacts*, vol. 3, pp. 1-4.

Hasnain, M.; Pasha, M. F.; Lim, C. H.; Ghani, I. (2019): Recurrent neural network for web services performance forecasting, ranking and regression testing. *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pp. 96-105.

Hundman, K.; Constantinou, V.; Laporte, C.; Colwell, I.; Soderstrom, T. (2018): Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 387-395.

Ibidunmoye, O.; Hernández-Rodriguez, F.; Elmroth, E. (2015): Performance anomaly detection and bottleneck identification. *ACM Computing Surveys*, vol. 48, no. 1, pp. 1-35.

Ibidunmoye, O.; Rezaie, A. R.; Elmroth, E. (2017): Adaptive anomaly detection in performance metric streams. *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 217-231.

Jackson, E. K.; Roberts, W.; Nelsen, B.; Williams, G. P.; Nelson, E. J. et al. (2019): Introductory overview: error metrics for hydrologic modelling-A review of common practices and an open source library to facilitate use and adoption. *Environmental Modelling & Software*, vol. 119, pp. 32-48.

Jayathilaka, H.; Krintz, C.; Wolski, R. M. (2018): Detecting performance anomalies in cloud platform applications. *IEEE Transactions on Cloud Computing*, pp. 1-14.

Jia, Q. X.; Chen, C. X.; Gao, X.; Li, X. P.; Yan, B. et al. (2019): Anomaly detection method using center offset measurement based on leverage principle. *Knowledge-Based Systems*, vol. 190, pp. 1-16.

Jin, X.; Cui, B.; Li, D.; Cheng, Z.; Yin, C. (2018): An improved payload-based anomaly detector for web applications. *Journal of Network and Computer Applications*, vol. 106, pp. 111-116.

Kim, T. Y.; Cho, S. B. (2018): Web traffic anomaly detection using C-LSTM neural networks. *Expert Systems with Applications*, vol. 106, pp. 66-76.

Kim, J.; Chung, K. (2019): Prediction model of user physical activity using data characteristics-based long short-term memory recurrent neural networks. *KSII Transactions on Internet & Information Systems*, vol. 13, no. 4, pp. 2060-2077.

Kardani-Moghaddam, S.; Buyya, R.; Ramamohanarao, K. (2019): Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications

in computing clouds. *Concurrency and Computation: Practice and Experience*, vol. 31, no. 20, pp. 1-17.

Kong, Z.; Tang, B.; Deng, L.; Liu, W.; Han, Y. (2020): Condition monitoring of wind turbines based on spatio-temporal fusion of SCADA data by convolutional neural networks and gated recurrent units. *Renewable Energy*, vol. 146, pp. 760-768.

Lakhina, A.; Crovella, M.; Diot, C. (2004): Characterization of network-wide anomalies in traffic flows. *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pp. 201-206.

Li, B.; Yuan, G.; Shen, L.; Zhang, R.; Yao, Y. (2019): Incorporating URL embedding into ensemble clustering to detect web anomalies. *Future Generation Computer Systems*, vol. 96, pp. 176-184.

Li, P.; Niggemann, O. (2020): Non-convex hull based anomaly detection in CPPS. *Engineering Applications of Artificial Intelligence*, vol. 87, pp. 1-11.

Liu, J.; Li, Q.; Chen, W.; Yan, Y.; Qiu, Y. et al. (2019): Remaining useful life prediction of PEMFC based on long short-term memory recurrent neural networks. *International Journal of Hydrogen Energy*, vol. 44, no. 11, pp. 5470-5480.

Ma, Y.; Wang, S.; Hung, P. C.; Hsu, C. H.; Sun, Q. et al. (2015): A highly accurate prediction algorithm for unknown web service QoS values. *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 511-523.

Naseer, S.; Saleem, Y. (2018): Enhanced network intrusion detection using deep convolutional neural networks. *KSII Transactions on Internet & Information Systems*, vol. 12, no. 10, pp. 5159-5178.

Pilastre, B.; Boussouf, L.; D'Escrivan, S.; Tourneret, J. Y. (2020): Anomaly detection in mixed telemetry data using a sparse representation and dictionary learning. *Signal Processing*, vol. 168, pp. 1-10.

Poornima, I. G. A.; Paramasivan, B. (2020): Anomaly detection in wireless sensor network using machine learning algorithm. *Computer Communications*, vol. 151, pp. 331-337.

Raj, G.; Mahajan, M.; Singh, D.; Singh, A. (2019): Imputing missing data analysis in web service quality dataset for improving QoS prediction. *Recent Trends in Programming Languages*, vol. 6, no. 2, pp. 8-22.

Ranaldo, N.; Zimeo, E. (2016): Capacity-driven utility model for service level agreement negotiation of cloud services. *Future Generation Computer Systems*, vol. 55, pp. 186-199.

Reddy, B. K.; Delen, D. (2018): Predicting hospital readmission for lupus patients: an RNN-LSTM-based deep-learning methodology. *Computers in Biology and Medicine*, vol. 101, pp. 199-209.

Rodriguez, M. A.; Kotagiri, R.; Buyya, R. (2018): Detecting performance anomalies in scientific workflows using hierarchical temporal memory. *Future Generation Computer Systems*, vol. 88, pp. 624-635.

Sauvanaud, C.; Kaâniche, M.; Kanoun, K.; Lazri, K.; Silvestre, G. D. S. (2018): Anomaly detection and diagnosis for cloud services: practical experiments and lessons learned. *Journal of Systems and Software*, vol. 139, pp. 84-106.

Syu, Y.; Kuo, J. Y.; Fanjiang, Y. Y. (2017): Time series forecasting for dynamic quality of web services: an empirical study. *Journal of Systems and Software*, vol. 134, pp. 279-303.

Thara, D. K.; PremaSudha, B. G.; Xiong, F. (2019): Epileptic seizure detection and prediction using stacked bidirectional long short term memory. *Pattern Recognition Letters*, vol. 128, pp. 529-535.

Tsuda, Y.; Samejima, M.; Akiyoshi, M.; Komoda, N.; Yoshino, M. (2014): An anomaly detection method for individual services on a web-based system by selection of dummy variables in multiple regression. *Electronics and Communications in Japan*, vol. 97, no. 2, pp. 9-16.

Tian, J.; Azarian, M. H.; Pecht, M. (2014): Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. *Proceedings of the European Conference of the Prognostics and Health Management Society*, pp. 1-9.

Wang, T.; Wei, J.; Zhang, W.; Zhong, H.; Huang, T. (2014): Workload-aware anomaly detection for web applications. *Journal of Systems and Software*, vol. 89, pp. 19-32.

Wang, R.; Li, C.; Fu, W.; Tang, G. (2019): Deep learning method based on gated recurrent unit and variational mode decomposition for short-term wind power interval prediction. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1-14.

Wang, J.; Jing, Y.; Qi, Q.; Feng, T.; Liao, J. (2019): ALSR: an adaptive label screening and relearning approach for interval-oriented anomaly detection. *Expert Systems with Applications*, vol. 136, pp. 94-104.

Wang, X.; Du, Y.; Lin, S.; Cui, P.; Shen, Y. et al. (2019): adVAE: a self-adversarial variational autoencoder with Gaussian anomaly prior knowledge for anomaly detection. *Knowledge-Based Systems*, vol. 190, pp. 1-12.

Wielgosz, M.; Mertik, M.; Skoczeń, A.; De Matteis, E. (2018): The model of an anomaly detector for HiLumi LHC magnets based on recurrent neural networks and adaptive quantization. *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 166-185.

Zhang, X.; Meng, F.; Chen, P.; Xu, J. (2016): TaskInsight: a fine-grained performance anomaly detection and problem locating system. *IEEE 9th International Conference on Cloud Computing*, pp. 917-920.

Zhang, Y.; Xiong, R.; He, H.; Pecht, M. G. (2018): Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries. *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5695-5705.

Zheng, Z.; Zhang, Y.; Lyu, M. R. (2010): Distributed QoS evaluation for real-world web services. *IEEE International Conference on Web Services*, pp. 83-90.

Zheng, Z.; Zhang, Y.; Lyu, M. R. (2012): Investigating QoS of real-world web services. *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 32-39.

Zuo, R.; Wang, J.; Chen, G.; Yang, M. (2015): Identification of weak anomalies: a multifractal perspective. *Journal of Geochemical Exploration*, vol. 148, pp. 12-24.