Identifying Erroneous Software Changes through Self-Supervised Contrastive Learning on Time Series Data

Xuanrun Wang[†], Kanglin Yin[†], Qianyu Ouyang[†], Xidao Wen[†], Shenglin Zhang[‡]
Wenchi Zhang[§], Li Cao[§], Jiuxue Han[¶], Xing Jin[¶], Dan Pei[*†]
[†]Tsinghua University, {xr-wang20, oyqy19}@mails.tsinghua.edu.cn, {yinkanglin, wenxidao, peidan}@tsinghua.edu.cn
[‡]Nankai University, zhangsl@nankai.edu.cn, [§]Bizseer, zhangwenchi@bizseer.com, xyzcaoli@outlook.com
[¶]China Construction Bank, {hanjiuxue.zh, jinxing.zh}@ccb.com

***Abstract*—Software changes are frequent and inevitable. However, erroneous software changes may cause failures and incidents, degrading user experience and system stability. Thus, it is critical to distinguish erroneous software changes from normal ones. Our empirical study from a global data center reveals that erroneous software changes have caused nearly one-third of the critical incidents in the last two years. Some quantitative results also imply that the number of software changes and that of the Key Performance Indicator (KPI) time series related to a software change are relatively large. Based on the observations, we propose *Kontrast*, a self-supervised, generic and adaptive approach using contrastive learning, aiming to identify erroneous software changes on time. Its key idea is to compare pre-change and post-change KPI time series related to the software change, assuring the time series is still in a normal state after the software change. Since contrastive learning approaches need a fully-labeled dataset, we propose a novel data augmentation technique inspired by self-supervised learning to generate data with pseudo labels. Our model significantly outperforms all the compared approaches on two datasets with a millisecond-level speed for each KPI and is proven to obtain cross-dataset adaptability. To better certify our contribution, we also exhibit some success cases of *Kontrast* from its deployment.**

***Index Terms*—Software Change, Contrastive Learning, Time Series**

I. Introduction

Software change is a major method for operators to add new features, fix bugs, or update configurations in large-scale online service systems. As the software change introduces new codes or configurations into the current running system, some **erroneous software changes** may incur incidents or failures even though these changes have been carefully analyzed and reviewed, possibly causing service outages and user dissatisfaction or even large economic losses. A survey [1] reports that over 70% of the incidents were directly or indirectly attributed to software changes in Google. Another report [2] points out that in October 2021, Facebook services were abruptly down for several hours in many countries, leading to a $60 million loss, whose root cause was an erroneous software change of the backbone network routers. Therefore, it is critical to detect incidents in the online service system induced by erroneous software changes quickly. Current approaches to improving the reliability of software changes are typically threefold, including impact assessment and risk analysis *before the software change* [3]–[5], optimizing the deployment strategy *during the software change* [6], [7], and identifying whether it is erroneous *after the software change* via monitoring performance [8]–[10]. Since there could be unknown differences between production and development environments, defects including bugs or misconfigurations would remain concealed during the testing phase before the software change [1], possibly causing incidents once deployed. Key Performance Indicators (**KPI**s, e.g., success rate, CPU usage) are essential for erroneous software change identification since they reflect the system's running status from different levels [8], [9]. Only monitoring after the software change is capable of identifying these hidden defects; therefore, we aim to perform post-change monitoring using KPI data.

To better understand software changes, we conducted a comprehensive empirical study based on real-world data collected from a global data center, revealing the following observations. 1) The erroneous software changes have caused roughly one-third of critical incidents in online services. 2) The number of concurrent software changes is rather large, and that of KPIs related to a software change is tremendous, causing millions of KPIs should be checked simultaneously. 3) To identify erroneous software changes, operators manually compare KPI time series before and after the software change based on their experience. 4) Current practices ignore the sequential order information of the compared KPI time series segments or require too many computation resources.

In the literature, to identify erroneous software changes, supervised learning approaches (e.g., Opprentices [11]) have a plausible accuracy but need a manually-labeled dataset, which is costly to obtain. Unsupervised learning approaches (e.g., SCWarn [8], Gandalf [10], FluxRank [12]) can work with unlabeled data. However, these approaches aim to learn normal patterns from independent KPIs; thus have to train a dedicated model for each KPI time series before the testing phase [8], suffering from extremely high training costs. Statistics approaches (e.g., t-Test, k-sigma [13]) run rapidly, but they

[*] Dan Pei is the corresponding author.

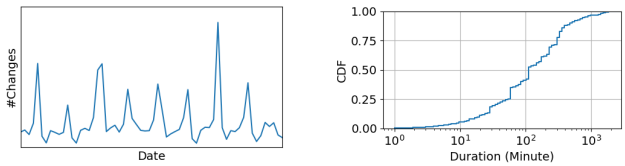
ignore the sequential order or the physical meanings of different KPIs, leading to unsatisfying performances. In summary, current approaches are not capable of solving these significant challenges: 1) **Lack of labeled data**: It is costly for operators to label anomalous KPI manually. 2) **Massive quantity of KPIs**: The number of KPI time series to be checked within several minutes is tremendous, requesting a highly efficient approach. 3) **Heterogeneous KPIs**: Time series with various physical meanings or collected from heterogeneous instances diverge on anomaly criteria. For example, a 10% decrease in *success rate* is much more critical than in *memory usage*. Thus they cannot be compared simply with the same set of rules.

Inspired by the operators' practices, we propose a novel self-supervised contrastive learning approach, named *Kontrast* (**K**PI C**ontrast**), to identify erroneous software changes by comparing the KPI time series before and after the software change deployment. We apply the self-supervised learning mechanism to tackle the problem of lacking labeled data, generating data with pseudo labels from the unlabeled via noise pattern injection. Since *Kontrast* is a comparison-based neural network approach, time series inputs from different KPIs can be processed simultaneously in a batch, dramatically increasing efficiency. A specially designed noise intensity mechanism is applied to learn different anomaly criteria for KPIs of heterogeneous nature, making it applicable for KPIs of varying kinds.

To evaluate the performance of *Kontrast*, we conduct comprehensive experiments on two datasets. Our experimental results suggest that *Kontrast* identifies erroneous software changes and anomalous KPI time series more accurately (0.01 to 0.08 on F1-score) and much faster (100 to 1,000 times) than the compared non-contrastive learning approaches. The results also certify the adaptability of *Kontrast*. In summary, this work makes the following contributions:

- We have done a comprehensive empirical study on erroneous software changes from a global data center, whose key observations motivated us to propose *Kontrast* and address the challenges.
- We propose *Kontrast*, an efficient, generic, and self-supervised contrastive learning approach. *Kontrast* aims to identify erroneous software changes, outperforming all the compared approaches with a millisecond-level speed for each KPI. *Kontrast* can also be trained and tested on different datasets while outperforming the majority of current models.
- We propose a novel KPI data augmentation approach based on noise pattern injection. Besides, for better reproductivity, we opensource *Kontrast* on GitHub [14].

The rest of this paper is structured as follows. In Section II, we report the comprehensive empirical study to understand erroneous software changes better and clarify our motivation. We introduce *Kontrast* in Section III, and its evaluation results in Section IV. In Section V, we discuss success application stories and limitations of our work. Finally, we review the current basis of academic research of several related domains in Section VI and conclude our paper in Section VII.



(a) The number of software changes per day in a two-month period.

(b) CDF of the deployment duration of a software change.

Fig. 1: Data collected from the empirical study.

II. Empirical Study

We conducted an empirical study on real-world data to understand the software change practices in the industrial environment. This section shows our observations on software changes and current practices of erroneous software change identification. Due to the security and privacy issues, the exact numbers reported in this study are masked.

A. Data Collection

We manually analyzed change tickets and incident reports from a global data center over a two-year period. Each change ticket records the software change's deployment period, change objectives, changed systems, etc.

B. Software Changes in the Online Service System

Software change is a vital part of the online service system. The online service system consists of thousands of applications, each supporting a particular set of online services. Typical types of software changes include bug fixing, new feature enhancement, configuration updates, etc.

To identify an erroneous software change, operators are expected to monitor a large number of KPIs closely after the software change. Software changes are frequent, especially for those critical applications. The daily count of the software changes is shown in Fig. 1(a), whose peak value is about one thousand. Because stability is crucial in the data center, the frequency of software changes is once per two weeks. There could be more than two hundred applications deploying software changes concurrently. For an application, there are hundreds of instances to keep high availability. Once a software change on a particular application is deployed, operators use hundreds of system-level (e.g., CPU usage) and service-level (e.g., transaction count) KPIs to monitor its instances and know their running status. Considering all the factors, for a software change, operators should check more than ten thousand KPIs. Thus, up to two million KPIs require to be checked concurrently during the peak. Operators expect to identify the erroneous software changes before a significant loss emerges; namely, the time for algorithms to identify erroneous software changes is limited to merely several minutes.

The software change deployment process. The process of software change deployment is displayed in Fig. 2. Since the scale of applications is enormous, the process takes some

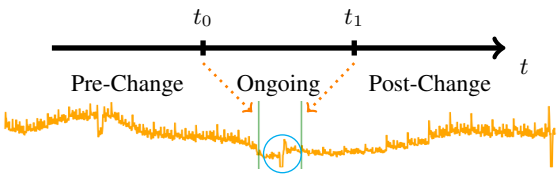


Fig. 2: The software change process. The orange line below shows a real-world KPI time series. The vertical green lines indicate the software change's start and end time points, respectively. KPI fluctuations can be found inside the circle in *ongoing period*.

time to finish all deployment actions. This time is called *ongoing period* in our paper, whose distribution is shown in Fig. 1(b). During *ongoing period*, we notice some KPI time series are influenced by the software changes irregularly, which is probably caused by service restart, traffic shift, etc. We denote t_0 as the start time point of the software change and t_1 as the finish time point, and *ongoing period* is the period between t_0 and t_1. t_1 denotes the time when all the related KPIs have reached stability, given by the input. We call the period before t_0 as *pre-change period*, and the one after t_1 as *post-change period*.

C. Erroneous Software Change Identification Approaches in Current Practice

Since the online service systems typically have their own Service Level Objectives (SLO), operators monitor closely for anomalies in the KPI time series. In the data center, several monitor systems (e.g., Prometheus [15]) collect KPI time series data from related service instances. Once several change-related KPI time series are considered anomalous, an incident ticket will be raised, labeling the software change as erroneous, requiring the immediate attention of operators. The root causes of erroneous software changes include code logical bugs, configuration errors, slow SQL queries, etc., each of which has a set of particular patterns on its related KPIs. Due to privilege issues, we cannot show detailed root cause analysis results here.

Operators' current approach to identifying anomalous KPIs is relatively simple: compare the post-change KPI time series segments (usually a short time span right after deployment) to the pre-change ones (the contemporaneous historical data of the post-change time series segment or a short time span right before deployment), checking whether the KPI time series segments are close. Some lightweight comparison algorithms, like t-Test, k-sigma [13] and Kernel Density Estimation (KDE) [16], are used to meet the efficiency requirements. However, the accuracy of these algorithms is unsatisfactory because these statistics-based algorithms ignore the sequential order information of the KPI time series. Thus, as a critical supplementary to the identifying process, manual verification of the current algorithms' outputs is needed to achieve reasonable accuracy. Operators judge the KPI time series based on their domain knowledge, which is primarily influenced by the KPI's physical meanings. For example, they treat success rate more strictly than memory usage; a 10% decrease in success rate is regarded as anomalous, while a 10% decrease in memory usage is not. However, as the quantity of KPI time series falsely alarmed by current approaches becomes too large, human labor is far from enough.

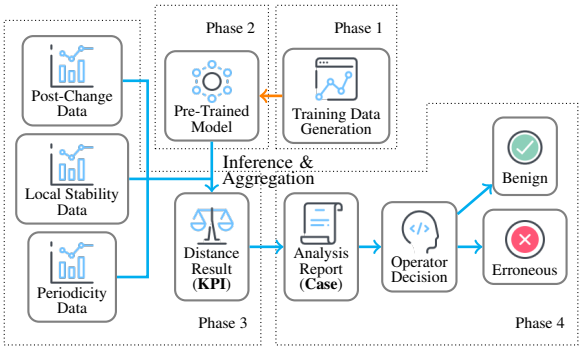


Fig. 3: An overview of *Kontrast*.

In addition, common approaches [8], [10] identifying erroneous software changes are not capable of handling cross-dataset adaptation situations. When running on a new application, there is seldom historical data. Thus, it is difficult to determine the parameters of these approaches. Operators seldom have appropriate tools to identify erroneous software changes in this situation. Nevertheless, adaptive models can learn general anomaly patterns from other systems' software change data, thus making accurate decisions in this new application.

In conclusion, a desired erroneous software change identification model should meet all the requirements below. 1) **Efficient**. The model should be capable of processing a massive number of KPIs in a short time with low delay. 2) **Accurate**. Stability is the primary consideration of the online service system. Thus, accurately identifying the erroneous software change is critical. 3) **Adaptive**. Models that are trained based on one application and can be used on another are preferred. As current approaches fail to satisfy all the requirements, we propose *Kontrast*, aiming to provide a solution meeting all the demands.

III. Approach

We propose a novel approach called *Kontrast*. In *Kontrast*, we first determine whether each change-related KPI time series is normal after the change deployment. Then decide whether the software change introduces defects according to the results of its related KPIs. Fig. 3 shows the overview of *Kontrast*.

Kontrast is composed of four phases. 1) To deal with the lack of labeled data, we apply noise pattern injection to generate pseudo-labeled KPI time series segment pairs for model training (Section III-B). 2) Upon the foundation of the generated data in phase 1, we train a set of generic comparison-based models (Section III-C). 3) When a software change is deployed, we extract specific post-change KPI time series segments (consecutive fragments of a KPI time series) and compare them with pre-change KPI time series segments using the model in phase 2 (Section III-D). 4) We aggregate the results to judge whether the software change is erroneous or not and send reports to operators (Section III-D).

Before introducing the four phases, we first exhibit our data extraction specifications for KPI time series segments.

A. Data Extraction Specifications

We found that a normal KPI time series roughly exhibit two properties: **periodicity** [17], [18] and **local stability** [19] through empirical observations. In real-world production environments, a large proportion of the KPIs follow a certain periodicity, which are also called *seasonal KPI*s in many other works [20]–[22]. These KPIs are closely related to user behavior and people's work and rest routines. To ensure a KPI time series is expected after a software change, checking whether it remains its periodicity is essential. If not, the KPI time series falls into an abnormal state, indicating a potential defect introduced from the software change process.

Another observation is that a KPI time series rarely changes abruptly in typical situations. Whenever a sudden change occurs, an anomaly is highly likely to reveal [9], [20]. This property is called local stability in our paper. Based on this observation, local stability should be kept before and after the software change if the software change is benign and relatively short. Moreover, according to our interview with the operators, local stability holds in both seasonal and non-seasonal KPIs.

We then extract the following three parts from a KPI time series to validate the properties:

1) X': Given the software change starts at t_0 and ends at t_1, we can get the post-change KPI time series data by $X[t_1+1, t_1+\omega]$ (short for $\{x_{t_1+1}, x_{t_1+2}, \cdots, x_{t_1+\omega-1}, x_{t_1+\omega}\}$), which is denoted as X'. Here X is the KPI time series, and ω is the inspection window size.

2) X^P: Denote T is the period of a given KPI time series; namely, T is the minimum interval of the recurrence of the general pattern. Our model treats T as a given input value, and this value can be easily inferred by expert knowledge or autocorrelation algorithms, which is out of our research domain. Typically in the business scenarios, including ours, T is one day. We use $\{X[t_1+1-\delta, t_1+\omega-\delta] \mid \delta \in \Delta\}$ where Δ contains a set of time intervals, typically including $1T, 2T, 3T, 7T, 14T, 21T$, etc. For example, if a software change deploys on a Monday, then the contemporaneous data from the last Friday ($3T$) reflects the newest pattern on weekdays, and the ones from the weekend ($1T, 2T$) show the latest pattern, the ones from the past Mondays ($7T, 14T, 21T$) reflect weekly patterns possibly due to weekly-scheduled tasks. For each aspect, three samples are enough to reflect the respective patterns because *Kontrast* uses minimum distance as the aggregation strategy (Section III-D). Thus, any one of the samples reflecting the normal pattern is sufficient.

We collect these KPI time series segments and denote them as X^P, here $\cdot^P$ is the abbreviate of periodicity.

3) X^{LS}: We extract $X[t_0-\omega, t_0-1]$ as X^{LS}, which displays the data pattern just before the software change deployment. Compare X^{LS} with X', and we can certify the local stability property. Here $\cdot^{LS}$ is the abbreviate of local stability.

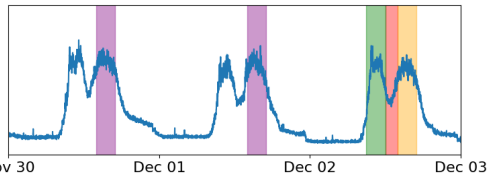


Fig. 4: Illustration of extraction specifications. Red shadow shows the *ongoing period*, orange shows X', green shows X^{LS} and purple shows X^P.

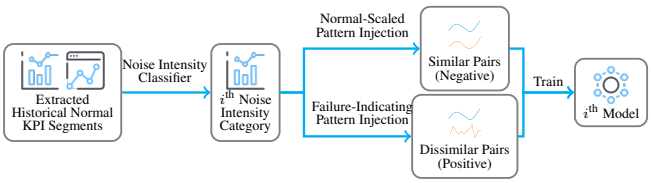


Fig. 5: Training data preparation of *Kontrast*.

Till now, we obtain the X^P, X^{LS}, and X'. Checking if X' follows the patterns inferred by X^P and X^{LS}, we can decide whether the KPI time series is anomalous. A brief illustration of the extraction phase can be found in Fig. 4.

B. Training Data Generation

To train the contrastive learning model in *Kontrast*, we need first to collect labeled training data containing similar and dissimilar KPI time series segment pairs. However, manually labeling large-scale training data is costly and time-consuming, thus not realistic. Self-supervised learning is a suitable tool to solve this problem by creating self-defined pseudo labels as supervision and learning the representations of the data [23]. Inspired by that, we propose a novel data augmentation approach whose architecture is shown in Fig. 5. It uses pre-change KPI time series data and generates sufficient data with pseudo labels, positive (dissimilar) and negative (similar), for the model to train. This approach contains two major components, as introduced below.

1) Noise Intensity Classifier: The KPI time series with different physical meanings may have divergent anomaly criteria. Even for the KPIs that share similar physical meanings being collected from different service instances, the patterns of anomaly may be diverse. Thus, using a single model to deal with all the KPIs is unrealistic. Thus, we need to make a simple yet effective classification first, dividing the KPI time series into several categories by their historical characteristics and checking them with different criteria.

To this end, a noise intensity classifier is required. Denote the noise intensity N as the divergence between the data on a specific time point and those on the same time point in other periods. The overall noise intensity of a time series is the average noise intensity of each specific time point, as (1) shows.

$$\begin{aligned} N_{X,i} &= std(X_i, X_{i+T}, X_{i+2T}, \cdots) \\ N_X &= \frac{1}{T}\sum_{i=0}^{T-1} N_{X,i} \end{aligned} \tag{1}$$

In (1), $std(\cdot)$ stands for standard deviation function, and N_X denotes the noise intensity of KPI time series X. For each of

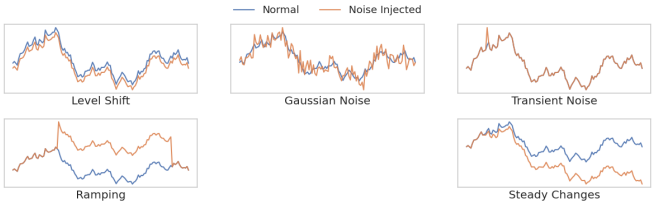


Fig. 6: Noise pattern injection modules. Modules in the second row show positive pair injection while the others show negative ones.

the X, an N_X is derived. We divide all the KPI time series into K categories based on their N. The noise intensity of a KPI time series implies its anomalous judgment threshold. We build a dataset and train a dedicated model for each noise intensity category.

2) Pair Selection & Noise Pattern Injection: The data generation procedures of each noise intensity category are similar, hence we take one category for example.

We aim to build a dataset containing similar and dissimilar time series segment pairs. Our solution is to construct pseudo-labeled pairs from the unlabeled KPI time series. As patterns of X^P and X^{LS} are not the same, we should build a model to compare X' with X^P, and another to compare X' with X^{LS}, which are called P model and LS model, respectively.

First, we introduce the data generation approach for P model. Based on the periodicity property, we randomly select these contemporaneous pairs from a KPI time series for **negative cases** to simulate X^P and X', aiming to cover all the possible time spans in the KPI time series, improving the comprehensiveness of the dataset. For **positive cases**, methods are twofold. The first is to select pairs ignoring the time information randomly. The second is to select a random segment and inject intense noises, making it dissimilar to its original pattern. A simple Euclidian distance filtering is also needed to ensure the pairs are indeed dissimilar. Furthermore, to improve the robustness of the contrastive model, mild and noise-category-specific noises should be injected into the selected data pairs both in positive cases and negative cases.

For the LS model, approaches are similar. The only difference is the pair selecting method in the negative case generation approach. Here, a random-lengthed *ongoing period* starting from an arbitrary time point is selected. Its front and back are extracted as the negative pair, simulating X^{LS} and X'.

The noise pattern injection method consists of several modules, each injecting one kind of failure pattern. We refer [24] and use the failure pattern from it, and conclude five modules of our noise pattern injection, namely, Level Shift, Gaussian Noise, Transient Noise, Ramping, and Steady Change. Their pattern demonstrations can be found in Fig. 6. These modules are not mutually exclusive, and we utilize multiple of them on one KPI time series. The intensity of the noise pattern injection is randomly generated, aiming to cover noise intensities from the slightest to the strongest.

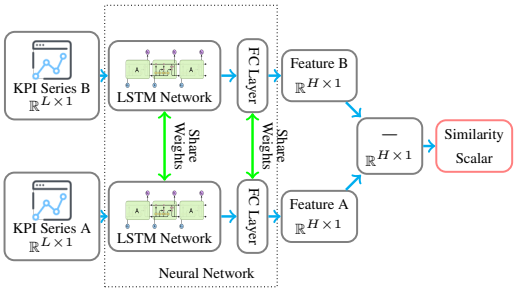


Fig. 7: The neural network design of *Kontrast*.

C. Model Design

To determine whether the KPI time series after the change is abnormal, we propose a novel contrastive learning model to make the comparison. As introduced, given the processed KPI time series, the next is to check the similarity between X' and X^P, X' and X^{LS} respectively. If X' does not follow the pattern inferred by X^P and X^{LS}, the KPI time series is judged as anomalous. Neural network architecture is applied to compare time series segments. This is because neural networks can fit distributions of arbitrary complexity, including encoding complex features of time series patterns. Moreover, neural networks support testing in batch, largely improving the degree of parallelism. The core idea of our model is to use Long Short-Term Memory (LSTM) [25] as the feature extractor, Siamese network architecture as the comparator, with the foundation of pseudo-labeled time series pairs, to learn the similarity computation principles of time series, which is shown in Fig. 7.

Siamese network [26] uses covariant-shared feature extractors to parse two input series simultaneously, producing two features. It is a classical model used for comparing two objects. The contrastive loss function (2) introduced in [27] affects the outputs of the Siamese network along with a provided label. It enforces the feature extractor to learn the similarities of the time series data when the label Y is 0 (negative case) and the differences when the label is 1 (positive case). Here M denotes a *margin* to avoid overfitting. $\overrightarrow{x_1}, \overrightarrow{x_2}$ are the input samples, and $\mathcal{L}(\mathcal{F},\cdot,\cdot,\cdot)$ is the contrastive loss function using the encoding of neural network setup $\mathcal{F}$.

$$\mathcal{L}(\mathcal{F}, Y, \overrightarrow{x_1}, \overrightarrow{x_2}) = (1-Y)\frac{1}{2}(\|\mathcal{F}(\overrightarrow{x_1}) - \mathcal{F}(\overrightarrow{x_2})\|_2^2) + (Y)\frac{1}{2}(\max\{0, M - \|\mathcal{F}(\overrightarrow{x_1}) - \mathcal{F}(\overrightarrow{x_2})\|_2\})^2 \quad (2)$$

The ability of LSTM in processing complex time series data has been proved by many emerging approaches nowadays [8], [28]. Inspired by the effectiveness of LSTM, we apply it to extract rich information about the shapes and patterns from the two input time series parallelly. The twin-tower architecture from Siamese network requires parameters from the two feature extractors to be shared during training and testing. Thus, the criteria for extracting useful information in the KPI time series are unified. The feature extractors' outputs are then sent to a fully connected module, encoded, and converted to a distance-comparable expression. In the

training phase, output of the fully connected layer is sent to (2); while in the testing phase, distance between two input samples is obtained via a simple subtraction, namely:

$$\mathcal{D}(\mathcal{F}, \overrightarrow{x_1}, \overrightarrow{x_2}) = \|\mathcal{F}(\overrightarrow{x_1}) - \mathcal{F}(\overrightarrow{x_2})\|_2^2 \quad (3)$$

where $\mathcal{D}(\mathcal{F}, \cdot, \cdot)$ is the distance function with the neural network setup $\mathcal{F}$.

Having obtained the distance function result of the KPI time series segment pairs, next, we should aggregate the results to a verdict of anomalous KPI.

D. Erroneous Software Change Identification

When a software change has been deployed, multiple related KPIs will be sent to *Kontrast*, requesting a check. Here, the KPIs being monitored include the ones of updated services and the influenced services, i.e., upstream and downstream services. Each input KPI time series is classified into a specific noise intensity category c by its historical characteristics and then performed data extraction, obtaining its X^P, X^{LS} and X'.

Then, we use the pre-trained P model and LS model of noise intensity category c (i.e., $\mathcal{F}_c^P$ and $\mathcal{F}_c^{LS}$) to get their distance predictions. Since there are multiple X^Ps, we have a series of $\mathcal{D}(\mathcal{F}_c^P, X_i^P, X')$. We use (4) to get the final result of the distance function:

$$pred = \min_i \{\mathcal{D}(\mathcal{F}_c^P, X_i^P, X')\} + \alpha \mathcal{D}(\mathcal{F}_c^{LS}, X^{LS}, X') \quad (4)$$

where α is a hyper-parameter, fusing the results from two models to a unified final distance result $pred$. We use a minimum function based on the observation: some historical data may contain anomalies; therefore, if X' is similar to one of its contemporaneous time series segments, it is regarded as normal. The KPI time series is abnormal if $pred$ is greater than a pre-set threshold. Finally, if a certain number of the KPI series are considered abnormal, the change would be regarded as erroneous. An alert will be raised and immediately reported to the operators in this scenario.

IV. Evaluation

To demonstrate the superior performance and the adaptability of *Kontrast*, we conducted experiments to answer the following research questions (RQs).

- ***RQ1***: What is the performance of *Kontrast*?
- ***RQ2***: What is the time efficiency of *Kontrast*?
- ***RQ3***: How does *Kontrast* adapt to different datasets?
- ***RQ4***: How do the main components contribute to *Kontrast*'s performance?
- ***RQ5***: What is the influence of the related hyper-parameters?

A. Experiment Setup

1) Dataset: To evaluate the performance of *Kontrast*, we prepared two different datasets, $\mathcal{A}$, and $\mathcal{B}$. Dataset $\mathcal{A}$ consists of 368 KPI time series, while dataset $\mathcal{B}$ contains 336 software change cases and 34,944 related KPI time series.

Table I: Dataset Properties.

Property	**Dataset $\mathcal{A}$**	**Dataset $\mathcal{B}$**
#KPI	368	34,944
KPI positive ratio	50%	13.6%
#Change case	-	336
Change case positive ratio	-	48.8%
Avg. #sample of KPI	about 6,000	about 7,200

Dataset $\mathcal{A}$ is based on AIOps2018 Challenge [29], a public dataset for anomaly detection that has been widely utilized in the works analyzing time series [30]–[32]. Since the original dataset only has an anomaly label (normal/anomaly) for each time point without software change information, a preprocessing step is required to apply it to our task. To generate positive cases, we first extracted the consecutive normal segments followed by an abnormal segment. Then, we randomly chose a point as a "software change timepoint" in the tailing part of the normal segment. The normal segment before the point is *pre-change period* and both the normal and the abnormal segments after the point belong to *post-change period*, simulating the failure occurs sometime after the software change ends. This is based on the observation that the pattern of anomalies in dataset $\mathcal{A}$ are similar to those in erroneous software change cases in Section V-A. For negative cases, we arranged software change timepoints inside the normal segments and made sure that there were no failures in the randomly-lengthed *post-change period*. Note that the KPI time series segment pairs in dataset $\mathcal{A}$ are individual; thus, there is no software change case information in each case, and it is only used for KPI-level experiments. The process of dataset generation is repeated for 3 times without manual intervention. Results of dataset $\mathcal{A}$ are averaged to reduce the influence of randomness.

Dataset $\mathcal{B}$ is collected from hundreds of change deployment experiments on a popular [33]–[35] microservice benchmark system: Hipster Shop [36]. The microservice-based architecture allows us to inject erroneous software changes into assorted microservices. Concerning the root causes of the erroneous software changes analyzed in our empirical study, we implemented 32 different problematic microservice versions, e.g., adding dead loops to the service source code, modifying the network configurations of a service, attaching extra random delay into the SQL query functions, etc. For each software change case, a version switch on specific microservices based on Kubernetes [37] was performed. Prometheus [15] continuously collected KPI time series data related to the change. For each software change case, an *ongoing period* with a random length that follows the distribution in Fig. 1(b) exists. The collected KPIs included CPU usage, memory utilization, network flow rate, service success rate, transaction count, and transaction reply time cost. Two authors independently labeled the KPI segments as anomalous or not for evaluating the performance of *Kontrast* in anomalous KPI identification. Disagreements during labeling were solved by discussion.

Detailed information of the datasets is shown in Table I.

2) Metrics: Our experiment contains two tasks. One is to classify whether a KPI is anomalous, and the other is to predict whether a software change is erroneous, both of which

are binary classification tasks. The former is the basis of the latter, thus experimented individually. We regard anomalous KPI or erroneous software change case as a positive sample, using precision, recall, and F1-score to report the performance. Besides, we measure the training time and testing time of *Kontrast* and other compared approaches to evaluate the efficiency. Note that since there is no change case information in dataset $\mathcal{A}$, we do not perform erroneous software change identification on dataset $\mathcal{A}$.

B. Compared Approaches

The compared approaches can be divided into three categories.

1) End-to-end erroneous software change identification approaches; take the software change information (i.e., software change deploy time, related KPIs) as the input and identify whether the software change is erroneous.

- **SCWarn** [8] State-of-the-art erroneous software change identification algorithm based on multivariate LSTM.
- **Gandalf** [10] End-to-end safe deployment framework using Holt-Winters [38].

2) Comparison approaches that directly compare time series (i.e., **DTW** (Dynamic Time Warping) [39], **Pearson** Correlation Coefficient, **Lumos** [40], **TS-CP^2** [41]); their core ideas are to compute the similarity of two time series (before and after a software change), which are similar to *Kontrast*. We used them as the substitution for our Siamese network module introduced in Section III-C, building a P model and an LS model for these algorithms with an aggregation, respectively.

3) Anomaly detection approaches that are adapted to our task with a conversion, following [8]; Change-point detection approach **Funnel** [9], univariate time series anomaly detection approach **Donut** [20] and multivariate time series anomaly detection **USAD** [28]. We choose them because they are representative and widely used in their corresponding tasks. They use the whole KPI time series as the input and output a value for each time point, indicating the anomaly scores. We call these converted approaches Funnel*, Donut*, and USAD*.

C. Implementation & Parameters

We built a set of P and LS models of *Kontrast* and finally aggregated the models' results. According to our experiment, we set the #noise intensity categories K as 5, and the thresholds of classifying noise intensity are 0.005, 0.03, 0.1, 0.3, and 1. Experiment results are shown not to be sensitive to the choice of threshold values. Hence we do not present the experiment result due to page limit. For every noise intensity category, we generated 40,000 positive KPI time series segment pairs and 40,000 negative ones. For the LSTM module, we used Bidirectional LSTM (BiLSTM) [42], whose hidden size is 30. The fully connected (FC) module contains two layers to convert the output of LSTM to a fixed-length vector. Their hidden sizes are set to 30. The time intervals in data extraction process are set as $1T, 2T, 3T, 7T, 14T, 21T$. Adam optimizer is utilized during training with a constant learning rate of 0.001. The batch size is 10,000, and the number of epochs is 100. For all the models including baselines, if the loss value converges or the total calculation time exceeds 10 seconds per one KPI, the training or testing is aborted. In result aggregation, we set $\alpha = 2.5$ based on our experiment results.

We used the parameter from [8], [9], [20], [28], [41] and their open-source code for the compared approaches. To improve our reproducibility, we open-sourced our code on GitHub [14].

The experiments were conducted on Ubuntu 18.04.5 LTS with Intel(R) Xeon(R) CPU (2.60GHz), a 64-bit operating system, and an NVIDIA RTX 2080Ti GPU.

D. Results

To evaluate the performance of our model, we tested the approaches on two tasks: Anomalous KPI identification and erroneous software change identification. The following experiments were conducted to answer the research questions:

*1) **RQ1**: What is the performance of Kontrast?*

Anomalous KPI Identification As shown in Table II, in the first task, *Kontrast* outperforms baselines by a significant margin (0.013, 0.084, respectively), achieving 0.932 and 0.648 F1-score on two datasets. Due to the data characteristic in dataset $\mathcal{A}$ and the property of Holt-Winters algorithm, Gandalf fails to converge within the time limit on dataset $\mathcal{A}$. SCWarn and USAD* are multivariate approaches and use multiple KPI time series. Therefore, they have no result on this task.

Erroneous Software Change Identification Since we have no case information on dataset $\mathcal{A}$, we only performed the erroneous software change identification task on dataset $\mathcal{B}$. Based on the observations from our empirical study and the proposed approach, we judged the software change by the derived predictions of its related KPI time series. We used the 95$^{\text{th}}$ percentile of the distance predictions from the KPIs to distinguish the erroneous software changes. We see that *Kontrast* outperforms all other approaches. SCWarn fuses the rich information of multiple KPIs, achieving reasonable results. But it is based on prediction; therefore, it is influenced by the fluctuations in *ongoing period*. Lumos is based on comparison, but it ignores the sequential order of the data, thus performing poorer. DTW fails to recognize the difference between KPIs with diverse anomaly patterns. The data augmentation modules of TS-CP^2 are not designed for our task, so they performs poorly on our datasets. The experiments above suggest that *Kontrast* performs better than the compared approaches.

*2) **RQ2**: What is the time efficiency of Kontrast?*

Testing Time Time efficiency is a critical aspect for considerations in real-world deployment to identify erroneous software changes. As *Kontrast* is a generic model, it can batch process different KPI time series using one single model. This property highly improves the parallelism of the algorithm, making it able to process data in a relatively large batch, reducing per KPI time cost, as shown in Table III. Since TS-CP^2 utilizes contrastive learning, it is also efficient on this task. For the models that take multivariate KPI time series as input (USAD* and SCWarn), we calculated the average time cost of one KPI time series. As the results show, *Kontrast* owns a

Table II: Performance of the Models of Two Tasks.

Task		**Anomalous KPI**						**Erroneous Software Change**		
Category	***Approach***	***Dataset $\mathcal{A}$***			***Dataset $\mathcal{B}$***			***Dataset $\mathcal{B}$***		
		F1	***P***	***R***	***F1***	***P***	***R***	***F1***	***P***	***R***
End-to-end	SCWarn	-	-	-	-	-	-	0.944	0.956	0.933
	Gandalf	-	-	-	0.537	0.549	0.526	0.790	0.711	0.890
Contrastive	DTW	0.919	0.910	0.929	0.559	0.600	0.523	0.913	0.894	0.933
	Lumos	0.775	0.692	0.880	0.564	0.471	0.702	0.912	0.847	0.988
	Pearson	0.757	0.641	0.923	0.246	0.143	0.899	0.687	0.552	0.908
	TS-CP^2	0.790	0.792	0.788	0.279	0.195	0.494	0.740	0.633	0.890
Anomaly detection	Funnel*	0.818	0.871	0.772	0.343	0.594	0.241	0.744	0.693	0.804
	Donut*	0.874	0.927	0.826	0.483	0.626	0.393	0.874	0.927	0.826
	USAD*	-	-	-	-	-	-	0.775	0.723	0.834
Cross-dataset	*Kontrast* ($\mathcal{A}$)	-	-	-	0.593	0.585	0.601	-	-	-
	Kontrast ($\mathcal{B}$)	0.879	0.868	0.891	-	-	-	-	-	-
	Kontrast	**0.932**	**0.934**	**0.929**	**0.648**	**0.626**	**0.672**	**0.948**	**0.945**	**0.951**

Table III: Test and Training Time Cost.

Approach	**Dataset $\mathcal{A}$**	**Dataset $\mathcal{B}$**	
	Test (ms)[1]	***Test (ms)***[1]	***Training (min)***[2]
SCWarn	-	15	254
Gandalf	∞	5.8	>1,000
DTW	270	370	0
Lumos	34	24	0
Pearson	9.1	16.3	0
TS-CP^2	0.22	0.18	240
Funnel*	7,900	1,400	0
Donut*	240	130	>1,000
USAD*	-	2.7	**40.7**
Kontrast	**0.20**	**0.14**	45.0

[1] Time cost per KPI time series
[2] Time cost of the overall training process

millisecond-level speed in the testing phase of a KPI, which is 100 (e.g., SCWarn, Lumos) to 1,000 (e.g., Donut*, DTW) times faster than the state-of-the-art approaches excluding the contrastive learning-based ones.

Training Time The training time cost is also a critical evaluation metric for an approach because it determines the detection delay of the approaches that require online training. Since the training time of *Kontrast* highly depends on the hyper-parameters, we compare the results in a specific context (when running on dataset $\mathcal{B}$). For the models with no need for training, we regarded their training time as 0. For the models that need training for each KPI time series, we summed up the training time. For *Kontrast*, we summed up K training data generation time cost and K model training time cost.

Through the last column of Table III, we find that in the case where a considerable number (i.e., about 35,000) of KPI time series are to be processed, the training time of KPI-specific algorithms becomes unacceptable. On the contrary, the training time of *Kontrast* is only related to some hyper-parameters (the size of the generated dataset, K, ω, and training epochs), which means the training time cost of *Kontrast* is insensitive to the input size. TS-CP^2 possesses a rather complex model structure to extract detailed information from the KPI time series, which leads to low efficiency in training.

*3) **RQ3**: How does Kontrast adapt to different datasets?*

Adaptability To demonstrate the adaptability of *Kontrast*, we trained and tested it using different datasets. Using the approach from [43], we proved the dissimilarity of two datasets: a DTW-based algorithm measured the inter-dataset similarity between dataset $\mathcal{A}$ and $\mathcal{A}$ is 0.289, $\mathcal{B}$ and $\mathcal{B}$ is 0.274, while between $\mathcal{A}$ and $\mathcal{B}$ is 1.357, which indicated that dataset $\mathcal{A}$ and $\mathcal{B}$ were quite different on the shape of time series. Results can be found in the "Cross-dataset" category of Table II.

The results show that although *Kontrast* might not have learned the exact data patterns in the dataset, it achieves better than most compared models. Holt-Winters in Gandalf does not converge on dataset $\mathcal{A}$, therefore omitted. Furthermore, we have also conducted a five-fold cross validation on dataset $\mathcal{B}$. Experimental results are close to those in Table II. We do not list them here due to the page limit. These results demonstrate the adaptability of *Kontrast*, making it potentially capable of being deployed on unfamiliar platforms even without knowing their normal data patterns.

*4) **RQ4**: How do the main components contribute to Kontrast's performance?*

Noise Pattern Injection Fig. 8 shows the F1-scores of *Kontrast* without each of the noise pattern injection modules. The original one with all components performs best in all experiments on both datasets, while any deletion of the noise pattern injection module degrades the overall performance. The results show that our data augmentation modules are beneficial to our contrastive-learning-based model.

With respect to the competitiveness of our data augmentation modules, we also found our data augmentation performs better than the one in the latest work of self-supervised loss for data augmentation [44]. In fact, if we replace the data augmentation modules in *Kontrast* with the ones in [44], F1-scores of anomalous KPI identification decrease by 0.12 and 0.17 on datasets $\mathcal{A}$ and $\mathcal{B}$, and the one of erroneous software change identification decreases by 0.14, according to our experiment results. We conjecture this is because our data augmentation is closer to realistic software change cases, hence performs better than other approaches.

K - #Categories of the Noise Intensity Classifier To verify the effectiveness of the noise intensity classifier, we explore the relationship between K and the model performance. To explore the influence of parameters on P model and LS model, we used the raw prediction results of the models before aggregation (4) in this experiment and in Section IV-D5. We find

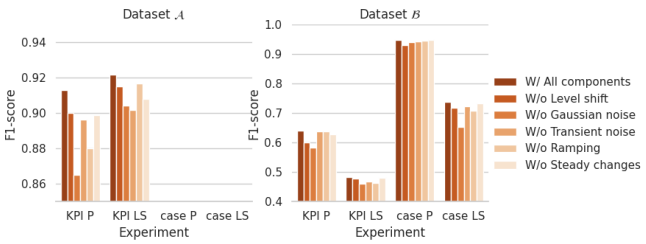
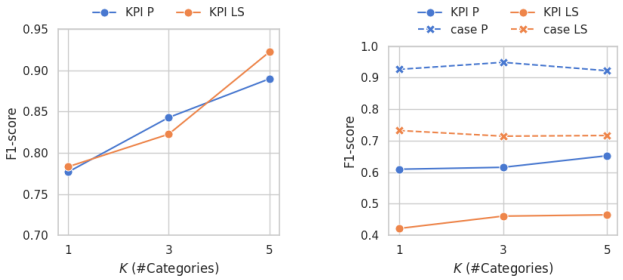


Fig. 8: The F1-scores of models without one of the components of data augmentation process. Note there is no software change case in dataset $\mathcal{A}$.



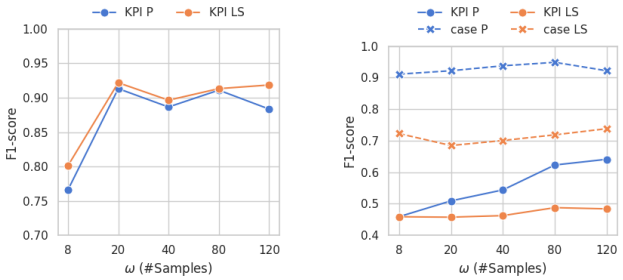
(a) Dataset $\mathcal{A}$. (b) Dataset $\mathcal{B}$.

Fig. 9: The best F1-scores on two datasets with different K.

that a greater K leads to a better performance with the trade-off of more data generating and training costs, according to the results in Fig. 9. Thus, we should reach an empirical balance between accuracy and time efficiency by trying multiple Ks on different datasets. We can also ensure that when $K = 1$ (processing all the KPI time series using one P model and one LS model), the performance on both datasets is the worst. This observation demonstrates the necessity of the noise intensity classification of *Kontrast*.

5) ***RQ5****: What is the influence of the related hyper-parameters?*

ω **- #Samples in the Inspecting Window** Fig. 10 shows the F1-scores with respect to the change of ω, where the larger the ω is, the better the model performs, roughly. However, a large ω introduces a significant detection time delay because we need to wait until the data of the ω^{th} time point has been obtained. In dataset $\mathcal{A}$, results are insensitive to ω when ω is big enough, showing the positive case generation of dataset $\mathcal{A}$ is robust enough for *Kontrast*.



(a) Dataset $\mathcal{A}$. (b) Dataset $\mathcal{B}$.

Fig. 10: The changing trend of F1-score along with the ω.

V. Real-World Applications and Limitations

To evaluate our approach in the practice conditions, we applied *Kontrast* in practical software change cases. We show several success stories and some limitations.

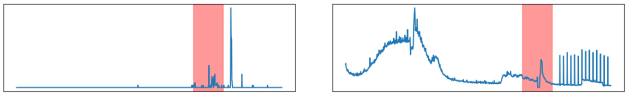
A. Success Stories

To evaluate and highlight the effectiveness of *Kontrast* in a realistic setup, we collect several real software change cases from the data center. Operators appreciate *Kontrast* in three-fold. 1) *Kontrast* is highly efficient in identifying erroneous software changes. The selected cases are tiny-scaled (#KPIs in one case is about 1,000) cases for manual analysis. *Kontrast* successfully identified all the erroneous software changes within one second, which is much faster than current practices. High efficiency helps the operators rapidly find the erroneous software changes to take immediate actions (e.g., roll-back) before a significant loss occurs. 2) Identification result is accurate even if only a small amount of historical data is available, thanks to the adaptability. Due to privilege issues in the data center, training data is not easy to obtain. Thus, model training is hard to carry out outside the data center, decreasing the efficiency of parameter fine-tuning. When processing the following software change cases, we found that even though we train *Kontrast* on our dataset $\mathcal{B}$, the overall performance does not degrade much and can still identify the anomalous KPIs and the erroneous software changes. The adaptability primarily improves data security, making it possible to fine-tune models without extensive sensitive data. 3) The accuracy is satisfying, reducing false alarm cases and missing rates. The analysis results conducted by *Kontrast* are closer to the human labels than those conducted by current practices, including FluxRank [12], k-sigma [13], and fixed threshold anomaly detection approaches. Applying *Kontrast*, the false alarm rate and the missing rate are lower, reducing operators' unnecessary efforts and improving the service's stability.

Some typical cases are listed as follows, and the figures of corresponding KPIs are shown in Fig. 11.

Case 1: On a specific application, after a regular function update, there was no anomaly except for a few transient spikes in some of the KPI time series. The spikes were probably caused by network service restart, which was self-healable and should not be considered abnormal. Compared approaches falsely alerted this pattern, causing a waste of operator's time, while *Kontrast* successfully recognized it as a false anomaly thanks to the Transient Noise module (Section III-B2). The module injects this sort of spikes into the negative cases, making the model be tolerant of this pattern.

Case 2: A newly deployed software change modified the log file saving function, causing a large increase in the output



(a) Case 1 (Negative). (b) Case 2 (Positive).

Fig. 11: KPIs from typical software change cases. Red shadow shows the *ongoing period*.

file size. The average transaction time had not significantly increased until the disk system was nearly full. *Kontrast* detected this incident after the software change by transaction time usage. It found that there were more frequent spikes in this KPI after deployment, while the compared approaches failed to detect it.

B. Limitations

There are some limitations with *Kontrast*. First, it is hard to detect silent incidents caused by erroneous software changes. Another limitation is that *Kontrast* currently processes each KPI time series separately, ignoring the inter-dependency [45] in the multivariate data. Due to the architecture of our model, multivariate and adaptability are exclusive. We leave these to our future improvements to tackle these issues.

VI. Related Works

A. Software Change

Software change has been a popular research domain in academia and industry for several years [46]–[48]. To improve the reliability of software changes, erroneous software change identification is critical. Existing erroneous software change identification approaches [8]–[10], [40] majorly regard this problem as an anomaly detection task, utilizing anomaly detection (or change point detection [49]–[51]) algorithms to apply to this problem directly. For instance, multimodal LSTM in [8], improved Singular Spectrum Transform (iSST) in [9], Holt-Winters in [10] and A/B Test in [40]. However, these approaches are not efficient and effective enough in our scenario. The computation cost of Funnel [9] is too high, and the accuracy of Lumos [40] is far from satisfactory. Other approaches are KPI-specific, requiring training models for every KPI, leading to low efficiency and high overhead, proved in Section IV.

B. Anomaly Detection

In the literature, anomaly detection approaches are also applied in identifying erroneous software changes [8], [9]. Donut [20] first used Variational Autoencoder (VAE) [52] to reconstruct the KPI time series, detecting anomalies in seasonal KPI time series. OmniAnomaly [53] used Stochastic Recurrent Neural Network (Stochastic RNN) to handle multivariate time series, performing anomaly detection. FluxRank [12] used KDE [16] to check if an anomaly exists. USAD [28], a multivariate anomaly detection approach, applied adversely trained autoencoders to train the network rapidly in an unsupervised way. However, these anomaly detection approaches are not sensitive to the vital information and properties of software changes (e.g., KPI fluctuations in *ongoing period*, comparison between pre-change and post-change period). Another fatal drawback of them is that they need to train a specific model for each KPI time series or software system, thus requiring massive calculation resources and leading to high overhead. Therefore, anomaly detection-based approaches are not suitable for directly applying to erroneous software change identification. *Kontrast* focus on the critical tenet of software changes, ignoring the unrelated parts of the KPI time series, thus achieving high efficiency and superior performance.

C. Contrastive Learning & Self-Supervised Learning

Contrastive learning is prevalent in CV and NLP domains [54], [55]. It helps models to better distinguish different data samples by learning a representation, maximizing similarity over samples from the same category, and dissimilarity over samples from different categories [56]. Contrastive loss [27] and Triplet loss [57] are two major loss functions used to train contrastive learning models. Siamese network [26] is a classical model in contrastive learning. Its twin-tower architecture is capable of handling homogenous or heterogenous data [58], [59].

Contrastive learning requires a fully labeled dataset, while the labels are costly in our scenario. Fortunately, data augmentation based on self-supervised learning is a valuable tool for generating training data [60], [61]. SimCLRv2 [54] applies it on image (cropping, resizing, etc.), and ConSERT [55] applies on text (cutoff, shuffing, etc.). In this way, we can generate pseudo-labeled pairs from unlabeled data for training contrastive learning models.

Applying self-supervised learning approaches on time series data is becoming increasingly popular [41], [44], [62]–[65]. These works typically apply data augmentation technique to generate time series segment pairs to train the model for various downstream tasks, i.e., time series classification in SimCLR-TS [63] and TimeCLR [65], representation learning in TS-TCC [64], change point detection in TS-CP^2 [41], and anomaly detection in TimeAutoAD [44]. Though their data augmentation approaches are effective in some datasets, they are not designed for erroneous change identification tasks, omitting the periodicity and local stability properties introduced in Section III-A that normal KPIs follow.

VII. Conclusion

To better understand software changes, we conducted a comprehensive empirical study from a global data center and revealed several key observations that motivate us to propose *Kontrast*. *Kontrast* is a novel self-supervised contrastive learning approach aiming to identify erroneous software changes rapidly and accurately. Its comparison-based architecture allows operators to process time series from different KPIs simultaneously in a batch, thus dramatically increasing efficiency. *Kontrast* possesses cross-dataset adaptability, making training and testing on different datasets possible. An extensive study including various erroneous software changes verified the effectiveness and efficiency of *Kontrast*, outperforming all the compared approaches.

Acknowledgment

This work is supported by the National Key R&D Program of China 2019YFB1802504, and the State Key Program of National Natural Science of China under Grant 62072264 and 61902200.

REFERENCES

[1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site reliability engineering: How Google runs production systems*. " O'Reilly Media, Inc.", 2016.

[2] "2021 Facebook Outage," https://en.wikipedia.org/wiki/2021_Facebook_outage, [Online; accessed 13-May-2022].

[3] S. Mehta, R. Bhagwan, R. Kumar, C. Bansal, C. Maddila, B. Ashok, S. Asthana, C. Bird, and A. Kumar, "Rex: Preventing bugs and misconfiguration in large services using correlated change analysis," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 435–448.

[4] S. Lehnert, "A review of software change impact analysis," 2011.

[5] E. Zhai, A. Chen, R. Piskac, M. Balakrishnan, B. Tian, B. Song, and H. Zhang, "Check before you change: Preventing correlated failures in service updates," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 575–589.

[6] G. Schermann, D. Schöni, P. Leitner, and H. C. Gall, "Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies," in *Proceedings of the 17th International Middleware Conference*, 2016, pp. 1–14.

[7] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor *et al.*, "The top 10 adages in continuous deployment," *IEEE Software*, vol. 34, no. 3, pp. 86–95, 2017.

[8] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, "Identifying bad software changes via multimodal anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 527–539.

[9] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, Z. Zang, X. Jing, and M. Feng, "Funnel: Assessing software changes in web-based services," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 34–48, 2016.

[10] Z. Li, Q. Cheng, K. Hsieh, Y. Dang, P. Huang, P. Singh, X. Yang, Q. Lin, Y. Wu, S. Levy *et al.*, "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)*, 2020, pp. 389–402.

[11] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 internet measurement conference*, 2015, pp. 211–224.

[12] P. Liu, Y. Chen, X. Nie, J. Zhu, S. Zhang, K. Sui, M. Zhang, and D. Pei, "Fluxrank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 35–46.

[13] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994.

[14] "*Kontrast*," https://github.com/WXR1998/kontrast, [Online; accessed 25-Aug-2022].

[15] "Promtheus," https://prometheus.io/, [Online; accessed 08-May-2022].

[16] R. A. Davis, K.-S. Lii, and D. N. Politis, "Remarks on some nonparametric estimates of a density function," in *Selected Works of Murray Rosenblatt*. Springer, 2011, pp. 95–100.

[17] A. F. Siegel, "Testing for periodicity in a time series," *Journal of the American Statistical Association*, vol. 75, no. 370, pp. 345–348, 1980.

[18] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Periodicity detection in time series databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 875–887, 2005.

[19] T. M. Chilimbi and V. Ganapathy, "Heapmd: Identifying heap-based bugs using anomaly detection," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 219–228, 2006.

[20] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 187–196.

[21] W. Chen, H. Xu, Z. Li, D. Pei, J. Chen, H. Qiao, Y. Feng, and Z. Wang, "Unsupervised anomaly detection for intricate kpis via adversarial training of vae," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1891–1899.

[22] Y. Su, Y. Zhao, W. Xia, R. Liu, J. Bu, J. Zhu, Y. Cao, H. Li, C. Niu, Y. Zhang *et al.*, "Coflux: Robustly correlating kpis by fluctuations for service troubleshooting," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[23] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 4037–4058, 2020.

[24] C. Wu, N. Zhao, L. Wang, X. Yang, S. Li, M. Zhang, X. Jin, X. Wen, X. Nie, W. Zhang *et al.*, "Identifying root-cause metrics for incident diagnosis in online service systems," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 91–102.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[26] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[27] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[28] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: UnSupervised Anomaly Detection on Multivariate Time Series," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Virtual Event CA USA: ACM, Aug. 2020, pp. 3395–3404.

[29] "AIOps2018 Challenge," https://github.com/NetManAIOps/KPI-Anomaly-Detection, [Online; accessed 08-May-2022].

[30] J. Qiu, Q. Du, and C. Qian, "Kpi-tsad: A time-series anomaly detector for kpi monitoring in cloud applications," *Symmetry*, vol. 11, no. 11, p. 1350, 2019.

[31] J. Qian, F. Liu, D. Li, X. Jin, and F. Li, "Large-scale kpi anomaly detection based on ensemble learning and clustering," *Journal of Cybersecurity*, vol. 2, no. 4, p. 157, 2020.

[32] Y. Xia, J. Lu, Y. Li, B. Zhang, H. Li, F. Xie, S. Liu, and C. Xu, "Anomaly detection and processing in artificial intelligence for it operations of power system," in *2019 IEEE 8th International Conference on Advanced Power System Automation and Protection (APAP)*, 2019, pp. 1099–1104.

[33] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.

[34] J. Paul Martin, A. Kandasamy, and K. Chandrasekaran, "Crew: Cost and reliability aware eagle-whale optimiser for service placement in fog," *Software: Practice and Experience*, vol. 50, no. 12, pp. 2337–2360, 2020.

[35] L. Larsson, W. Tärneberg, C. Klein, M. Kihl, and E. Elmroth, "Adaptive and application-agnostic caching in service meshes for resilient cloud applications," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 176–180.

[36] "Hipster Shop," https://github.com/lightstep/hipster-shop, [Online; accessed 08-May-2022].

[37] "Kubernetes," https://kubernetes.io/, [Online; accessed 08-May-2022].

[38] C. Chatfield, "The holt-winters forecasting procedure," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 27, no. 3, pp. 264–279, 1978.

[39] R. Bellman and R. Kalaba, "On adaptive control processes," *IRE Transactions on Automatic Control*, vol. 4, no. 2, pp. 1–9, 1959.

[40] J. Pool, E. Beyrami, V. Gopal, A. Aazami, J. Gupchup, J. Rowland, B. Li, P. Kanani, R. Cutler, and J. Gehrke, "Lumos: A library for diagnosing metric regressions in web-scale applications," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2562–2570.

[41] S. Deldari, D. V. Smith, H. Xue, and F. D. Salim, "Time series change point detection with self-supervised contrastive predictive coding," in *Proceedings of the Web Conference 2021*, 2021, pp. 3124–3135.

[42] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[43] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Transfer learning for time series classification," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1367–1376.

[44] Y. Jiao, K. Yang, D. Song, and D. Tao, "Timeautoad: Autonomous anomaly detection with self-supervised contrastive loss for multivariate time series," *IEEE Transactions on Network Science and Engineering*, 2022.

[45] Z. Li, Y. Zhao, R. Liu, and D. Pei, "Robust and rapid clustering of kpis for large-scale anomaly detection," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[46] X. Zhang, C. Du, Y. Li, Y. Xu, H. Zhang, S. Qin, Z. Li, Q. Lin, Y. Dang, A. Zhou, S. Rajmohan, and D. Zhang, "HALO: Hierarchy-aware fault localization for cloud systems," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, pp. 3948–3958.

[47] Y. Xu, X. Zhang, C. Luo, S. Qin, R. Pandey, C. Du, Q. Lin, Y. Dang, and A. Zhou, "CARE: Infusing causal aware thinking to root cause analysis in cloud system," in *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems*. ACM, pp. 1–3.

[48] A. Mahimkar, C. E. de Andrade, R. Sinha, and G. Rana, "A composition framework for change management," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. ACM, pp. 788–806.

[49] A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks," p. 12.

[50] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert, "Rapid detection of maintenance induced changes in service performance," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11*. ACM Press, pp. 1–12.

[51] A. Mahimkar, Z. Ge, J. Yates, C. Hristov, V. Cordaro, S. Smith, J. Xu, and M. Stockert, "Robust assessment of changes in cellular networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, pp. 175–186.

[52] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[53] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2828–2837.

[54] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, "Big self-supervised models are strong semi-supervised learners," *Advances in neural information processing systems*, vol. 33, pp. 22 243–22 255, 2020.

[55] Y. Yan, R. Li, S. Wang, F. Zhang, W. Wu, and W. Xu, "Consert: A contrastive framework for self-supervised sentence representation transfer," *arXiv preprint arXiv:2105.11741*, 2021.

[56] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, "What should not be contrastive in contrastive learning," *arXiv preprint arXiv:2008.05659*, 2020.

[57] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification." *Journal of machine learning research*, vol. 10, no. 2, 2009.

[58] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[59] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8971–8980.

[60] T. Yao, X. Yi, D. Z. Cheng, F. Yu, T. Chen, A. Menon, L. Hong, E. H. Chi, S. Tjoa, J. Kang *et al.*, "Self-supervised learning for large-scale item recommendations," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4321–4330.

[61] J. Ma, C. Zhou, H. Yang, P. Cui, X. Wang, and W. Zhu, "Disentangled self-supervision in sequential recommenders," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 483–491.

[62] K. Wickstrøm, M. Kampffmeyer, K. Ø. Mikalsen, and R. Jenssen, "Mixing up contrastive learning: Self-supervised representation learning for time series," *Pattern Recognition Letters*, vol. 155, pp. 54–61, 2022.

[63] J. Pöppelbaum, G. S. Chadha, and A. Schwung, "Contrastive learning based self-supervised time-series analysis," *Applied Soft Computing*, p. 108397, 2022.

[64] E. Eldele, M. Ragab, Z. Chen, M. Wu, C. K. Kwoh, X. Li, and C. Guan, "Time-series representation learning via temporal and contextual contrasting," *arXiv preprint arXiv:2106.14112*, 2021.

[65] X. Yang, Z. Zhang, and R. Cui, "Timeclr: A self-supervised contrastive learning framework for univariate time series representation," *Knowledge-Based Systems*, vol. 245, p. 108606, 2022.