Identifying Bad Software Changes via Multimodal Anomaly Detection for Online Service Systems

Nengwen Zhao*
Tsinghua University; BNRist
Beijing, China

Junjie Chen†
College of Intelligence and Computing, Tianjin University
Tianjin, China

Zhaoyang Yu
Tsinghua University; BNRist
Beijing, China

Honglin Wang
BizSeer
Beijing, China

Jiesong Li
China Guangfa Bank
Guangzhou, China

Bin Qiu
China Guangfa Bank
Guangzhou, China

Hongyu Xu
China Guangfa Bank
Guangzhou, China

Wenchi Zhang
BizSeer
Beijing, China

Kaixin Sui
BizSeer
Beijing, China

Dan Pei
Tsinghua University; BNRist
Beijing, China

ABSTRACT

In large-scale online service systems, software changes are inevitable and frequent. Due to importing new code or configurations, changes are likely to incur incidents and destroy user experience. Thus it is essential for engineers to identify bad software changes, so as to reduce the influence of incidents and improve system reliability. To better understand bad software changes, we perform the first empirical study based on large-scale real-world data from a large commercial bank. Our quantitative analyses indicate that about 50.4% of incidents are caused by bad changes, mainly because of code defect, configuration error, resource contention, and software version. Besides, our qualitative analyses show that the current practice of detecting bad software changes performs not well to handle heterogeneous multi-source data involved in software changes. Based on the findings and motivation obtained from the empirical study, we propose a novel approach named *SCWarn* aiming to identify bad changes and produce interpretable alerts accurately and timely. The key idea of *SCWarn* is drawing support from multimodal learning to identify anomalies from heterogeneous multi-source data. An extensive study on two datasets with various bad software changes demonstrates our approach significantly outperforms all the compared approaches, achieving 0.95 F1-score on average and reducing MTTD (mean time to detect) by 20.4%~60.7%. In particular, we shared some success stories and lessons learned from the practical usage.

CCS CONCEPTS

• **Software and its engineering** → *Maintaining software.*

KEYWORDS

Software Change, Anomaly Detection, Online Service Systems

ACM Reference Format:
Nengwen Zhao, Junjie Chen, Zhaoyang Yu, Honglin Wang, Jiesong Li, Bin Qiu, Hongyu Xu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. Identifying Bad Software Changes via Multimodal Anomaly Detection for Online Service Systems. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3468264.3468543

1 INTRODUCTION

For large-scale online service systems, such as social networking, E-bank, and search engines, engineers need to frequently conduct software changes, aiming to fix bugs, deploy new features, adapt to environmental change, and improve software performance [39, 77]. Because of importing new code or configurations, software changes are more likely to incur service outages, user dissatisfaction, and huge economic loss [6, 7, 9–11, 24, 35]. Based on the experience from Google SRE (Site Reliability Engineering) [4], about 70% of incidents are related to software changes. Therefore, it is essential to avoid incidents and ensure service quality under software changes.

In the literature, tremendous efforts have been devoted to ensuring the quality of software changes, which can be divided into three categories: 1) risk analysis and impact assessment *before deployment* [37, 49, 74], 2) reliable launching strategy *during deployment*, and 3) monitoring performance and identifying bad changes *after deployment* [39, 46, 77, 79]. Although each software change must

*BNRist: Beijing National Research Center for Information Science and Technology
†Junjie Chen is the corresponding author.

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3468543

be rigorously reviewed and extensively tested before deployment (e.g., unit test and integration test), errors and bugs could remain uncaught in the real production environment due to the discrepancies between testing and production environment in cluster size, complex component interactions, resource contention, OS/library versions, and unexpected workload [39, 77]. Thus, it is necessary to monitor service and identify bad changes in time after deployment, which is the target of our work, so that proactive actions could be taken to avoid further outages and economic loss. In the real world, it is time-consuming and error-prone for engineers to check each software change manually. It is because there are hundreds of changes per day in large systems and numerous monitoring data are involved in each software change. To overcome it, some works are proposed to replace manual inspection with an automated anomaly detection algorithm [39, 46, 47, 56, 77]. These existing works mainly focus on the behaviors of business KPIs (Key Performance Indicators, e.g., response time and success rate). Nevertheless, it is a little late to identify anomalies from business KPIs, which may have incurred poor user experience and economic loss.

To better understand the influence and behavior of bad software changes, we conduct the first empirical study based on real-world data from a large commercial bank over the last two years and obtain three key findings. (1) About 50.4% of incidents are caused by changes on average, indicating that software change is indeed failure-prone (RQ1). (2) Abnormal behaviors of bad software changes are very complex and involve heterogeneous multi-source data, including business KPIs [70] (characterizing the status of application layer), machine KPIs [64] (characterizing the status of underlay infrastructure like database, server and middleware, e.g., CPU usage, JVM heap space) and logs [16, 78] (recording detailed running information of the service). Besides, some normal changes could also lead to abnormal but expected behaviors, e.g., resource expansion would bring a decrease in CPU usage and response time (RQ2). (3) The current practice fails to identify abnormal signals earlier hidden in other data sources besides business KPIs, and ignores the expected changes, resulting in unsatisfactory performance (RQ3). To sum up, on the one hand, this study further motivates the necessity of identifying bad changes earlier; on the other hand, it provides us some guidelines to design an effective approach.

In this paper, we propose a novel approach named *SCWarn* to identifying bad changes and producing warning signals earlier by fusing heterogeneous multi-source data. There are two major challenges when designing such an approach. The first one is the lack of sufficient abnormal labeled data since systems tend to run stably. Besides, different bad changes tend to exhibit various abnormal patterns on different data sources. Thus it is challenging to design the approach in supervised ways. The second one is how to extract useful information and identify anomalies from heterogeneous multi-source data involved in software changes accurately. To tackle these challenges, we leverage the idea of multimodal learning to identify anomalies in an unsupervised manner, which has been widely used to learn information from cross modality (e.g., text and images) [59]. More specifically, *SCWarn* consists of four main steps, i.e., data preparation (§3.1), multimodal anomaly detection (§3.2), alerting with analysis report (§3.3), and action decision (§3.4). *SCWarn* first preprocesses raw data to transform heterogeneous data into unified time series based on log parsing [25]. Then, due to the ability of LSTM (Long Short Term Memory) [30] to model on temporal data, *SCWarn* adopts multimodal LSTM to detect anomalies from multi-source data [53, 59]. The core ideas are using LSTM to learn the normal pattern in each single-source time series and utilizing multimodal fusion to capture the inter-correlations among multi-source data. Next, for online detection, equipped with our scenario-specific alerting strategy, the model could generate alerts with reports to notify engineers to pay attention to suspicious software changes. Finally, abnormal but expected changes would be excluded by a policy-based strategy based on a knowledge base. Engineers then confirm the result and take corresponding actions (e.g., rollback) to prevent further service outages.

We conduct an extensive study to investigate the performance of *SCWarn* on two widely-used benchmark systems, i.e., Train-Ticket [65] and E-commerce [18], following the existing work [21, 31, 42, 68, 82]. Specifically, based on some common and typical incidents caused by software changes in the real world (RQ2), we simulate ten types of bad changes (Table 2) to the two systems and construct two datasets, which contain a total of 246 bad changes and 196 normal changes. Our experimental results show that *SCWarn* is able to identify bad changes more accurately and earlier. In detail, our approach could achieve 0.95 F1-score on average, while the average F1-score of three state-of-the-art compared approaches (Gandalf [39], Funnel [77] and Lumos [56]) are only 0.83, 0.77, and 0.80, respectively. The Mean Time To Detect (MTTD) is reduced by 20.4%∼60.7% with *SCWarn*. Besides, the superiority of our multimodal LSTM algorithm is illustrated by comparing it with several related anomaly detection algorithms [16, 32, 64, 70], and the average F1-score is improved by up to 0.27. Also, the contributions and necessity of integrating multi-source data are confirmed by comparing with methods using single data source.

To sum up, this work makes the following major contributions:

- We perform the first large-scale empirical study of incidents induced by bad software changes and obtain some key observations, which largely motivate our work.
- We propose a novel approach named *SCWarn* to identifying bad software changes more accurately and earlier, which could notify engineers to pay attention to bad changes, so as to take proactive actions to avoid further service outages in advance.
- An extensive study shows that *SCWarn* significantly outperforms all the compared approaches, achieving 0.95 F1-score on average and reducing MTTD by 20.4%∼60.7%. Besides, we have released our tool on GitHub [62] for better reproducibility.

2 BACKGROUND AND EMPIRICAL STUDY

2.1 Software Change Management

In online service systems, software changes are frequent and inevitable, aiming to deploy new features, fix bugs, adapt to environmental change, and improve performance. Figure 1 presents a typical procedure of software change management in industry [4], including the following five steps:

- *Problem identification.* During the lifecycle of software, once a problem is identified by engineers through daily operation, monitoring alerts or user complaints, they will conduct a change to fix the problem. The common problems include code bugs, poor performance, error configurations, new requirements, etc.

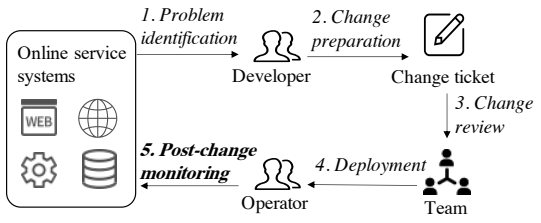


Figure 1: Workflow of software change management

- *Change preparation.* After problem identification, engineers would prepare for a change, including proposing a solution to the problem, developing code, and software testing. Then they will submit a change ticket to service teams, which records some detailed information of this change, such as service system, start time, operations, involved monitoring data, emergency action.
- *Change review.* A group of developers and team leaders will get together to review this change and conduct risk assessment (e.g., how significant this change is, how many users it will affect, and which components it will affect.)
- *Deployment.* After passing the change review, engineers would conduct deployment according to the description in the change ticket. Engineers typically deploy software changes using the "Dark Launching" strategy.
- *Post-change monitoring.* Once the deployment is completed, engineers need to continuously and closely monitor the service performance so as to identify and mitigate bad changes timely.

In this paper, we focus on *post-change monitoring*, aiming to identify bad changes timely and prevent further service outages.

2.2 An Empirical Study of Software Changes

To better understand the characteristics of bad software changes, we conducted the first empirical study based on real-world data from a large commercial bank, aiming to address the following research questions (RQs):

- **RQ1**: What is the percentage of incidents induced by bad software changes?
- **RQ2**: What are the root causes and behaviors of bad changes?
- **RQ3**: Is the current practice of identifying bad software changes good enough?

2.2.1 Study Method. We used five service systems as subjects that are developed and maintained by different product teams. For each service, all incident tickets and change tickets over the last two years are collected and analyzed. Due to the privacy policy, we hid some details, such as the specific number of collected incidents and changes. About ticket analysis and labeling, three authors manually labeled whether an incident is caused by a change (RQ1) and the root cause of a bad change (RQ2) independently through analyzing troubleshooting steps and incident reason description recorded in the ticket under the supervision of "Cohen's Kappa coefficient" [67]. Disagreements were discussed with engineers in charge of the tickets to reach a consensus finally.

2.2.2 RQ1: The Percentage of Incidents Induced by Software Changes. To answer this RQ, we counted how many incidents are caused by software changes according to the study method in §2.2.1. Figure 2 shows the percentage of incidents incurred by software changes in these five services. Clearly, we observe that change is an important factor leading to incidents, accounting for from 39% to 64% (50.4% on average). In addition to this study, we also investigated some public data. As stated in [4] and [77], changes account for about 70% and 54% of incidents in Google and Baidu, respectively. Besides, we also looked at 88 publicly disclosed recent incidents from Google Cloud [12]. Each incident report includes a detailed issue summary, service impact, root cause and remediation action. We found that 37 of the 88 incidents are incurred by changes, accounting for 42%.

To further illustrate the failure-proneness of change, we counted the number of software changes per day and the number of all incidents per day in service $S1$ during two months, as shown in Figure 3. It is clear that the number of incidents has a positive relationship with the number of changes. We conducted Pearson correlation analysis between #Incidents and #Changes. The Pearson coefficient is 0.57 (indicating a moderate positive correlation) with p-value<0.01, which further supports our observation.

In summary, software changes are indeed failure-prone, which could bring great trouble for engineers and customers in software maintenance. Thus, it is imperative to design an automated and accurate approach to identifying bad changes timely and reducing the influence of incidents, which largely motivates our work.

2.2.3 RQ2: Root Causes and Abnormal Behaviors of Bad Changes. Through analyzing all change tickets, we summarized three types of changes, i.e., code change (e.g., adding new features), configuration change and infrastructure-layer change (e.g., replacing hardware devices). In particular, bad changes may be induced by different factors, which we call root causes. In this RQ, following the study method stated in §2.2.1, we summarized five types of root causes including code defect, configuration error, incompatible software version, resource contention and some other factors (e.g., error operations). The percentage of different root causes is shown in Figure 4. It can be observed that code defect and configuration error account for a large proportion, accounting for 69%.

Besides, to further help understand the behaviors of bad changes, we summarize some typical incidents, as shown in Table 1 (the first nine rows). We can observe that various monitoring data (business KPIs, machine KPIs and logs) from multiple sources could be influenced by software changes. Taking the code defect leading to memory leak as an example, the "FullGC" log patterns behave abnormally in Java GC (garbage collection) logs, CPU usage and memory usage would increase, and finally response time would increase. Thus, it is essential to integrate heterogeneous multi-source data to detect bad changes. Another interesting observation is abnormal data behavior does not necessarily mean that this software change is bad. It is because some software changes could lead to abnormal but expected behaviors of monitoring data. For example, replacing an old server with a new high-performance server could incur a decrease in CPU usage and response time. Table 1 (the last three rows) presents some common software changes that could lead to abnormal but expected behaviors. Therefore, how to filter out expected changes and reduce false alarms should be considered.

2.2.4 RQ3: Investigation of Current Practice. The current practice of identifying bad changes in the bank we studied adopts 3-σ [1]

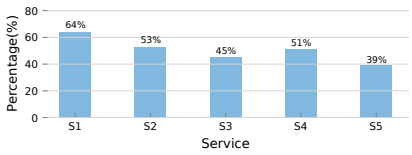


Figure 2: Percentage of incidents induced by changes

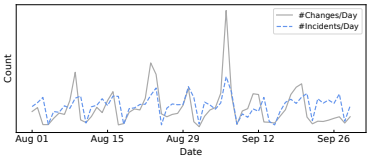


Figure 3: #Incidents per day and #Changes per day

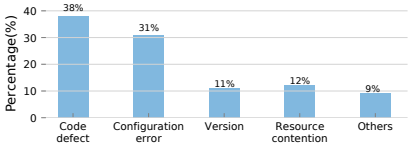


Figure 4: Percentage of different root causes

Table 1: Typical bad/expected software changes and corresponding behaviors of monitoring data

Type	Change operation	Abnormal behaviors of multi-source monitoring data
Bad software change	Configuration error - Wrong IP address	Error messaegs in network logs (e.g., "address conflicted"); related machine KPIs and business KPIs behave abnormally
	Configuration error - Missing modification of correlated configuration [49]	Error messages in application logs; business KPIs behave abnormally
	Configuration error - Deleting white list by mistake	Business KPIs behave abnormally
	Code performance - Slow SQL, full table scan and some related database problems	Database (e.g., active session, lock wait) and related machine KPIs (e.g., disk space, CPU usage) behave abnormally; response time increases
	Code performance - Memory leak	"FullGC" log pattern appears frequently in GC log; machine KPIs (e.g., JVM heap space, memory usage) behave abnormally; response time increases
	Code performance - Code self-loop or dead loop	System load, CPU usage and other machine KPIs behave abnormally; response time increases
	Code logic bug - Wrong database table name; error date format	Error messages in application logs; success rate decreases
	Resource contention	Related machine KPIs (e.g., I/O wait, CPU usage) behave abnormally
Expected software change	Replace high-performance server; Resource expansion [45]	Related machine KPIs (e.g., CPU usage, memory usage) decrease; response time decreases
	Traffic switch	CPU usage decreases
	Code logic changes (e.g., some new steps are added to transaction process)	Related business KPIs behave abnormally (e.g., response time increases)

strategy to inspect the behavior of business KPIs (e.g., response time). Through analyzing incident tickets and discussing with engineers, we found that the performance of the current practice is far from satisfactory. There are three reasons accounting for it. First, as stated in §2.2.3, software changes usually involve multi-source data. Thus, only focusing on business KPIs fails to identify anomalies hidden in other monitoring data, leading to long MTTD (sometimes up to tens of days) and some missing alarms. For example, an incident was noticed five days after the change due to the increase in response time. If integrating multi-source data, however, the incident could be identified earlier from the anomalies of JVM heap space. Second, 3-σ, as well as some other anomaly detection methods (e.g., Holt-winters [72]), is not designed especially for identifying bad changes. Thus, directly applying it ignores the specific scenario and characteristics of software change (e.g., integrating various data, abnormal but expected behaviors, and non-transient anomalies), resulting in some false alarms. The third reason is the drawback of the anomaly detection algorithm itself, since it is hard for 3-σ and other simple statistical anomaly detection methods to deal with complex data. Although there exist some popular products (e.g., Dynatrace [17] and Datadog [13]) in industry used to monitor service, they still fail to overcome the above drawbacks together. To sum up, the current practice of identifying bad changes should be further improved, and it is essential to propose an effective approach to tackle the above drawbacks.

2.3 Summary

Based on the above quantitative and qualitatively analysis, we obtain three key findings:

- Software change is frequent but failure-prone. Thus, it is promising to ensure service quality by catching bad changes timely.
- Software changes usually involve heterogeneous multi-source data. Besides, abnormal behaviors are not necessarily caused by bad changes. Thus, it is crucial to integrate multi-source data and exclude noises when designing an approach.
- Current practice mainly focuses on the behavior of business KPIs and fails to filter out expected changes (false alarms), resulting in poor performance.

These findings support the motivation to identify bad software changes. Thus, it is necessary to propose an effective and automated approach to ensuring service quality under software change, which can identify bad software changes accurately and earlier based on involved multi-source monitoring data and make a decision with timely protective actions to avoid further service outages.

3 APPROACH

The overview of *SCWarn* is presented in Figure 5, which includes four main steps: data preparation, multimodal anomaly detection, alerting with analysis report, and action decision. The key idea of *SCWarn* is drawing support from multimodal anomaly detection to deal with heterogeneous multi-source data. Specifically, the first

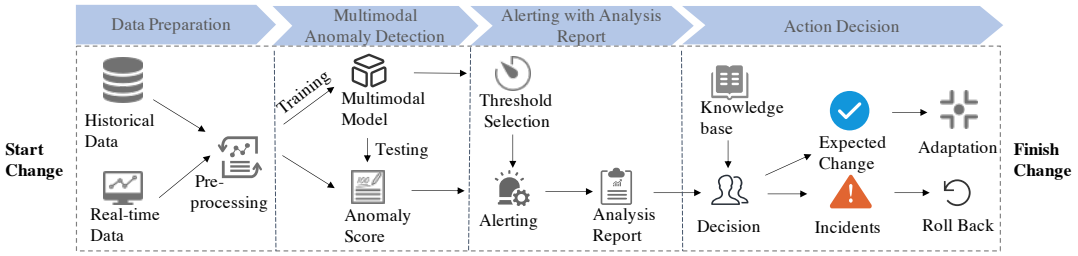


Figure 5: Overview of *SCWarn*

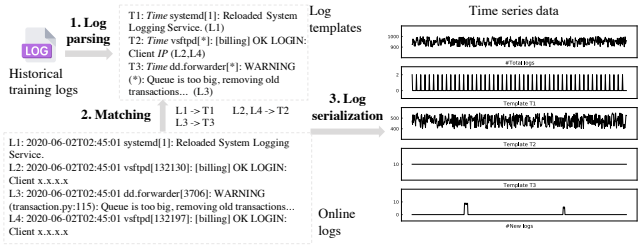


Figure 6: Illustration of log preprocessing

step of *SCWarn* is transforming heterogeneous data into unified time seriesa based on log parsing (§3.1). Then we adopt multimodal learning to detect anomalies from multi-source monitoring data (logs and KPIs). The core ideas are capturing the temporal dependency in each time series via LSTM model [30] and encoding the inter-correlations among multi-source data via multimodal fusion [59] (§3.2). Next, once the anomaly score provided by multimodal LSTM violates our designed alerting rule, a warning signal would be triggered to notify engineers to pay attention to the suspicious change with an interpretable analysis report provided by *SCWarn* (§3.3). After identifying suspicious changes, we design the action decision component to remove expected changes and catch the real bad changes. Finally, engineers could take proactive actions to stop bad changes (e.g., rollback) and prevent further service outages (§3.4). In the following, we will present each step in detail.

3.1 Data Preparation

To characterize the behavior of each software change, we need to collect all related monitoring data because we do not have prior knowledge about what failure this change will cause. For example, the database change could incur database crash or performance degradation, which are reflected in different data sources. As stated in §1, integrating multi-source data not only helps to identify bad changes in advance but also could discover the latent relationship across various data sources, so as to obtain more accurate and comprehensive results. There are three types of data involved in our scenario, i.e., business KPIs (e.g., response time), machine KPIs (e.g., CPU utilization,) and logs, which characterize the running status of the service from multiple aspects. Thus, how to fuse these heterogeneous data is a significant challenge.

Intuitively, KPIs are in the format of time series and can be easily handled, while logs are usually semi-structured or unstructured texts, which are generated using the "print" function with a string template and detailed information as parameters. Typically, logs should be properly parsed for further analysis [26]. We adopt the state-of-the-art log parsing algorithm, Drain [25], to extract log templates, and its superiority and efficiency have been demonstrated in [83]. As shown in Figure 6, given historical raw logs, we first utilize Drain to extract n log templates (the first step in Figure 6 and $n = 3$). When online logs arrive, we could match each log message to the corresponding template (the second step) and count the number of occurrences of each template per minute and acquire n template time series. Meanwhile, the total number of logs and the number of new logs that cannot match the existing templates are also counted (the third step). In general, in the normal state, most online logs could be matched to templates, while the number of new logs would increase in the abnormal state. Thus, after log preprocessing, raw log messages can be transformed into $n + 2$ time series. In this way, all heterogeneous data can be transformed into the format of time series, which will be easily handled in the following model.

3.2 Multimodal Anomaly Detection

After data preparation, it is still challenging to model the multi-source monitoring data since it not only requires capturing the normal pattern of each time series, but also needs to encode the inter-correlations among multi-source data. Another significant challenge is we cannot obtain enough high-quality labeled data and tend to adopt unsupervised approaches. In *SCWarn*, We propose to adopt the technique of multimodal learning in an unsupervised manner to overcome the above two challenges.

In recent years, the ability of LSTM [30] to handle complex temporal or sequential data has ensured its widespread application in domains including natural language processing, speech recognition, and time series forecasting. Compared with traditional Recurrent Neural Network (RNN), LSTM has shown its strong capability to maintain the memory of long-term dependencies due to a context-based weighted self-loop that allows them to forget past information in addition to accumulating it [32]. Therefore, it can learn the relationship between past and current data and has shown remarkable performance in various sequential data. In *SCWarn*, we apply LSTM to capture the temporal dependency in each time series.

To encode the inter-correlations among multi-source data, we draw support from multimodal learning, which is a powerful model for data fusion to acquire the joint representation of different modalities (e.g., images and texts, KPIs and logs in our problem). To the best of our knowledge, there are three popular types of multimodal

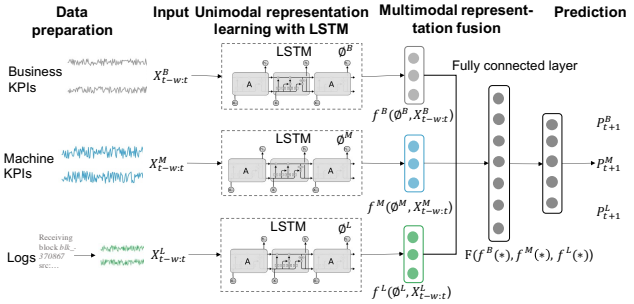


Figure 7: Multimodal anomaly detection model

fusion in the literature [53, 59]. (1) Early (Data-level) fusion, integrating multi-source data into a single feature vector as the model input. (2) Late (Decision-level) fusion, aggregating the decisions from multiple models where each model is trained on a separate modality. (3) Intermediate fusion, which converts raw inputs to a higher-level joint representation by mapping the input through a pipeline of neural network layers [59]. In particular, the majority of existing works adopt intermediate fusion due to its strong ability to learn fused joint information. Inspired by this idea, we also utilize intermediate fusion in our approach to capture the hidden correlations existing in multi-source data.

Overall, the core ideas of our multimodal anomaly detection in *SCWarn* are using LSTM to learn the temporal dependency in each time series and adopting multimodal fusion to encode the inter-correlations among multi-source data. Figure 7 displays the detailed network structure of multimodal LSTM. Specifically, given a window of multi-source data $X_{t-w:t}$, consisting of business KPIs ($X^B_{t-w:t}$), machine KPIs ($X^M_{t-w:t}$) and logs ($X^L_{t-w:t}$). We first apply LSTM to each time series separately to learn the unimodal representation (ϕ is network weight and f is activation function). Then the multimodal joint information is captured by the shared representation layer, which is constructed by merging units with connections coming into this layer from multiple modality-specific paths. Finally, the model output is the predicted value P_{t+1} at time $t+1$ consisting of P^B_{t+1}, P^M_{t+1} and P^L_{t+1}. The anomaly score of data X_{t+1} can be calculated as the absolute value of the difference between real value and predicted value, i.e., $|X_{t+1} - P_{t+1}|$. The loss function of multimodal LSTM can be calculated by the sum of mean square error (MSE) on each modality, which can be formulated as:

$$L = \frac{1}{n}\sum_{i=1}^{n}[(P^B_i - X^B_i)^2 + (P^M_i - X^M_i)^2 + (P^L_i - X^L_i)^2]$$

where n is the number of time steps in the testing set. The detailed hyperparameters of the network are presented §4.1.2. The effectiveness of multimodal LSTM algorithm will be illustrated in §4.3.

3.3 Alerting with Analysis Report

3.3.1 Threshold Selection and Alerting Strategy. After obtaining anomaly scores, a threshold should be selected to decide whether the current time step is abnormal or not. Motivated by [64], during offline training on the monitoring data before the software change, we can compute an anomaly score for each time step in training data and obtain a univariate time series $AS_{train} = \{s_1, s_2, \cdots, s_{n_{train}}\}$. Intuitively, the values of AS_{train} are not very large, and we can determine the threshold based on the distribution of AS_{train}. Specifically, we utilize k-σ [1] principle to get the threshold. If the current anomaly score is larger than $\mu + k\sigma$, we could declare the current time step is abnormal, where μ and σ are the mean and standard deviation of AS_{train}, respectively.

In general, in the field of anomaly detection, if the anomaly score exceeds a pre-defined threshold, an alert will be triggered to notify engineers. In our scenario, however, the anomalies induced by software changes are non-transient and last for a long time without human intervention. Transient anomalies (e.g., temporary network issues) are more likely to be noises. If these noises are reported, it may increase engineers' burdens of manual investigation and cause innocent software changes to be stopped. Therefore, we adjust the alerting strategy, i.e., an alert is generated for a consecutive anomalous points (exceeding the threshold), to remove noises and mitigate false alarms. The effect of the two parameters (k and a) on the performance of *SCWarn* will be discussed in §4.5 in detail.

3.3.2 Analysis Report. After generating alerts to notify responsible engineers to pay attention to suspicious software changes, it would be better to provide an interpretable analysis report, which could enable engineers to have a global view of the software change and inspect related monitoring data conveniently. Figure 8 presents an example of the interpretable report provided by *SCWarn*. Specifically, *SCWarn* can provide some information about the software change ticket and generate the overall health score of this change in real time. The health score is derived from the anomaly score outputted from our multimodal anomaly detection model based on Min-max normalization, which can be computed as follows (x and AS_{train} are anomaly scores of current data and training data):

$$\text{health score} = \max(100 - \frac{x - \min(AS_{train})}{\max(AS_{train}) - \min(AS_{train})}, 0)$$

Furthermore, we rank all involved data by their individual anomaly scores (prediction error of each time series) and display the top-k abnormal data, which may closely relate to the root cause. For example, since the top-1 abnormal data is CPU utilization in Figure 8, we can infer that this alert may be incurred by CPU resource. Also, the data comparison of each time series between before and after the change can be presented in a visual manner, which is also convenient for engineers to investigate more details for troubleshooting.

3.4 Action Decision

As introduced in §2.2.3, some successful changes could also result in abnormal but expected behaviors. Based on the observations from the empirical study, we could construct a knowledge base of expected change operations and corresponding data behaviors in *SCWarn* (e.g., resource expansion would lead to the decrease of CPU usage and response time). Then we can design a policy-based method to distinguish the abnormal behavior is unexpected or not. If the change operation not in the knowledge base, the decision process is conducted by engineers manually, and new scenarios can be updated to the knowledge base. Actually, despite the result recommended by the policy-based method, the final decision still

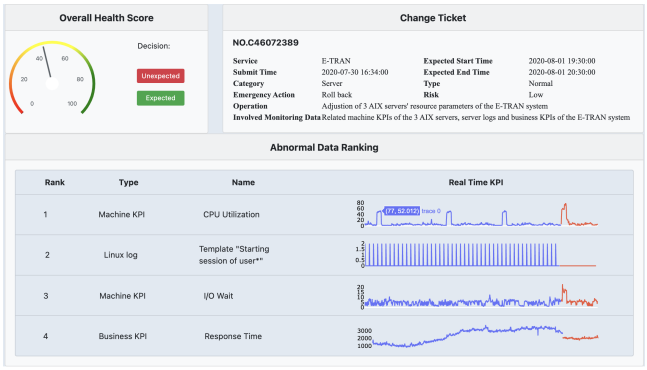


Figure 8: A demo of analysis report provided by *SCWarn*

needs to be confirmed by engineers and cannot be entirely replaced by automated algorithms, since the cost of false positives (innocent changes are stopped and diagnosis time is wasted) and false negatives (missing failures) are huge.

If a software change is confirmed as unexpected, engineers would take prompt protective action (e.g., rollback) to stop the bad change so as to avoid further service unavailability and economic loss. If the abnormal behavior is expected, we could leverage the state-of-the-art adaptation algorithm (StepWise [45]) to enable real-time monitoring system to adapt to new data pattern.

4 EVALUATION

In this study, we aim to address the following research questions:

- **RQ4**: What is the effectiveness of *SCWarn* in identifying bad software changes?
- **RQ5**: What is the effectiveness of multimodal LSTM?
- **RQ6**: What is the time efficiency of *SCWarn*?
- **RQ7**: What is the impact of parameters on the result of *SCWarn*?

4.1 Experiment Setup

4.1.1 Datasets and Metrics. To evaluate the performance of *SCWarn*, we conducted the study on two widely-used benchmark systems.

Benchmark Systems. The first system is Train-Ticket [65], an open-source microservice system, which has been widely applied in the literature [31, 42, 68, 82]. Train-Ticket serves as a system of selling train tickets, containing more than 30 microservices (pay, price, order, food, etc.). The other one is E-commerce, an end-to-end application benchmark for e-commerce search system [18]. It covers the major modules and critical paths of an industry scale e-commerce provider. We run the two systems on Kubernetes [36], which is an open-source system for automating deployment, scaling, and management of containerized applications.

Bad Software Changes Injection. As presented in §2.2.3, we find that major root causes of bad software changes include code defect, error configurations, incompatible software versions, and resource contention. Based on the key observation, to evaluate the effectiveness of *SCWarn* in identifying bad software changes, we carefully designed and simulated ten types of bad software change operations in the study, where all the above four root causes are involved. Table 2 presents detailed descriptions of bad software change operations designed by us. Besides, we also simulated some normal changes for evaluation, including correct configuration modification and importing new code without bugs. These software changes are injected into different components of each benchmark system, ensuring the diversity of our datasets.

Table 2: Types and descriptions of bad software changes we injected on the benchmark systems for evaluation

Failure type	Description
Code defect	*F1* - Create large Java objects in program, leading to frequent fullGC and OutOfMemory error
	F2 - Inject delay into program to simulate code performance issue
	F3 - SQL statement defect leading to slow query
Configuration error	*F4* - Invalid paths which will be opened or executed
	F5 - Unsuitable size of JVM heap memory
	F6 - Database port error
	F7 - Limited number of database connections
	F8 - Non-existent database table
Software version	*F9* - Incompatible software version
Resource contention	*F10* - CPU contention

Data Collection. For each benchmark system, KPI data are collected by Prometheus [57] and stored in InfluxDB [33], and logs are collected by Logstash [43] and stored in ElasticSearch (ES) [19]. We collected two datasets $\mathcal{A}$ and $\mathcal{B}$ based on the two benchmark systems. Dataset $\mathcal{A}$ contains 121 bad software changes and 96 normal changes, and dataset $\mathcal{B}$ contains 125 bad changes and 98 normal changes. There are 12~13 cases for each failure type (Table 2) in each dataset. For each software change case, the involved multi-source data including business KPIs, machine KPIs and logs are used for experiments.

Metrics. Intuitively, identifying bad software changes is a binary classification task. We run *SCWarn* and compared approaches to see if bad software changes can be caught successfully. We utilized popular binary classification metrics, i.e., *precision, recall* and *F1-score* as metrics. Besides, we also considered the mean time (minute) to detect bad changes (*MTTD*) after change deployment, which indicates the ability of *SCWarn* to identify bad changes in advance.

4.1.2 Implementations and Parameters. We used two weeks of data before the software change as training data and the online data after the change as testing data. Specifically, for multimodal LSTM, we used one LSTM layer (sequence length is 10; hidden size is 128) and one fully connected (FC) layer on each time series separately. Then one FC layer is used to learn the joint representation. After that, two FC layers are used to obtain the final results. The hidden sizes of all FC layers are set to 64. During training, Adam optimizer with a learning rate of 0.01 is adopted, the batch size is 64, and the number of epochs is 50. Besides, we found that the final results are insensitive to hyperparameters within a small range. For the compared approaches, we used the parameters provided by their papers and open-source code [39, 56, 64, 70, 77]. Besides, we have publicly published our experimental code on GitHub [62] for better reproducibility. All approaches are implemented by Python with widely-used libraries, including NumPy [54], pandas [55], scikit-learn [61], Pytorch [58], etc.

Table 3: Precision (P), recall (R), F1-score (F1) and MTTD (minutes) comparison between *SCWarn* and baselines

	Dataset $\mathcal{A}$				Dataset $\mathcal{B}$			
Approaches	P	R	F1	MTTD	P	R	F1	MTTD
SCWarn	**0.91**	**0.95**	**0.93**	**5.1**	**0.97**	**0.98**	**0.97**	**2.3**
Gandalf–AD	0.68	0.95	0.79	6.2	0.77	0.99	0.87	3.1
Funnel	0.77	0.69	0.73	14.0	0.76	0.87	0.81	6.4
Lumos	0.66	0.94	0.78	10.0	0.77	0.93	0.82	10.0
mFunnel	0.69	0.93	0.79	9.0	0.82	0.79	0.80	3.0
mLumos	0.85	0.83	0.84	10.0	0.72	0.99	0.83	10.0

4.2 RQ4: Performance of Identifying Bad Software Changes

We compared *SCWarn* with the three following baseline approaches.

- Gandalf [39] is proposed for end-to-end safe deployment and contains an anomaly detection (AD) component based on KPIs and logs. For logs, it utilizes error messages clustering to extract error patterns (e.g., 404 code) and detects anomalies on each log error pattern via Holt-winters. Considering the details about KPI anomaly detection in Gandalf are unclear [39], we also utilized Holt-winters on KPIs in the experiments.
- Funnel [77] adopts improved Singular Spectrum Transform (iSST) to detect change point on business KPIs after software changes.
- Lumos [56] uses A/B testing to compare the KPI pattern during a time window (10 minutes in our experiments) before and after the change by statistical hypothesis testing. Considering that χ^2-test in [56] is suitable to categorical data, we adopted t-test in experiments since it is more appropriate and has been widely used to time series [38, 44].

Table 3 displays the results between *SCWarn* and compared approaches. Clearly, *SCWarn* performs the best taking all measurements into consideration, achieving 0.93 and 0.97 F1-score on two datasets. In comparison, the average F1-score of Gandalf, Funnel and Lumos are only 0.83, 0.77 and 0.80, respectively. The average precision and recall of *SCWarn* achieve 0.94 and 0.96, indicating that *SCWarn* could detect bad changes accurately with few false negatives (leading to missing failures and degrading service quality) and false positives (engineers' efforts to diagnose are wasted and innocent changes are stopped). Besides, the MTTD of *SCWarn* is 3.7 minutes on average, reducing by 20.4%∼63.7%, illustrating that *SCWarn* could identify bad changes earlier. Furthermore, considering both Funnel and Lumos only target at business KPIs, to compare fairly with our approach, we also extended their anomaly detection algorithms to multi-source data (mFunnel and mLumos in Table 3). We observe that F1-score and MTTD of mFunnel and mLumos are slightly improved compared with raw methods, indicating that fusing multi-source information is indeed helpful to some degree, while they still perform worse than *SCWarn* due to the drawbacks of their algorithms (iSST and t-test) as explained below.

We further analyzed the reasons why the compared approaches perform not well. Although Gandalf utilizes both KPIs and logs, it detects anomalies separately on single data source and fails to identify the hidden correlations. Thus, its overall performance is weaker than *SCWarn*. Besides, it adopts Holt-winters to detect anomalies on each log error pattern. On the one hand, Holt-winters is only applicable to seasonal KPI. On the other hand, error message clustering is exclusively designed for logs recording error information, which is not generic for all log data. In comparison, our log preprocessing technique has the ability to deal with various logs. For Funnel, it uses iSST to detect change point after the change, while iSST is not generic for various time series (seasonal, variational and stationary). Besides, iSST requires accumulating enough data, leading to long MTTD. In terms of Lumos, it simply adopts statistical testing for anomaly detection, which is only applicable to data with significant change. Besides, t-test relies on enough new data to compare with old data before the change to obtain accurate results, and the window size is set to 10 data points in our experiments. It is difficult for t-test to achieve short MTTD and high accuracy at the same time. Therefore, *SCWarn* fusing multi-source data based on the multimodal anomaly detection could deliver better results.

Another interesting observation is *SCWarn* performs differently in different types of bad changes, especially in MTTD. Specifically, the MTTD of some failure types (*F4*, *F6*, *F8* and *F9*) is shorter than others. This is because these bad software changes tend to behave abnormally immediately after the deployment (e.g., error database port would directly cause the service cannot access the database). In comparison, some other failures would perform a slow deterioration process (e.g., creating large Java objects would incur JVM heap space to rise slowly until it overflows), resulting in longer MTTD.

Overall, taking both F1-score and MTTD into consideration, *SCWarn* with good interpretability is indeed able to identify bad software changes accurately and timely.

4.3 RQ5: Effectiveness of Multimodal LSTM

Actually, identifying bad changes is an anomaly detection task, aiming to detect abnormal patterns after change deployment. To demonstrate the effectiveness of our multimodal LSTM algorithm, we compared it with several state-of-the-art anomaly detection methods, including Donut [70] and LSTM [30] for business KPIs (B-LSTM), LSTM-NDT [32] and OmniAnomaly [64] for machine KPIs, and DeepLog [16] for logs. Besides, we also implemented several methods integrating multi-source data based on raw Auto-encoder (M-AE) [23], raw LSTM (M-LSTM) [32] and multimodal AE [53], to further compare with our algorithm. Although these algorithms are not specially designed for identifying bad software changes, they can be extended to solve the problem.

The performance comparison on two datasets is shown in Table 4 and we obtained two key observations. First, multimodal LSTM is superior to those approaches only relying on single data source. It is because incorporating multi-source data could identify the failure signals hidden in various data sources in advance. Besides, the results provided by multimodal LSTM could also be more accurate when fusing comprehensive information. Notice that DeepLog performs poorly since it endeavors to detect sequential anomalies of log templates, which is unsuitable in our problem. The second finding is multimodal LSTM outperforms other methods (M-AE, M-LSTM and multimodal AE) that also incorporate multi-source data. Specifically, both directly applying LSTM without multimodal fusion (average F1-score of M-LSTM is 0.92) and replacing LSTM with AE to model on the single data source in multimodal LSTM (average F1-score of multimodal AE is 0.91) perform worse than

Table 4: Effectiveness of multimodal LSTM algorithm compared with existing anomaly detection approaches

		Dataset $\mathcal{A}$				Dataset $\mathcal{B}$			
Data source	Approaches	P	R	F1	MTTD	P	R	F1	MTTD
Business KPIs	Donut	0.65	0.92	0.76	7.3	0.94	0.75	0.83	5.4
	B-LSTM	0.86	0.75	0.80	8.2	0.99	0.88	0.93	6.6
Machine KPIs	LSTM-NDT	0.80	0.71	0.76	5.2	0.85	0.83	0.86	3.5
	OmniAnomaly	0.71	0.99	0.83	5.4	0.88	0.87	0.87	3.2
Logs	DeepLog	0.57	0.96	0.71	11.2	0.55	0.83	0.66	8.9
Multi-source data	M-AE	0.79	0.85	0.81	5.1	0.82	0.90	0.86	2.6
	M-LSTM	0.80	0.95	0.87	5.3	0.99	0.94	0.96	3.2
	Multimodal AE	0.94	0.83	0.88	6.1	0.95	0.93	0.94	4.0
	Multimodal LSTM	**0.91**	**0.95**	**0.93**	**5.1**	**0.97**	**0.98**	**0.97**	**2.3**

our algorithm (0.95). Considering that the performance of multimodal LSTM has a small difference with M-LSTM and multimoal AE on dataset $\mathcal{B}$, we split $\mathcal{B}$ into 10 sub-datasets based on failure types (Table 2) and calculated F1-score on each sub-dataset. Following existing works [8, 48], we conducted a paired sample Wilcoxon signed-rank test [69] and Vargha-Delaney effect size measure [66] to analyze the difference degree between multimodal LSTM and M-LSTM/multimodal AE. The p-values are 0.0416 (<0.05) and 0.0019 (<0.05), and the effect size are 0.6860 (medium difference) and 0.8388 (large difference), demonstrating the superiority of multimodal LSTM.

4.4 RQ6: Time Efficiency

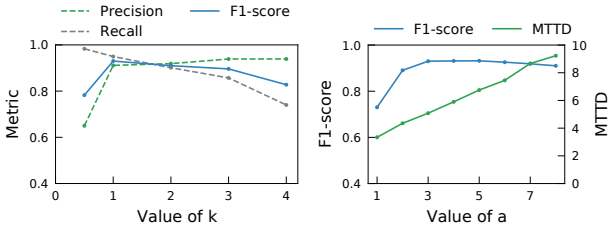
As an approach to identifying bad software changes, time efficiency is a vital factor. For incoming data, if the detection result cannot be provided in time, it could cause that engineers cannot identify bad changes and take protective actions immediately. Thus, we investigated the efficiency of *SCWarn* and compared approaches, including training time and detection time (Funnel and Lumos do not need to train models). As displayed in Table 5, the training time of *SCWarn* is about several minutes, which is acceptable compared with OmniAnomaly and DeepLog. Despite LSTM model used in *SCWarn*, the training cost is relatively short because of the lightweight and effective network structure. Considering numerous software changes per day in large systems, short training time could significantly reduce resource overhead. Besides, since the training phase is offline, training cost cannot lead to the delay in identifying bad software changes. In terms of the detection time, *SCWarn* and most compared approaches could give a result nearly in real time (less than 1 second), which is negligible. Overall, *SCWarn* has acceptable offline training time and negligible online detection time.

4.5 RQ7: Parameter Sensitivity

Here, we take dataset $\mathcal{A}$ as the subject and mainly discuss the effect of two parameters in threshold selection and alerting strategy on *SCWarn*. As stated in §3.3.1, the value of k in k-σ strategy directly influences the threshold selection and final results. Figure 9(a) presents the effect of the value of k on the performance of identifying bad changes. Clearly, with small k, the threshold is relatively loose, leading to high recall but low precision. On the contrary, the threshold is strict with large k, leading to high precision but low recall. Another parameter we studied is the number of continuous

Table 5: Training time (minutes) and detection time (seconds) comparison

	Dataset $\mathcal{A}$		Dataset $\mathcal{B}$	
Approaches	Training	Detection	Training	Detection
SCWarn	**4.84**	**0.94**	**3.43**	**0.86**
Gandalf–AD	6.28	0.78	6.03	0.74
Funnel	-	1.44	-	1.04
Lumos	-	1.23	-	1.08
Donut	11.57	1.01	8.32	0.84
B-LSTM	3.45	0.55	2.89	0.49
LSTM-NDT	3.89	0.61	3.01	0.52
OmniAnomaly	28.11	1.34	24.89	1.88
DeepLog	50.65	2.43	46.23	2.77
M-AE	3.11	0.40	2.80	0.26
M-LSTM	4.42	0.49	4.19	0.44
Multimodal AE	3.65	0.67	3.20	0.60



(a) k in threshold selection (b) The number of continuous points

Figure 9: The effect of parameters

anomalous points (α) in alerting strategy. Figure 9(b) presents its effect on F1-score and MTTD. Intuitively, too small α will generate some false alarms caused by noises (e.g., temporal network issue). In comparison, although large α can achieve satisfactory precision, too strict the alerting strategy could result in long MTTD. k and α are set to 1 and 3 in our experiments, respectively. In practice, parameter selection can be decided based on the validation set.

5 DISCUSSION

5.1 Success Stories

To illustrate the practical effectiveness of *SCWarn*, we conducted an informal user study with 12 engineers from different service

systems in the bank. These engineers provided several historical cases to us, and we reported the results of *SCWarn* to them. In particular, engineers appreciated *SCWarn* from three perspectives. 1) *SCWarn* could identify bad changes more accurately and earlier than their traditional tool based on 3-σ [1] (§2.2.4), so that protective actions (e.g., rollback) can be taken immediately to prevent further economic loss. 2) Engineers confirmed the analysis reports provided by *SCWarn* could assist them in investigating these cases with good interpretability [14]. Taking the case II below as an example, the report displays all abnormal signals of this change, and the top-3 abnormal data are heap space, #"GC operation" log template and memory usage. Engineers could easily infer that the incident is caused by frequent FullGC and localize the code defect in 5 minutes incorporating Java dump files. Besides, displaying all related information of this change in a web page could save about 5~20 minutes for engineers to log in different devices or platforms (e.g., Grafana [2] and Kibana [3]) to inspect data manually. 3) Incorporating multi-source data could provide more comprehensive information. Specifically, KPIs could directly reflect the health status of the change entity and logs could provide clues for further diagnosis. For example, using GC logs could indicate the frequent FullGC problems in case II; using network device logs could help localize the IP address conflict problem incurred by error configuration (the appearance of new log pattern "address conflict detected"). They also gave us some suggestions to make our tool more user-friendly, e.g., displaying system topology in the user interface and incorporating monitoring data from correlated systems. In the following, we present four typical real cases and anonymize some confidential details.

Case I: A new database table imported by a software change was not indexed, and then a full table scan was conducted. This incident was noticed by the alert of high response time during busy hours. With *SCWarn*, however, the incident can be discovered 23 minutes in advance by some related machine KPIs, including database metrics (lock wait and CPU usage) and middleware metrics (JDBC connection pool). In contrast, the MTTD of Gandalf is 12 minutes longer than that of *SCWarn*.

Case II: A piece of code imported by a software change was defective and created a large java object, incurring frequent fullGC operations and full heap space. Engineers noticed this incident by the alert of high response time. In comparison, *SCWarn* could identify the incident 5.6 hours in advance from GC logs and related machine KPIs, including JVM metrics (heap space and GC count) and server metrics (CPU usage and memory usage). Besides, the details in GC logs could provide clues for further troubleshooting.

Case III: After a software upgrade of service A, it shared the same server with service B for data synchronization, resulting in an efficiency bottleneck of A when B was busy accessing this server. Our approach could discover (57 minutes earlier compared with traditional monitoring tool) and locate the problem through related server performance KPIs (I/O wait time is high) of service A.

Case IV: The slow SQL statement in the new code led to the abnormal behavior of related database performance KPIs (the number of average active sessions is high). Our approach could capture it accurately and rapidly (6 minutes in advance) to avoid further negative impact on service availability.

5.2 Lessons Learned

Generality of our approach. It is a common phenomenon that bad changes could incur service incidents, both in our studied bank and other large companies like Google [4] and Baidu [77] (§2.2.2). Besides, *SCWarn* based on multi-source data anomaly detection is generic since most systems include KPIs and logs which have been extensively studied in the literature [27, 34, 50, 60, 64, 70, 78]. Thus, the problem of identifying bad software changes and *SCWarn* are not limited to the bank we studied. Besides, *SCWarn* could also be applied to incident diagnosis and incident prediction. Similar to identifying bad changes, these two tasks also aim to detect abnormal signals from monitoring data and existing works mainly focus on single data source independently [40, 41, 71, 76]. Thus, fusing KPIs and logs may provide more comprehensive information to diagnosis or prediction, which can be our future work.

Fully automated software change without human involvement is difficult. Although our proposed *SCWarn* could significantly reduce manual efforts on identifying bad changes, it is unrealistic to conduct software changes entirely automatically without any human involvement. It is because only the engineers in charge of the software change have requisite domain knowledge of the expected behavior and can make accurate decisions (e.g., whether the abnormal behavior is expected or not, and how to take emergency actions to mitigate unexpected incidents). Actually, such domain expertise cannot be replaced by a fully automated algorithm accurately.

Multi-source data fusion may be helpful to other software engineering tasks. In our work, integrating multi-source data in an interpretable and visual manner could help engineers obtain a global view of the software change. Similarly, multi-source data fusion can also be applied to other tasks, aiming to improve accuracy and extend features, for example, incident linking based on incident tickets and graph dependency [10], alert prioritization using textual alerts and KPIs [80], trace anomaly detection combining texts and KPIs [52], flaw detection in software programs [28].

The experience obtained from our work could provide some insights for software testing. Software testing is vital to guarantee code quality and service reliability under development. In practice, engineers could investigate the root cause of the bad software change (Figure 4) and analyze whether the incident could be avoided by more comprehensive testing, so as to obtain some lessons for software testing. In this way, engineers can take effective actions to improve the testing to cover more aspects, for example, adding stress testing to simulate the impact of the unexpected burst of user requests on the service and testing the associated system to remove the incompatibility or resource contention problem.

5.3 Threats to Validity

Subject systems: We conducted the experimental study on two popular benchmark systems and injected various bad software changes (Table 2). In fact, these failure types we collected from industry may be limited and cannot cover all scenarios. Thus, the failure cases in experiments, as well as the scale of benchmark systems, are threats in our study. Besides, the injected changes used for evaluation are synthetic and relatively simple, which could introduce bias. In the future, we will further evaluate our approach using more large-scale real-world software change cases with various failure types.

Data quality: Although the two microservice systems used for evaluation are widely applied, some hidden bugs existing in systems or other external factors (e.g., network flapping) would threaten the quality of collected data. Thus, the experimental data may contain some noises, and the performance may be affected. To reduce this threat, we have deployed the systems with high availability and checked our data carefully.

Evaluation metrics: To demonstrate the effectiveness of *SCWarn*, we used precision/recall/F1-score and MTTD as measurements, which have been widely adopted in existing works. In the future, to reduce this threat, we will further analyze the performance of *SCWarn* on different types of bad changes in detail and consider more comprehensive metrics (e.g., FPR/TPR) to more sufficiently evaluate the effectiveness and efficiency.

5.4 Limitations and Future Work

We describe several limitations of our approach. One limitation is some silent incidents caused by software changes are challenging to detect. For example, code defects hidden in seldom-used functions might go undetected for a long time. The reason is that *SCWarn* is a data-driven approach, relying on abnormal patterns on monitoring data. If some bad changes do not behave abnormally on monitoring data, they cannot be caught by *SCWarn*. Another limitation is real large-scale deployment. Currently, we just reviewed some historical incident cases induced by software changes to illustrate the practical effectiveness. Due to some limitations, however, we only deployed *SCWarn* in a service system of the bank and large-scale deployment in the real world can be our future work.

6 RELATED WORK

Software Change. In the literature, considerable efforts have been dedicated to software change in academic and industry, which can be divided into three categories: 1) impact and risk analysis before change [37, 49, 74], for example, Sonu et al. [49] proposed to prevent bugs and misconfigurations via correlated change analysis; 2) reliable launching strategy during deployment (e.g., dark launching); 3) identifying bad changes after deployment [22, 39, 47, 77]. In this work, we endeavor to tackle the third way because some bugs and errors could remain uncaught due to the discrepancies between testing and the production environment. Thus closely monitoring system behavior after deployment is a vital task. The core idea of existing works about identifying bad changes is adopting an anomaly detection (or change point detection) algorithm to detect abnormal behaviors of business KPIs after deployment, for example, iSST adopted by Funnel [77], CUSUM used by Mercury [47], and A/B testing in Lumos [56]. However, all of these approaches only utilize single-source monitoring data. Thus, they fail to discover the failure signals hidden in other data sources and cannot obtain comprehensive results, resulting in unsatisfactory performance. Gandalf [39] consumes various data (KPIs and logs) to ensure safe deployment, while it adopts a separate anomaly detection model for each single-source data and cannot capture the inter-correlations among multi-source data, and the anomaly detection algorithm in Gandalf is not generic. Besides, the goal of Gandalf is to localize which change should be responsible for the abnormal behavior, not for identifying bad changes. In our approach, we leverage multimodal data fusion for identifying bad software changes based on heterogeneous multi-source data. The superiority of *SCWarn* compared with existing works has been illustrated in §4.2.

Anomaly Detection. Identifying bad changes is also an anomaly detection task. Over the years, there has been a great deal of effort spent on anomaly detection of KPIs and logs [27, 34, 41, 50, 60, 64, 70, 73, 78, 81]. About KPI anomaly detection, Xu et al. [70] proposed Donut to apply Variational Auto-encoder (VAE) to detect anomalies in seasonal KPIs. Su et al. [64] proposed an approach to detect anomalies on multivariate KPIs through Stochastic RNN. In terms of logs, Zhang et al. proposed LogRobust [78] to extract semantic information and utilize an attention-based Bi-LSTM model to identify log anomalies. In industry, some products[13, 17] also have the ability to monitor service and detect anomalies from KPIs using some statistical methods. The technique of anomaly detection has also been widely applied in other software engineering fields, such as misbehavior detection for autonomous driving systems [29, 63, 75], detecting security issues [15, 16], identifying workflow errors [20, 51], and detecting issues from programming language compilers [5]. However, existing anomaly detection algorithms or tools are not specially designed for identifying bad changes and ignore some characteristics of the problem (e.g., fusing multi-source data, abnormal but expected behaviors), as stated in §2.2.4. In our approach, through integrating multi-source data involved in software changes and adopting multimodal anomaly detection, *SCWarn* delivers better performance than existing works.

7 CONCLUSION

In online servicer systems, software change is frequent but failure-prone. To better understand bad software changes, we conduct the first empirical study based on large-scale real-world data from a large commercial bank. Our qualitative and quantitative analysis show about 50.4% of incidents are caused by changes on average, and the current practice of identifying bad changes is unsatisfactory due to ignoring multi-source data involved in the change. Therefore, it is necessary to identify bad changes to prevent service outages. Towards this direction, we propose a novel approach named *SCWarn* to identifying bad changes accurately and timely. The core idea of *SCWarn* is drawing support from multimodal learning to detect anomalies from heterogeneous multi-source data. An extensive study including various bad software changes confirms the effectiveness of *SCWarn* (average F1-score is 0.95 with a short MTTD), which significantly outperforms all the compared approaches.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (Grant No.2019YFE0105500), the State Key Program of National Natural Science of China under Grant 62072264, the Beijing National Research Center for Information Science and Technology (BNRist) key projects, and National Natural Science Foundation of China 62002256.

REFERENCES

[1] 3-sigma rule. https://en.wikipedia.org/wiki/68-95-99.7_rule. [Online; accessed 10-Feb-2021].

[2] Grafana. https://grafana.com/. [Online; accessed 10-Feb-2021].

[3] Kibana. https://www.elastic.co/kibana. [Online; accessed 10-Feb-2021].

[4] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems.* " O'Reilly Media, Inc.".

[5] Timofey Bryksin, Victor Petukhov, Ilya Alexin, Stanislav Prikhodko, Alexey Shpilman, Vladimir Kovalenko, and Nikita Povarov. 2020. Using Large-Scale Anomaly Detection on Code to Improve Kotlin Compiler. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 455–465. https://doi.org/10.1145/3379597.3387447

[6] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. 111–120. https://doi.org/10.1109/ICSE-SEIP.2019.00020

[7] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *34th IEEE/ACM International Conference on Automated Software Engineering*. 364–375. https://doi.org/10.1109/ASE.2019.00042

[8] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical Accuracy Estimation for Efficient Deep Neural Network Testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–35. https://doi.org/10.1145/3394112

[9] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems. In *35th IEEE/ACM International Conference on Automated Software Engineering*. 373–384. https://doi.org/10.1145/3324884.3416624

[10] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, et al. 2020. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 304–314. https://doi.org/10.1145/3368089.3409768

[11] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497. https://doi.org/10.1145/3368089.3417055

[12] Google Cloud. [n.d.]. https://status.cloud.google.com/summary.

[13] Datadog. [n.d.]. https://www.datadoghq.com/. [Online; accessed 10-Feb-2021].

[14] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).

[15] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1283–1297. https://doi.org/10.1145/3319535.3363226

[16] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1285–1298. https://doi.org/10.1145/3133956.3134015

[17] Dynatrace. [n.d.]. https://www.dynatrace.com/. [Online; accessed 10-Feb-2021].

[18] E-commerce. [n.d.]. https://github.com/alibaba/eCommerceSearchBench. [Online; accessed 10-Feb-2021].

[19] Elasticsearch. [n.d.]. https://github.com/elastic/elasticsearch. [Online; accessed 10-Feb-2021].

[20] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158. https://doi.org/10.1109/ICDM.2009.60

[21] Wanling Gao, Fei Tang, Lei Wang, Jianfeng Zhan, Chunxin Lan, Chunjie Luo, Yunyou Huang, Chen Zheng, Jiahui Dai, Zheng Cao, et al. 2019. AIBench: an industry standard internet service AI benchmark suite. *arXiv preprint arXiv:1908.08998* (2019).

[22] Aitor Gartziandia. 2021. Microservice-based Performance Problem Detection in Cyber-Physical System Software Updates. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 147–149. https://doi.org/10.1109/ICSE-Companion52605.2021.00062

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[24] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing? Lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 1–16. https://doi.org/10.1145/2987550.2987583

[25] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40. https://doi.org/10.1109/ICWS.2017.13

[26] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: system log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 207–218. https://doi.org/10.1109/ISSRE.2016.21

[27] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. IEEE Computer Society, 207–218. https://doi.org/10.1109/ISSRE.2016.21

[28] Scott Heidbrink, Kathryn N Rodhouse, and Daniel M Dunlavy. 2020. Multimodal Deep Learning for Flaw Detection in Software Programs. *arXiv preprint arXiv:2009.04549* (2020).

[29] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Cristofer Englund, Sankar Raman Sathyamoorthy, and Stig Ursing. 2019. Towards structured evaluation of deep neural network supervisors. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 27–34. https://doi.org/10.1109/AITest.2019.00-12

[30] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[31] Xiaofeng Hou, Jiacheng Liu, Chao Li, and Minyi Guo. 2019. Unleashing the Scalability Potential of Power-Constrained Data Center in the Microservice Era. In *Proceedings of the 48th International Conference on Parallel Processing* (Kyoto, Japan) *(ICPP 2019)*. Association for Computing Machinery, New York, NY, USA, Article 10, 10 pages. https://doi.org/10.1145/3337821.3337857

[32] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 387–395. https://doi.org/10.1145/3219819.3219845

[33] InfluxDB. [n.d.]. https://github.com/influxdata/influxdb. [Online; accessed 10-Feb-2021].

[34] Mohammad S Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. 2021. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 150–159. https://doi.org/10.1109/ICSE-SEIP52600.2021.00024

[35] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420. https://doi.org/10.1145/3368089.3417054

[36] Kubernetes. [n.d.]. https://kubernetes.io/. [Online; accessed 10-Feb-2021].

[37] Steffen Lehnert. 2011. *A review of software change impact analysis.* Univ.-Bibliothek.

[38] Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, et al. 2020. Predictive and Adaptive Failure Mitigation to Avert Production Cloud {VM} Interruptions. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 1155–1170.

[39] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, et al. 2020. Gandalf: An Intelligent, End-To-End Analytics Service for Safe Deployment in Large-Scale Cloud Infrastructure. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 389–402.

[40] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. Predicting Node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 480–490. https://doi.org/10.1145/3236024.3236060

[41] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 102–111.

[42] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 48–58. https://doi.org/10.1109/ISSRE5003.2020.00014

[43] LogStash. [n.d.]. https://github.com/elastic/logstash. [Online; accessed 10-Feb-2021].

[44] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1176–1189. https://doi.org/10.14778/3389133.3389136

[45] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 13–24. https://doi.org/10.1109/ISSRE.2018.00013

[46] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. 2011. Rapid detection of maintenance induced changes in service performance. In *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*. ACM, 13. https://doi.org/10.1145/2079296.2079309

[47] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2011. Detecting the performance impact of upgrades in large operational networks. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 303–314. https://doi.org/10.1145/1851182.1851219

[48] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 94–105. https://doi.org/10.1145/2931037.2931054

[49] Sonu Mehta, Ranjita Bhagwan, Rahul Kumar, Chetan Bansal, Chandra Maddila, B Ashok, Sumit Asthana, Christian Bird, and Aditya Kumar. 2020. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 435–448.

[50] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs.. In *IJCAI*. 4739–4745.

[51] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 215–224. https://doi.org/10.1145/2939672.2939712

[52] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly detection from system tracing data using multimodal deep learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 179–186. https://doi.org/10.1109/CLOUD.2019.00038

[53] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *ICML*.

[54] NumPy. [n.d.]. https://numpy.org/. [Online; accessed 10-Feb-2021].

[55] pandas. [n.d.]. https://pandas.pydata.org/. [Online; accessed 10-Feb-2021].

[56] Jamie Pool, Ebrahim Beyrami, Vishak Gopal, Ashkan Aazami, Jayant Gupchup, Jeff Rowland, Binlong Li, Pritesh Kanani, Ross Cutler, and Johannes Gehrke. 2020. Lumos: A Library for Diagnosing Metric Regressions in Web-Scale Applications. *arXiv preprint arXiv:2006.12793* (2020).

[57] Prometheus. [n.d.]. https://prometheus.io/. [Online; accessed 10-Feb-2021].

[58] PyTorch. [n.d.]. https://pytorch.org/. [Online; accessed 10-Feb-2021].

[59] D. Ramachandram and G. W. Taylor. 2017. Deep Multimodal Learning: A Survey on Recent Advances and Trends. *IEEE Signal Processing Magazine* 34, 6 (2017), 96–108. https://doi.org/10.1109/MSP.2017.2738401

[60] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3009–3017. https://doi.org/10.1145/3292500.3330680

[61] scikit learn. [n.d.]. https://scikit-learn.org/.

[62] SCWarn. [n.d.]. https://github.com/FSEwork/SCWarn. [Online; accessed 24-Feb-2021].

[63] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour Prediction for Autonomous Driving Systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 359–371. https://doi.org/10.1145/3377811.3380353

[64] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA. https://doi.org/10.1145/3292500.3330672

[65] Train-Ticket. [n.d.]. https://github.com/FudanSELab/train-ticket/. [Online; accessed 10-Feb-2021].

[66] András Vargha and Harold D Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[67] Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med* 37, 5 (2005), 360–363.

[68] T. Wang, W. Zhang, J. Xu, and Z. Gu. 2020. Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications With Statistics. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2350–2363. https://doi.org/10.1109/TNSM.2020.3022028

[69] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. Springer, 196–202.

[70] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, 187–196. https://doi.org/10.1145/3178876.3185996

[71] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. 2018. Improving service availability of cloud systems by predicting disk error. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 481–494.

[72] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Dan Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. 2012. Argus: End-to-end service anomaly detection and localization from an isp's point of view. In *2012 Proceedings IEEE INFOCOM*. IEEE, 2756–2760. https://doi.org/10.1109/INFCOM.2012.6195694

[73] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised Log-based Anomaly Detection via Probabilistic Label Estimation. In *43rd IEEE/ACM International Conference on Software Engineering*. 1448–1460. https://doi.org/10.1109/ICSE43902.2021.00130

[74] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. 2020. Check before You Change: Preventing Correlated Failures in Service Updates. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 575–589.

[75] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 132–142. https://doi.org/10.1145/3238147.3238187

[76] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. 2018. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 2.

[77] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. 2015. Rapid and robust impact assessment of software changes in large internet-based services. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2. https://doi.org/10.1145/2716281.2836087

[78] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, and et. al. 2019. Robust Log-Based Anomaly Detection on Unstable Log Data *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 807–817. https://doi.org/10.1145/3338906.3338931

[79] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-time incident prediction for online service systems. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 315–326. https://doi.org/10.1145/3368089.3409672

[80] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2420–2429. https://doi.org/10.1109/INFOCOM41043.2020.9155219

[81] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, and et.al. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* (2019). https://doi.org/10.1109/TNSM.2019.2919327

[82] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* (2018). https://doi.org/10.1109/TSE.2018.2887384

[83] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19)*. IEEE Press, 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021