Unsupervised Anomaly Detection on Microservice Traces through Graph VAE

Zhe Xie
Tsinghua University; BNRist
Beijing, China
xiez22@mails.tsinghua.edu.cn

Haowen Xu
Tsinghua University
Beijing, China
xhw15@mails.tsinghua.edu.cn

Wenxiao Chen
Tsinghua University
Beijing, China
chen-wx17@mails.tsinghua.edu.cn

Wanxue Li
eBay
Shanghai, China
wanxuli@ebay.com

Huai Jiang
eBay
Shanghai, China
huajiang@ebay.com

Liangfei Su
eBay
Shanghai, China
suliangfei@gmail.com

Hanzhang Wang
eBay
San Jose, USA
wanghanz@umich.edu

Dan Pei[*]
Tsinghua University; BNRist
Beijing, China
peidan@tsinghua.edu.cn

ABSTRACT

The microservice architecture is widely employed in large Internet systems. For each user request, a few of the microservices are called, and a trace is formed to record the tree-like call dependencies among microservices and the time consumption at each call node. Traces are useful in diagnosing system failures, but their complex structures make it difficult to model their patterns and detect their anomalies. In this paper, we propose a novel dual-variable graph variational autoencoder (VAE) for unsupervised anomaly detection on microservice traces. To reconstruct the time consumption of nodes, we propose a novel dispatching layer. We find that the inversion of negative log-likelihood (NLL) appears for some anomalous samples, which makes the anomaly score infeasible for anomaly detection. To address this, we point out that the NLL can be decomposed into KL-divergence and data entropy, whereas lower-dimensional anomalies can introduce an entropy gap with normal inputs. We propose three techniques to mitigate this entropy gap for trace anomaly detection: Bernoulli & Categorical Scaling, Node Count Normalization, and Gaussian Std-Limit. On five trace datasets from a top Internet company, our proposed TraceVAE achieves excellent F-scores.

CCS CONCEPTS

• **Networks** → **Network services**; • **Computing methodologies** → *Artificial intelligence*.

[*] Corresponding author.

WWW '23, April 30-May 4, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583215

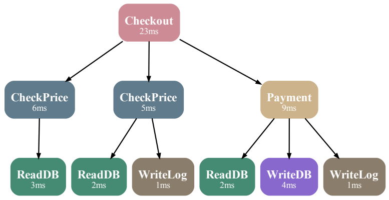


Figure 1: An Example of Microservice Trace

KEYWORDS

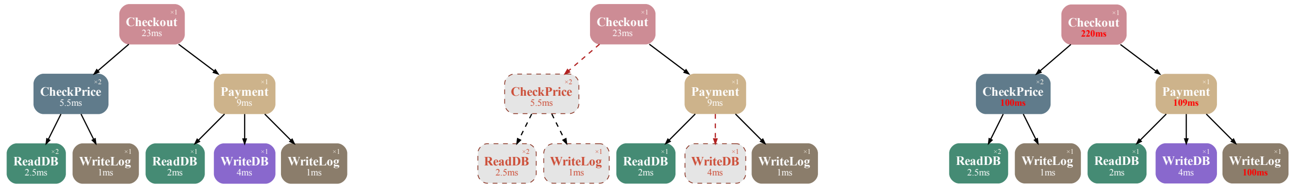
microservice trace, anomaly detection, graph vae, variational autoencoder, deep learning

ACM Reference Format:
Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. 2023. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583215

1 INTRODUCTION

To provide web services to global Internet users, many large Internet companies divide their backend services into thousands of *microservices*. Each microservice provides several APIs. For each user request, the APIs of some of the microservices are called, and a *trace* (see Figure 1) is formed to record the tree-like call dependencies among the called APIs as well as the invocation features (*e.g.* time consumption and return code). As with the collection of metrics and log data, traces can be collected through common monitoring tools such as OpenTelemetry[1]. Since traces record the information about the internal invocations in the microservice system, traces are helpful for operators in failure diagnosis. Therefore, detecting anomalies on microservice traces is needed to maintain the quality of service. Unsupervised algorithms are demanded due to a large number of traces and high labor costs of labeling. Previous

[1] https://opentelemetry.io/



(a) Merged Trace **(b) Structure Anomaly** **(c) Time Anomaly**

Figure 2: Merged Trace and Anomalies

work on trace anomaly detection uses handcrafted methods [3, 10, 22], or more powerful deep learning-based methods [19, 23], but most existing deep learning-based methods require vectorizing the traces. The vectorization might potentially cause loss of important structure information, degrading anomaly detection performance. Although some studies [32, 33] proposed to learn the structural patterns of traces with GNN or Tree-LSTM, they require supervised training or additional data for detection, which still do not well address the problems.

Therefore, it is still a challenge to accurately model the structural features of traces in an unsupervised way and detect their anomalies. The variational autoencoder (VAE) has become a popular choice for unsupervised anomaly detection on web services since [31]. TraceAnomaly [19] and CRISP [35] have successfully applied VAE in trace anomaly detection, but the graph data are not considered. Most VAE-based models on graph data follow the architecture of variational graph autoencoder (VGAE) [14], in which the latent variable $\mathbf{Z}$ is a matrix. Each row of $\mathbf{Z}$ in VGAE encodes one node, rather than $\mathbf{Z}$ as a whole to encode the entire graph. Meanwhile, only a few graph VAEs [16, 20, 24] encode the entire graphs, but they use fully-connected layers to decode both the adjacency matrix $\mathbf{A}$ and the node features $\mathbf{X}$. We conjecture that the use of fully-connected layers may lead to poor generalization performance for node features, resulting in inaccurate trace reconstruction. In this work, we introduce a novel *dispatching layer* to mitigate this performance degradation in trace anomaly detection, which uses location-independent global encoding to help the decoder learn the time-consuming association between nodes.

The negative log-likelihood (NLL) has been widely used as anomaly scores [19]. However, for some of the anomalous inputs, we find that their NLLs are smaller than normal inputs, even if the VAE can correctly reconstruct their normal patterns. This phenomenon of *NLL inversion* makes NLLs no longer reliable as anomaly scores for trace anomaly detection. In [21], it has been found that out-of-distribution (OOD) inputs may exhibit lower NLLs than training distribution inputs. To address this, some works proposed solutions [29] for mitigating the impact of OOD samples on classification tasks. [12, 17] proposed to detect these OOD samples with specially designed classifiers. However, existing works did not give an explanation for such phenomena in anomaly detection nor proposed a solution. In this paper, we find that the NLL can be decomposed into a KL-divergence term plus an entropy term, where the entropy can be significantly smaller for lower-dimensional data than for higher-dimensional data. From the perspective of trace anomaly detection, we further explain why this might cause the problem and result in performance degradation. To mitigate the impact of this, we propose several techniques to reduce the entropy gap in trace anomaly detection: *Bernoulli & Categorical Scaling*, *Node Count Normalization* and *Gaussian Std-Limit*.

This paper proposes a novel model *TraceVAE* to detect trace anomalies. TraceVAE models the trace features with a dual-variable graph VAE model (Section 4) and detects anomalies with NLL, to which the techniques to reduce the entropy gap are applied (Section 5). We conduct experiments to investigate the performance of TraceVAE on five datasets collected from *real* microservice system in an International e-commerce company. The experimental results show that TraceVAE achieves significant advantages compared with the baseline methods and the techniques to reduce the entropy gaps are effective. In summary, our contributions are:

- We propose *TraceVAE*, a novel dual-variable graph VAE. TraceVAE encodes the structural features and node features into $\mathbf{z}$ and $\mathbf{z}_2$ respectively. To achieve better generalization, we propose a novel *dispatching layer* which treats $\mathbf{z}_2$ as the graph-level context and expands it into node-level features.
- We show that lower-dimensional out-of-distribution inputs can have lower NLLs than higher-dimensional training distribution inputs because of the gap introduced by the data entropy. The entropy is an *intrinsic property* of data, thus increasing the "*apparent dimensionality*" cannot fix the problem.
- To reduce the entropy gap, we propose three effective techniques: Bernoulli & Categorical Scaling, Node Count Normalization, and Gaussian Std-Limit.
- On five trace datasets collected from a top International e-commerce company, our proposed TraceVAE achieves excellent F-scores.

2 PROBLEM STATEMENT

Trace. The nodes in a trace represent calls to APIs, while the edges represent their call dependencies. Figure 1 shows an exemplary trace, where the bold texts (and the colors) label the microservices at each node, and the texts below with smaller font indicate the time consumptions. The time consumption of a node includes the time consumption of all its children plus the processing time of this node itself. Note that the same microservice (*e.g., CheckPrice* in Figure 1) can be called for multiple times in one single trace. We normalize the time consumptions of the nodes *w.r.t.* each API separately. We denote a trace as $G = (\mathbf{A}, \mathbf{X}, \mathbf{Y})$. $\mathbf{A}$ is the $N \times N$ adjacency matrix of the trace, where $A_{ij} = 1$ if i is the parent of node j. $\mathbf{X}$ is an $N \times 1$ matrix, where each row is the time consumption of the corresponding node. $\mathbf{Y}$ is also an $N \times 1$ matrix, where each row is the API ID.

Trace Merging. Normally, the graph autoencoders require expensive *graph matching* [24], since the nodes of an ordinary graph have equal roles and thus are permutation invariant. However, traces are trees. The root node of a tree and the parent nodes of all subtrees have different roles with their children, thus the nodes are not permutation invariant. We thus need to assign deterministic node orders. To avoid the inaccurate orders caused by parallel calls (which may have different start times), we merge the nodes with the same caller APIs. We then sort the nodes with BFS search according to their IDs assigned to each API. Since the nodes are sorted in BFS orders, $\mathbf{A}$ is an upper triangular matrix, where $A_{ij} \equiv 0$ for all $i \geq j$.

Trace Anomalies. There are two types of trace anomalies: *structure anomalies*, and *time-consumption anomalies* (*time anomalies* for short). Structure anomalies are the anomalies where some nodes should be but are not called, where the missing nodes are annotated with dashed lines in Figure 2b. Time anomalies are the anomalies where the time consumptions of some nodes (and their parent nodes) increase significantly, where the increased time consumptions are highlighted with red color in Figure 2c. The detection of structure anomalies requires to *exactly match* the structure of the whole trace, where the algorithm should be accurate enough to avoid interpreting normal traces as anomalies and also sensitive enough to find out the anomalies with just one node missing. This requires the model to encode the trace structure very precisely, having more emphasis on the *whole structure.* On the other hand, the detection of time anomalies requires neighborhood generalization since the time consumption at each node should mostly be determined by its surrounding nodes. This requires the model to emphasize *neighborhood nodes* when encoding the time consumptions. These different properties of the two types of anomalies suggest that we should encode the structure and time consumption by separate networks, which inspires us to propose a dual-variable graph VAE in this paper.

In summary, trace anomaly detection is formulated as follows. The goal of this paper is to obtain an anomaly score $s(G)$ for each given trace G, where large $s(G)$ should indicate that G is likely to be an anomaly trace.

3 PREREQUISITES

3.1 Variational Autoencoder

Denote the observed data distribution as $\mathbf{x}$. Variational autoencoder (VAE) [13] models the distribution of $\mathbf{x}$ by $p_\theta(\mathbf{x}) = \mathbb{E}_{p_\lambda(\mathbf{z})}[p_\theta(\mathbf{x}|\mathbf{z})]$, where $\mathbf{z}$ is the latent variable that encodes $\mathbf{x}$, whose prior is $p_\lambda(\mathbf{z})$. It can be trained by maximizing the variational lower-bound (ELBO):

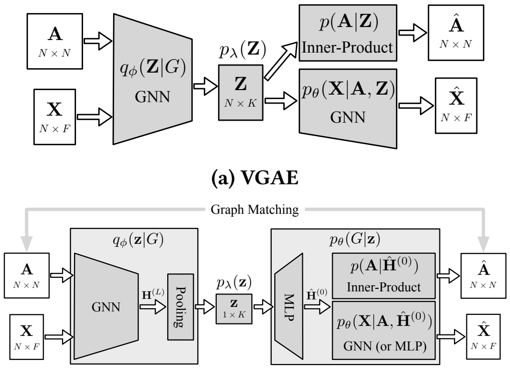
$$\begin{aligned} \log p_\theta(\mathbf{x}) &\geq \log p_\theta(\mathbf{x}) - \mathrm{KLD}[q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\lambda(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \end{aligned} \tag{1}$$

The variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is a approximation of the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, which encodes $\mathbf{x}$ into $\mathbf{z}$, while $p_\theta(\mathbf{x}|\mathbf{z})$ decodes $\mathbf{x}$ back from $\mathbf{z}$.

3.2 Graph Attention Networks

Graph neural networks [11, 15, 34] are designed to process graph data $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A}$ is the $N \times N$ adjacency matrix, and $\mathbf{X}$ is an $N \times F$ matrix, in which N denotes the number of nodes, F denotes the node features size, and the i-th row x_i is the feature vector of node i. We use the graph attention networks (GAT) [28] in this paper. GAT leverages the attention mechanism to specify different weights for different neighbor nodes in aggregation, which is suitable for learning the different invocation patterns (*e.g.* synchronous or asynchronous) in traces.

3.3 VGAE and Graph VAE



(a) VGAE

(b) Graph VAE

Figure 3: VGAE and Graph VAE

Existing VAEs on graph data can be divided into two categories: the *variational graph autoencoders (VGAE)* [14] which dominates most of the previous work, and the seldomly used *graph variational autoencoders (graph VAE)* [16, 20, 24]. Figure 3 shows the architectures of VGAE and graph VAE.

The main difference between VGAE and graph VAE lies in the latent variable $\mathbf{z}$. For a graph G with N nodes, the $\mathbf{Z}$ of VGAE is an $N \times K$ matrix, where each row encodes one node. Since the rows of $\mathbf{Z}$ are (conditionally) independent of each other, in both the posterior $q_\phi(\mathbf{Z}|G)$ and the prior $p_\lambda(\mathbf{Z})$, VGAE might be seen as an autoencoder for graph nodes rather than for the graph itself. It is impossible for the missing nodes to be included in the reconstructed graph when a structure anomaly happens with VGAE. As a result, VGAE is not suitable for trace anomaly detection.

On the contrary, the latent $\mathbf{z}$ of graph VAE is a vector of length K, encoding the whole graph. The node features from graph neural networks in $q_\phi(\mathbf{z}|G)$ are passed through a graph pooling layer to produce the graph features, which are used in sampling the latent $\mathbf{z}$. In existing graph VAEs, a fully-connected network is usually employed to decode the graph-level $\mathbf{z}$ back into node-level features, and such node-level features are then used to further produce the reconstructed structures. However, this design may make them more sensitive to small structure anomalies, but lack the generalization ability required for time anomaly detection. We will present how to improve this structure in Section 4.

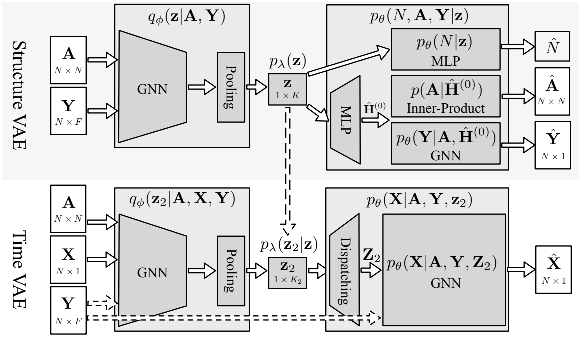

Structure VAE
A
$N \times N$
Y
$N \times F$
$q_\phi(\mathbf{z}|\mathbf{A}, \mathbf{Y})$
GNN
Pooling
$p_\lambda(\mathbf{z})$
$\mathbf{z}$
$1 \times K$
$p_\theta(N, \mathbf{A}, \mathbf{Y}|\mathbf{z})$
$p_\theta(N|\mathbf{z})$
MLP
MLP
$\hat{\mathbf{H}}^{(0)}$
$p(\mathbf{A}|\hat{\mathbf{H}}^{(0)})$
Inner-Product
$p_\theta(\mathbf{Y}|\mathbf{A}, \hat{\mathbf{H}}^{(0)})$
GNN
$\hat{N}$
$\hat{\mathbf{A}}$
$N \times N$
$\hat{\mathbf{Y}}$
$N \times 1$
Time VAE
A
$N \times N$
X
$N \times 1$
Y
$N \times F$
$q_\phi(\mathbf{z}_2|\mathbf{A}, \mathbf{X}, \mathbf{Y})$
GNN
Pooling
$p_\lambda(\mathbf{z}_2|\mathbf{z})$
$\mathbf{z}_2$
$1 \times K_2$
$p_\theta(\mathbf{X}|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2)$
Dispatching
$\mathbf{Z}_2$
$p_\theta(\mathbf{X}|\mathbf{A}, \mathbf{Y}, \mathbf{Z}_2)$
GNN
$\hat{\mathbf{X}}$
$N \times 1$

Figure 4: Network Architecture of TraceVAE

4 NETWORK ARCHITECTURE

As the introduction describes, graphs enable precise modeling of complex structures and time relationships in traces. Graph VAE [16, 20, 24] has demonstrated its great generalization ability in graph modeling. Therefore, we propose TraceVAE based on an improved graph VAE model to take advantage of the graph features in traces.

Overview. We will introduce the overall architecture of our *TraceVAE*, shown as Figure 4. The goal of VAE is to learn the normal distributions for traces. Given a trace G, TraceVAE outputs its normal distribution $p_{model}(G)$ through reconstruction. TraceVAE consists of two parts: structure VAE and time VAE, which reconstruct structural features and time features of traces respectively. To precisely encode the whole structure and to satisfy the neighborhood generalization property of time consumptions, we encode the structure and time information in two separated VAEs, resulting in the structural latent variable $\mathbf{z}$ and the time latent variable $\mathbf{z}_2$.

Structure VAE. The structure information of the microservices is captured in the structure VAE. In graph VAEs, $\mathbf{A}$ is encoded by a fixed-length $\mathbf{z}$ and decoded with a fully-connected network, thus $\mathbf{A}$ should have a fixed size of $N_{max} \times N_{max}$, where N_{max} is the maximum number of nodes per trace in the dataset ($N_{max} = 32$ in our paper). To know the exact sizes of traces, we introduce an auxiliary variable N. We use *average pooling* in this paper to encode the graph-level latent variables. Unlike ordinary graph VAEs, TraceVAE does not use *graph matching* since the trace nodes have been sorted and have deterministic orders (see Section 2).

Time VAE. The time VAE captures the time information of nodes and their relationship. In time VAE, both structural and time features are used as inputs, where the gradients to the structural features are blocked (denoted as dashed arrows in Figure 4) to avoid them from being populated by time information.

Dispatching Layer. As mentioned earlier, most of the existing graph VAEs use fully-connected networks to directly map the graph-level $\mathbf{z}_2$ into node-level features. However, the detection of time anomalies requires neighborhood generalization, which is difficult to achieve through such direct mapping. Therefore, we conjecture that the fully-connected network is not suitable for encoding the time consumption. We thus propose the novel **dispatching layer**, which treats $\mathbf{z}_2$ as a **shared graph-level context** for all the nodes.

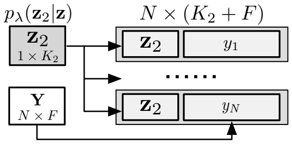


Figure 5: Architecture of the Dispatching Layer

Figure 5 demonstrates the dispatching layer, where $\mathbf{z}_2$ is concatenated with the node embeddings y_i and decoded with another GNN. The dispatching layer can be formulated as $\mathbf{Z}_{2,i} = [\mathbf{z}_2 \parallel y_i]$. In this way, the dispatching layer uses $\mathbf{z}_2$ as shared information in the node latency embeddings $\mathbf{Z}_2$, mitigating the decreasing generalizability of latency features due to the structural information encoded in $\mathbf{z}_2$.

Summary. Given that $G = (\mathbf{A}, \mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ is the time consumption matrix and $\mathbf{Y}$ is the microservice ID matrix, our proposed *TraceVAE* can be formulated as:

$$
\begin{aligned}
p_{\theta,\lambda}(G, N, \mathbf{z}, \mathbf{z}_2) &= p_{\theta,\lambda}(N, \mathbf{A}, \mathbf{X}, \mathbf{Y}, \mathbf{z}, \mathbf{z}_2) \\
&= \left[p_\theta(N, \mathbf{A}, \mathbf{Y}|\mathbf{z})\, p_\lambda(\mathbf{z})\right] \cdot \left[p_\theta(\mathbf{X}|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2)\, p_\lambda(\mathbf{z}_2|\mathbf{z})\right] \\
p_\theta(N, \mathbf{A}, \mathbf{Y}|\mathbf{z}) &= p_\theta(N|\mathbf{z}) \cdot \textstyle\prod_{ij} p_\theta(A_{ij}|\mathbf{z}) \cdot \prod_i p_\theta(y_i|\mathbf{z}) \\
p_\theta(\mathbf{X}|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2) &= \textstyle\prod_i p_\theta(x_i|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2) \\
q_\phi(\mathbf{z}, \mathbf{z}_2|G, N) &= q_\phi(\mathbf{z}|\mathbf{A}, \mathbf{Y}) \cdot q_\phi(\mathbf{z}_2|\mathbf{A}, \mathbf{X}, \mathbf{Y})
\end{aligned}
\tag{2}
$$

where $q_\phi(\mathbf{z}|\mathbf{A}, \mathbf{Y})$ and $p_\theta(N, \mathbf{A}, \mathbf{Y}|\mathbf{z})$ are the encoder and decoder of the structure VAE, while $q_\phi(\mathbf{z}_2|\mathbf{A}, \mathbf{X}, \mathbf{Y})$ and $p_\theta(\mathbf{X}|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2)$ are those of the time VAE. N and y_i are categorical distributions, A_{ij} are Bernoulli distributions, and x_i are Gaussian distributions. $p_\lambda(\mathbf{z})$ is a learnable RealNVP prior [30], and $p_\lambda(\mathbf{z}_2|\mathbf{z})$ is a learnable Gaussian prior conditioned on $\mathbf{z}$.

In summary, we propose a dual-variable graph VAE, the *TraceVAE*, to encode the structure and time consumptions of traces in separated VAEs. The structure VAE uses a fully-connected network to reconstruct the structure precisely, whereas the time VAE uses the novel *dispatching layer* to benefit the neighborhood generalization on node times.

5 ANOMALY SCORE

5.1 NLL and the Entropy Gap

5.1.1 NLL of Traces. The negative log-likelihood (NLL) has been widely used as the anomaly score in many existing works [8, 12]. For a given trace G, the NLL is formulated as $-\log p_{\text{model}}(G)$, which indicates how unlikely the given trace G is to follow the model distribution. The NLL for G can be approximated by Monte Carlo integration [4]:

$$
\begin{aligned}
NLL_G &= -\log p_{\text{model}}(G) \\
&= -\log \mathbb{E}_{q_\phi(\mathbf{z},\mathbf{z}_2|G,N)} \left[\frac{p_{\theta,\lambda}(G, N, \mathbf{z}, \mathbf{z}_2)}{q_\phi(\mathbf{z}, \mathbf{z}_2|G, N)} \right] \\
&\approx -\log \left[\frac{1}{L} \sum_{l=1}^{L} \frac{p_{\theta,\lambda}(N, \mathbf{A}, \mathbf{X}, \mathbf{Y}, \mathbf{z}^{(l)}, \mathbf{z}_2^{(l)})}{q_\phi(\mathbf{z}^{(l)}, \mathbf{z}_2^{(l)}|G, N)} \right]
\end{aligned}
\tag{3}
$$

where $\mathbf{z}^{(l)}$ and $\mathbf{z}_2^{(l)}$ are samples from $q_\phi(\mathbf{z}^{(l)}, \mathbf{z}_2^{(l)}|G, N)$, and L denotes the number of samples.

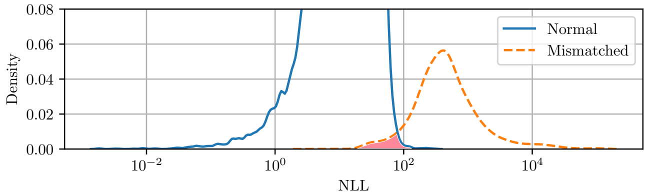


Figure 6: NLL distribution of normal and mismatched data. The red area demonstrates the phenomenon of "Inversion".

5.1.2 Entropy Gap. Although NLL is a simple and effective way to express the degree of anomalies, we find the phenomenon of "NLL Inversion" in our experiments. Figure 6 shows the NLL distribution of the data labeled as normal (Normal) and the anomalous data whose reconstructed structures are different from themselves (Mismatched). The mismatches indicate that these traces have been considered anomalous by TraceVAE, but we find that some have much lower NLLs compared with normal traces, resulting in the red area in Figure 6.

To perform more accurate anomaly detection on traces, we further investigate and try to address the implications of this phenomenon. [21] discovered that on some popular image datasets, the out-of-distribution inputs (*i.e.*, inputs not from the dataset used to train the model) might exhibit lower NLLs than the training distribution inputs, using various deep generative models including VAEs. It is thus questionable to use NLLs as anomaly scores since the anomalies could just be seen as "out-of-distribution" inputs. However, existing works neither gave a solid explanation for this problem in anomaly detection nor gave a practical solution.

Here we provide an explanation in terms of entropy. Consider Equation (1) in [21], which describes the KL-divergence between the model distribution and the true distribution:

$$\mathrm{KLD}[p^\star \,\|\, p_{\mathrm{model}}] \approx \overline{NLL_x} - \mathbb{H}\left[p^\star\right] \tag{4}$$

However, by re-arranging it, we obtain:

$$\overline{NLL_x} \approx \mathrm{KLD}[p^\star \,\|\, p_{\mathrm{model}}] + \mathbb{H}\left[p^\star\right] \tag{5}$$

where $p^\star$ is the data distribution being tested and $\overline{NLL_x}$ denotes the average value of NLLs from p_{model}.

The average of NLLs over a given data distribution $p^\star$ is composed of two parts: the KL-divergence $\mathrm{KLD}[p^\star \,\|\, p_{\mathrm{model}}]$, as well as the data entropy of $\mathbb{H}\left[p^\star\right]$. The KL-divergence measures the difference between data $p^\star$ and the model p_{model}, which is always ≥ 0, and is equal to 0 only if $p^\star \equiv p_{\mathrm{model}}$. For a well-trained model, the KL-divergence on the training distribution (denoted as p_d) inputs is expected to be lower than on the out-of-distribution ($p^\star$) inputs. But there is one more term, the data entropy $\mathbb{H}\left[p^\star\right]$, which lies aside the KL-divergence. The entropy is an intrinsic property of the test data distribution $p^\star$, which the model does not control. If $\mathbb{H}[p^\star] \leq \mathbb{H}[p_d]$, then it is possible for NLLs to be lower on $p^\star$ than on p_d.

In summary, we point out that the NLL is composed of a KL-divergence plus a data entropy term. The entropy is an intrinsic property of the test data distribution, which might cause the expectation of NLLs to be lower on out-of-distribution inputs than on the training distribution inputs. We call this problem the *entropy gap* in our paper.

5.2 Explaining the Entropy Gap

At first glance, the "anomalies" are supposed to have higher entropy than the normal data since the anomalies' uncertainty (variety) should be much larger. However, we notice that lower dimensional data can have much lower entropy than higher dimensional data, which will bring trouble to using NLLs as anomaly scores.

Suppose there are two distinct groups of traces in the dataset, namely A and B, where each trace in A has 8 nodes, and each trace in B has 32 nodes. Since time consumptions are normalized, if we model the node times as Gaussian distributions, then the stds of the times should be close to 1. Suppose we further ignore the conditional dependencies among the nodes in one trace. In that case, the entropy contributed by the time consumptions should be roughly $8 \times \mathbb{H}[\mathcal{N}(0,1)]$ in group A, and $32 \times \mathbb{H}[\mathcal{N}(0,1)]$ in group B, which forms a huge gap between these two groups (up to $24 \times \mathbb{H}[\mathcal{N}(0,1)]$). If there is a trace from group A with just one node having increased time consumption, the KL divergence raised by that single node may not cover the entropy gap caused by the extra 24 nodes from group B. As a result, the time anomalies from group A may have lower NLLs than the normal traces from group B, causing the NLLs unable to detect such anomalies.

One may seek to expand the traces from group A into 32 nodes by adding zero-time dummy nodes. However, the entropy of these dummy nodes is extremely small if the model has no minimum bound on the stds of the Gaussian distributions. This will make the entropy gap between groups A and B larger, which makes the NLLs even more misleading in anomaly detection (see experiments in Section 6.6). The entropy of traces in group A with dummy nodes is much lower than that of group B, suggesting that there is an "*intrinsic dimensionality*" for a particular dataset related to the data entropy. Adding constant-zeros to the data only increases the "*apparent dimensionality*", which cannot reduce the aforementioned entropy gap.

In summary, some anomalies may have (much) lower NLLs than some normal data, causing the NLLs inaccurate and misleading in anomaly detection. Increasing the *apparent dimensionality*, however, may not work, or may even introduce extra errors. This motivates us to reduce the entropy gap by using the adjusted NLL for anomaly detection.

5.3 Techniques for Reducing the Entropy Gap

So far, there is no existing method to exactly estimate the values of $\mathrm{KLD}[p^\star \,\|\, p_{\mathrm{model}}]$ or $\mathbb{H}[p^\star]$. However, as long as the KL-divergence $\mathrm{KLD}[p^\star \,\|\, p_{\mathrm{model}}]$ can indeed distinguish between the training data p_d and the anomalies $p^\star$, there must exist some outputs from the model (*e.g.*, the stds of time consumptions in the decoder), which can tell apart the normal traces and the anomalies. Along this direction, we propose three techniques to reduce the entropy gap:

Bernoulli & Categorical Scaling. The variables relevant to structure anomalies are the node count N and the adjacency matrix $\mathbf{A}$. N is modeled as a categorical distribution, while $\mathbf{A}$ is modeled as an $N_{max} \times N_{max}$ matrix of conditionally independent (given $\mathbf{z}$) Bernoulli distributions. Note that although $\mathbf{A}$ is an $N_{max} \times N_{max}$ matrix, A_{ij} for non-existing nodes are always filled with zeros, hence the intrinsic dimensionality of $\mathbf{A}$ differs among the traces.

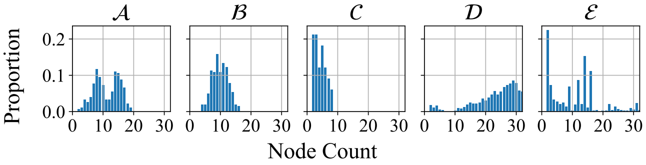


Figure 7: Histograms of the Node Counts of the Processed Traces

We observe that for structure anomalies, $p(N|\mathbf{z})$ and some of the $p(A_{ij}|\mathbf{z})$ are likely below 0.5, whereas for most normal traces these components are above 0.5 (see Section A.5). We thus propose the following Bernoulli & Categorical scaling method:

$$\begin{aligned} \log \widetilde{p_\theta(N|\mathbf{z})} &= \log p_\theta(N|\mathbf{z}) \cdot \left(N_{max}^2\right)^{I(p_\theta(N|\mathbf{z})<0.5)} \\ \log \widetilde{p_\theta(A_{ij}}|\mathbf{z}) &= \log p_\theta(A_{ij}|\mathbf{z}) \cdot (N_{max})^{I(p_\theta(A_{ij}|\mathbf{z})<0.5)} \end{aligned} \tag{6}$$

where $I(\cdot)$ is 1 if the condition is true, and 0 otherwise. Such scaling can enlarge log-likelihoods from anomaly dimensions, hence helping to reduce the entropy gap.

Node Count Normalization. In time VAE, we normalize the time distribution $\log p(x_i|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2)$ according to the node count N:

$$\log \widetilde{p(x_i|\mathbf{A}, \mathbf{Y}}, \mathbf{z}_2) = \frac{N_{max}}{N} \log p(x_i|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2) \tag{7}$$

Note that x_i are normalized to zero mean and one std over the dataset. Hence the above method can normalize the entropy among the traces with any number of nodes.

Gaussian Std-Limit. We find that the stds of the reconstructed x_i are larger on time anomalies than on the normal traces. We thus clip the stds of the Gaussian $p(x_i|\mathbf{A}, \mathbf{Y}, \mathbf{z}_2)$ as follows. We evaluate TraceVAE on the validation set to find the 99% percentile of the stds, denoted as σ_{p99}. We then clip every σ_i as $\widetilde{\sigma_i} = \min\left\{\sigma_i, \sigma_{p99}\right\}$. Since $\widetilde{\sigma_i} < \sigma_i$ are more likely to happen on anomalies, the NLLs are then enlarged more on these anomalies, which helps reduce the entropy gap.

6 EXPERIMENTS

6.1 Datasets

We collect our own datasets from a top Internet company. We use 5 microservice trace datasets ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$) in our work. The numbers of traces in these datasets are gigantic, ranging from 110k to 5.5M. Figure 7 shows the distributions of node counts in these datasets, and it can be found that the distributions of the node counts vary significantly.

Datasets $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$. These four datasets are generated through resampling and synthetic anomalies injection for evaluation. More details about data collection and preprocessing of these datasets can be found in Section A.4.

Dataset $\mathcal{E}$. To evaluate the performance of TraceVAE on *real* dataset, we also collected a dataset $\mathcal{E}$ which contains real online failure cases and manually labeled them. Dataset $\mathcal{E}$ contains 62k traces for training, 6.4k traces for validation, and 29k traces for testing. There are 162 structure anomalies and 208 time anomalies in the test set.

6.2 Experimental Setup

The most important parameters of TraceVAE are the sizes of the latent variables. We choose the sizes empirically according to the complexity of the datasets (see Section A.3). We use $K = 5$ for the structure variable $\mathbf{z}$ on $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, and $K = 10$ on $\mathcal{D}$ and $\mathcal{E}$, because $\mathcal{D}$ and $\mathcal{E}$ has much more nodes than the others. We use $K_2 = 10$ for the time variable $\mathbf{z}_2$ on $\mathcal{A}$, $\mathcal{B}$, $\mathcal{D}$ and $\mathcal{E}$, and $K_2 = 5$ on $\mathcal{C}$, because $\mathcal{C}$ has few nodes (up to 8 nodes) but large depth (up to 4 levels), suggesting it has much simpler neighborhood contexts than the others.

We use 4 layers (2 multi-heads and 500 units in each head) GAT on all five datasets. We train the whole model for 40 epochs, then fine-tune the time VAE for another 20 epochs with the parameters of all other parts being fixed. We use early-stopping techniques during training to prevent over-fitting. The source code of TraceVAE can be found in https://doi.org/10.5281/zenodo.7197839.

6.3 Evaluation Metrics

We use "total", "struct" and "time" best F-scores on all, structure and time anomalies respectively as the evaluation metrics. The best F-score is the highest F-score given all decision thresholds. Many baseline anomaly detection algorithms are sensitive to thresholds (but TraceVAE is not, see Section A.2), thus we do not choose thresholds for the baselines. The three best F-scores are computed on different test sets. The "total" F-scores are computed on all of the test traces, the "struct" F-scores are computed on the traces that are normal or with structure anomalies, and the "time" F-scores are computed on the traces that are normal or with time anomalies. Besides, we evaluate TraceVAE with ROC-AUC (see Section A.1), which is another threshold-free metric. We also designed and evaluated a simple threshold selection strategy for TraceVAE (see Section A.2).

6.4 Baselines

We compare with existing unsupervised methods that claim to detect structure and time anomalies for traces. The baselines are:

FSA [10]. A finite-state automaton (FSA) models the call dependencies and time consumptions. A trace is anomalous if the FSA rejects it. It is a classical method to detect trace anomalies.

LSTM-AD [23]. The microservices and time consumptions of nodes form a sequence and are further modeled by an LSTM VAE. The node anomaly scores are maximized instead of summed up.

TraceAnomaly [19]. Traces are encoded into vectors. Each dimension of the trace vector encodes the time consumption of a "path", where a path corresponds to a certain call context. NLLs are used as anomaly scores.

CRISP [35]. CRISP uses the same anomaly detection framework as TraceAnomaly, but encodes only the critical paths to the vectors generated with Critical Path Analysis (CPA).

VGAE. We list VGAE as a baseline (although not used in previous works for trace anomaly detection) to support our statements in Section 3.3. We use the same GNN architectures as TraceVAE. The node anomaly scores are averaged to avoid the entropy gap.

Table 1: Best F-Scores of TraceVAE and the Baselines

	$\mathcal{A}$			$\mathcal{B}$			$\mathcal{C}$			$\mathcal{D}$			$\mathcal{E}$		
	Total	Struct	Time	Total	Struct	Time	Total	Struct	Time	Total	Struct	Time	Total	Struct	Time
FSA	0.664	0.497	0.497	0.737	0.583	0.583	0.813	0.685	0.685	0.527	0.358	0.358	0.199	0.090	0.134
LSTM-AD	0.745	0.470	0.872	0.710	0.420	0.820	0.565	0.184	0.881	0.758	0.558	0.927	0.442	0.213	0.513
TraceAnomaly	0.560	0.091	0.832	0.570	0.105	0.812	0.528	0.182	0.717	0.530	0.090	0.775	0.410	0.048	0.565
CRISP	0.438	0.164	0.502	0.416	0.179	0.520	0.526	0.092	0.769	0.334	0.090	0.382	0.344	0.041	0.422
VGAE	0.275	*N/A*	0.454	0.261	*N/A*	0.408	0.631	*N/A*	0.682	0.387	*N/A*	0.625	0.450	*N/A*	0.529
TraceVAE	**0.954**	0.935	**0.945**	0.944	0.903	**0.940**	0.923	0.911	**0.911**	**0.980**	0.988	**0.965**	**0.791**	**0.813**	**0.772**
TraceVAE-FC	0.936	0.889	0.938	0.925	0.877	0.936	0.915	0.903	0.907	0.975	0.983	0.959	0.729	0.742	0.677
TraceVAE-SingleZ	0.854	0.849	0.829	0.888	0.921	0.816	0.919	0.881	0.894	0.946	0.943	0.931	0.632	0.702	0.507
TraceVAE-DimEx	0.789	0.768	0.777	0.818	0.705	0.863	0.841	0.892	0.762	0.897	0.901	0.882	0.579	0.268	0.692
TraceVAE-NLL	0.918	0.930	0.867	0.928	0.954	0.879	0.927	0.902	0.885	0.957	0.969	0.937	0.645	0.769	0.561
TraceVAE-BCScale	0.925	**0.967**	0.868	0.931	**0.971**	0.878	0.925	**0.925**	0.883	0.965	**0.990**	0.935	0.662	0.813	0.558
TraceVAE-NCNorm	0.918	0.877	0.916	0.904	0.891	0.882	0.873	0.798	0.880	0.965	0.964	0.955	0.687	0.731	0.627
TraceVAE-StdLimit	0.940	0.910	0.928	**0.947**	0.924	0.934	**0.930**	0.892	0.904	0.963	0.957	0.957	0.732	0.769	0.680

6.5 Comparison with Baselines

Table 1 shows the best F-scores of TraceVAE and baselines. In addition, we also show the ROC-AUC results in Section A.1, which have similar trends to the F-Score. The results show that our TraceVAE consistently outperforms all the baselines on all five datasets for both structure and time anomalies by large margins.

FSA achieves 1.0 recalls on all five datasets and both anomalies, but its precisions are very low. Also, the outputs of the FSA are either "accepted" or "rejected", with no continuous anomaly scores, making it difficult to express the degree of anomalies.

LSTM-AD claimed to detect structure anomalies, but the performance is poor, likely because the node sequence + LSTM cannot preserve all structure information. *TraceAnomaly & CRISP* also claimed to detect "innovation path" anomalies (a combination of the structure and time anomalies in our paper). Still, their best F-scores on structure anomalies are very low, suggesting they may be hard to effectively model the structural information of traces in large-scale systems. These three works use VAEs without GNNs, so their poor performance on structure anomalies highlights the need to use GNNs. Moreover, for TraceAnomaly and CRISP, since the numbers of existing paths are not equal for all the traces and the non-existing paths are filled with -1, the *entropy gap* might be a problem.

VGAE shows poor performance on time anomalies. This may be due to the independence of the node-level $\mathbf{Z}$, which highlights the benefit of our graph-level $\mathbf{z}_2$ in TraceVAE. VGAE cannot detect structure anomalies by its architecture, so we mark its results in Table 1 as N/A.

6.6 Ablation Study

Table 1 also shows the experiment results of the ablation study. Note that the anomaly score comprises the (modified) NLLs of both structure and time VAEs. Thus increasing the F-scores on one type of anomaly may decrease the F-scores of the other type.

The dispatching layer. We replace the *dispatching layer* in TraceVAE with a fully-connected network (while still using the adjusted NLLs). The result is shown as TraceVAE-FC in Table 1, which shows that the dispatching layer effectively improves the detection accuracy of time anomalies. Surprisingly, the detection of structural anomalies is also improved, which is likely to be caused by the less misleading in time NLLs (as the final anomaly score is the sum of both NLLs).

The dual-variable graph VAE.. We train a single-variable graph VAE instead of the original dual-variable one. The result is shown as TraceVAE-SingleZ in Table 1. Our TraceVAE improves the F-scores over TraceVAE-SingleZ on "total" anomalies by $0.4\%-25.2\%$, which highlights the benefits of our design choice of using the two separated latent variables.

Apparent dimensionality. We show TraceVAE trained and tested with all traces expanded to 32 nodes by filling zero-time dummy nodes, as TraceVAE-DimEx in Table 1. We see that increasing the *apparent dimensionality* makes the performance of anomaly detection much worse than TraceVAE-NLL, which supports our statements about the intrinsic and apparent dimensionality.

The techniques to reduce entropy gap. For each trained TraceVAE, we derive four variants by directly replacing the anomaly scores (without retraining a new model) as follows: (1) *TraceVAE-NLL* uses only the original NLL; (2) *TraceVAE-BCScale* uses NLL + Bernoulli & Categorical Scaling; (3) *TraceVAE-NCNorm* uses NLL + Node Count Normalization; (4) *TraceVAE-StdLimit* uses NLL + Gaussian Std-Limit.

We can see that *TraceVAE* consistently improves the F-scores over *TraceVAE-NLL* on time anomalies by 2.6% – 37.6%. On structure anomalies, TraceVAE has smaller improvements (0.5% – 5.7%) except on dataset $\mathcal{B}$, where TraceVAE is worse than TraceVAE-NLL. Note that on dataset $\mathcal{B}$, TraceVAE has a very large improvement on time anomalies, and thus the F-score on "total" anomalies is still better (by 1.6%), which suggests that this counterexample is caused by the problem in the aforementioned trade-off between structure and time NLLs.

TraceVAE-BCScale alone improves the F-scores on structure anomalies by 1.6% – 5.7%, with little or no harm to time anomalies.

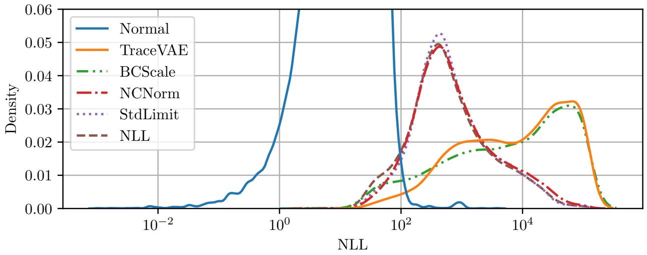


Figure 8: The NLL Distribution with Different Techniques

TraceVAE-NCNorm alone improves the F-scores on time anomalies for most datasets, except for C, which has very few nodes. *TraceVAE-StdLimit* consistently improves the F-scores on time anomalies by 1.9%−13.5%. However, both *TraceVAE-NCNorm* and *TraceVAE-StdLimit* may decrease the F-scores on structure anomalies.

To further investigate how these techniques affect anomaly detection performance, we show the NLL distributions of $\mathcal{D}$ under different settings in Figure 8. It can be found that the intersection areas between the "Normal" line and the "TraceVAE" line are getting smaller, indicating that the proposed techniques effectively reduced the number of anomalous traces with low NLL. The effectiveness of our proposed techniques highlights the benefits of our proposed theory about the entropy gap.

6.7 Threats to Validity

The internal threats to validity mainly lie in the configuration of the models. The hyper-parameters in TraceVAE and the baselines need adjustment to fit different datasets. The techniques to reduce the entropy gap are relatively empirical. The decision boundary of 0.5 for BCScale may need tuning in other models. Nevertheless, as long as the model is well-trained and KLD$[p^{\star} \parallel p_{\text{model}}]$ can distinguish between the normal data and anomalies, certain techniques can constantly be developed to reduce the entropy gap.

The external threats to validity mainly lie in the datasets. To alleviate such threats, we collected data of different sizes from different parts of the real system. We also use both manually labeled and synthetic anomalies data in the five datasets to comprehensively evaluate the performance of our model.

7 RELATED WORK

Anomaly detection is a hot topic owing to its wide application. Time series-based [6, 31], log-based [9], and trace-based [19, 35] approaches are employed to detect system failures. Traces record invocation relationships between microservices, which are useful in diagnosing failures while more difficult to be accurately modeled. To model the invocations in traces, FSA [10] constructs a Finite State Automaton to learn the time distributions and detect anomalies with k-sigma [5]. While FSA distinguishes between different time distributions by preceding call paths, the tree structures are not considered, thus making it hard to detect drop anomalies. Many works introduce deep learning to trace anomaly detection to learn the complex structural features. TraceAnomaly [19] and CRISP [35] represent a trace as an stv (service trace vector) and detect both drop and time anomalies with VAE reconstruction. TraceLingo [32] and DeepTraLog [33] capture the tree structures in traces with Tree-LSTM [1] and GNN [11] to obtain accurate structural representations. However, TraceLingo needs supervised training, and DeepTraLog uses extra log data, which cannot be directly compared with TraceVAE.

As a common technique in deep learning, VAE has also been widely employed in anomaly detection to model normal patterns [2, 25–27, 31]. To avoid excessive latent variables, flow-based generative models [7, 18, 30] are employed to express complex latent distributions. Although VAE has been successfully applied in many areas, some works find the phenomenon of NLL inversion [21] in out-of-distribution data. In [29] and [12], some techniques are proposed to mitigate this phenomenon in the representation learning of these kinds of data or try to detect them. However, the impact of this phenomenon on anomaly detection has not been analyzed in the existing studies. Our study takes the perspective of anomaly detection and tries to reduce this entropy gap.

8 CONCLUSION

In this paper, we study the problem of anomaly detection on microservice traces and propose a novel model named TraceVAE. In TraceVAE, a dual-variable graph VAE architecture and a novel dispatching layer are used to encode the structure of traces and the time consumption of nodes separately. We find the phenomenon of "NLL Inversion" in trace anomaly detection, which may cause performance degradation. To investigate and mitigate this phenomenon, we point out that the NLL can be decomposed into KL divergence plus the data entropy. Lower-dimensional anomalies can have much lower entropy than higher-dimensional normal traces, making the NLLs misleading for anomaly detection. We propose three techniques to reduce the entropy gap: Bernoulli & Categorical Scaling, Node Count Normalization, and Gaussian Std-Limit. On five trace datasets from a top Internet company, our proposed TraceVAE outperforms all baselines and achieves excellent best F-scores. The results of the ablation study demonstrate the effectiveness of our model design and the three techniques to reduce the entropy gap. Our discovered entropy gap may also have a broader impact in other domains, serving as a key ingredient to solving the problem raised by [21].

ACKNOWLEDGMENTS

We thank Changhua Pei and Shenglin Zhang for their helpful discussions and suggestions on this work. This work is supported by the National Key Research and Development Program of China under Grant 2019YFE0105500, the Research Council of Norway under Grant 309494, the State Key Program of National Natural Science of China under Grant 62072264, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

REFERENCES

[1] Mahtab Ahmed, Muhammad Rifayat Samee, and Robert E Mercer. 2019. Improving tree-LSTM with tree attention. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. IEEE, 247–254.

[2] Jinwon An and Sungzoon Cho. 2015. *Variational Autoencoder Based Anomaly Detection Using Reconstruction Probability.* Technical Report. Technical Report.

[3] Liang Bao, Qian Li, Peiyao Lu, Jie Lu, Tongxiao Ruan, and Ke Zhang. 2018. Execution Anomaly Detection in Large-Scale Systems through Console Log Analysis. *Journal of Systems and Software* 143 (2018), 172–186. https://doi.org/10/gm7rdq

[4] Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. 2016. Importance Weighted Autoencoders. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1509.00519

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15. https://doi.org/10/d44frx

[6] Wenxiao Chen, Haowen Xu, Zeyan Li, Dan Peiy, Jie Chen, Honglin Qiao, Yang Feng, and Zhaogang Wang. 2019. Unsupervised Anomaly Detection for Intricate KPIs via Adversarial Training of VAE. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1891–1899.

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using Real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=HkpbnH9lx

[8] Kota Dohi, Takashi Endo, Harsh Purohit, Ryo Tanabe, and Yohei Kawaguchi. 2021. Flow-based self-supervised density estimation for anomalous sound detection. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 336–340.

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. (2017).

[10] 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/fmzggh

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[12] Dan Hendrycks, Mantas Mazeika, and Thomas G. Dietterich. 2019. Deep Anomaly Detection with Outlier Exposure. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=HyxCxhRcY7

[13] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations*.

[14] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. In *Bayesian Deep Learning Workshop (NIPS 2016)*. arXiv:1611.07308

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR 2017*. Toulon, France.

[16] Youngchun Kwon, Jiho Yoo, Youn-Suk Choi, Won-Joon Son, Dongseon Lee, and Seokho Kang. 2019. Efficient Learning of Non-Autoregressive Graph Variational Autoencoders for Molecular Graph Generation. *Journal of Cheminformatics* 11, 1 (Nov. 2019), 70. https://doi.org/10/gm64fs

[17] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. 2017. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325* (2017).

[18] Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. 2021. Multivariate Time Series Anomaly Detection and Interpretation Using Hierarchical Inter-Metric and Temporal Embedding. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3220–3230. https://doi.org/10/gngc5t

[19] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, and Wenman Xue. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 48–58. https://doi.org/10/gjqs7d

[20] Joshua Mitton, Hans M. Senn, Klaas Wynne, and Roderick Murray-Smith. 2020. A Graph VAE and Graph Transformer Approach to Generating Molecular Graphs. In *arXiv:2104.04345 [Cs]*. arXiv:2104.04345 [cs]

[21] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. 2019. Do Deep Generative Models Know What They Don't Know?. In *7th International Conference on Learning Representations, ICLR 2019*. New Orleans, LA, USA. arXiv:1810.09136

[22] 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/gngc57

[23] 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/gn3pv4

[24] Martin Simonovsky and Nikos Komodakis. 2018. Graphvae: Towards Generation of Small Graphs Using Variational Autoencoders. In *International Conference on Artificial Neural Networks*. Springer, 412–422.

[25] Andrea Stocco, Paulo J Nunes, Marcelo D'Amorim, and Paolo Tonella. 2022. Thirdeye: Attention maps for safe autonomous driving systems. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[26] Andrea Stocco and Paolo Tonella. 2022. Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems. *Journal of Software: Evolution and Process* 34, 10 (2022), e2386.

[27] 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*. 359–371.

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018*. Vancouver, BC, Canada.

[29] Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. 2022. Handling Distribution Shifts on Graphs: An Invariance Perspective. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[30] Haowen Xu, Wenxiao Chen, Jinlin Lai, Zhihan Li, Youjian Zhao, and Dan Pei. 2020. Shallow VAEs with RealNVP Prior Can Perform as Well as Deep Hierarchical VAEs. In *Neural Information Processing*, Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King (Eds.). Springer International Publishing, Cham, 650–659.

[31] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, and Yang Feng. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 187–196.

[32] Yong Xu, Yaokang Zhu, Bo Qiao, Hongshu Che, Pu Zhao, Xu Zhang, Ze Li, Yingnong Dang, and Qingwei Lin. 2021. TraceLingo: Trace representation and learning for performance issue diagnosis in cloud services. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 37–40.

[33] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. 2022. DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning. (2022).

[34] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[35] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. 2022. CRISP: Critical Path Analysis of Large-Scale Microservice Architectures. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 655–672.

A ADDITIONAL DETAILS OF THE EXPERIMENTS

A.1 The ROC-AUC Results

Table 2: ROC-AUC of TraceVAE and the Baselines

Dataset	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$
FSA	0.497	0.583	0.685	0.358	0.104
LSTM-AD	0.262	0.247	0.106	0.303	0.079
TraceAnomaly	0.492	0.513	0.534	0.476	0.395
CRISP	0.355	0.383	0.494	0.207	0.195
VGAE	0.263	0.262	0.678	0.351	0.432
TraceVAE	**0.984**	**0.981**	**0.954**	**0.994**	**0.830**

The ROC-AUC results are shown in Table 2. As the F-Score results in Table 1, the ROC-AUC results also illustrate the advantages of TraceVAE in anomaly detection.

A.2 Threshold Selection

Table 3: The F-Scores Using Selected Thresholds

Dataset	Total	Struct	Time
$\mathcal{A}$	0.954	0.926	0.942
$\mathcal{B}$	0.943	0.898	0.930
$\mathcal{C}$	0.910	0.896	0.888
$\mathcal{D}$	0.968	0.949	0.940
$\mathcal{E}$	0.763	0.782	0.723

In TraceVAE, we use a simple way to choose the threshold: we compute the anomaly scores $s(G)$ on the whole validation set and use the 99% percentile of $s(G)$ as the threshold. Table 3 shows the F-scores using the selected thresholds, which are actually very close to the best F-scores (Table 1).

A.3 The Statistics of the Experiment Datasets

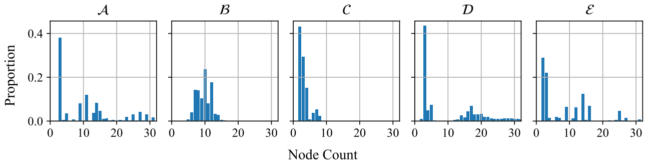


Figure 9: Histograms of Node Counts of the Original Traces

Table 4 shows the basic statistics of the 5 datasets before and after processing. It can be noticed that the datasets we choose have different node counts and depths, which can help us to verify the performance of the algorithm in different scenarios. Figure 9 and Figure 10 demonstrate the distribution of their node counts, respectively. For datasets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, the processed dataset has a smoother node count distribution, which means that the resampled dataset also maintains a balance of different node counts.

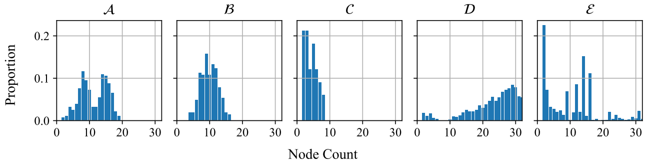


Figure 10: Histograms of Node Counts of the Processed Traces

A.4 Details of Data Collection and Preprocessing

Each dataset contains the traces collected from different business subsystems in an online large-scale microservices system. These subsystems are basically independent, so we divide them into five datasets and train different detectors for each.

For datasets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, we find the original datasets are highly imbalanced, where small and simple traces are dominant. Such imbalance (towards simple traces) would bias the model training. Thus we need to resample the datasets to balance the traces of different structures. So we preprocess and resample the original datasets as follows. *Firstly*, we drop all traces whose total time consumptions are larger than the 99% percentile (to reduce the interference of time anomalies on model training). *Secondly*, we use the 3-tuple *(trace depth, root node API, appeared APIs of all nodes)* as the key to divide the traces into separated groups and drop the groups with less than 100 samples to only keep the dominant parts for training. *After that*, we divide each group into 10:2:5 disjoint parts and further obtain the train, validation, and test sets (having 100, 20, and 50 thousand traces) by sampling from each part of the group separately. The number of traces taken from each group is equal. *Finally*, we sort and merge the nodes of the traces according to the method described later. Note that these preprocessing techniques can also be applied to model training in online systems.

The anomalies from online systems are rare and are not guaranteed to cover all groups mentioned above. For each dataset of $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, to perform controlled experiments on as wide a variety of anomalies as possible, we inject synthetic anomalies. We also generate 2,500 structure anomalies and 2,500 time anomalies (in total 5,000 anomalies) based on the test set as follows:

Structure anomalies. To generate a trace of structure anomaly, we first pick a normal trace and sample a discard rate $p \sim U[0.1, 0.5]$. We then repeat the following steps until the p portion of nodes is discarded: pick a random node, then discard it and all its children.

Time anomalies. To generate a trace of time anomaly, we first pick a normal trace and sample an altering rate $p \sim U[0.1, 0.5]$. We then repeat the following steps until the p portion of nodes is altered: pick a random node, sample $\rho \sim U[2, 10]$, then set the time consumption of the node to $\rho \times t_{p99}$, where t_{p99} is the 99% percentile of the time consumptions of the corresponding microservice of node, and finally add the time increment to all the parents of that node.

For dataset $\mathcal{E}$, to evaluate the TraceVAE results under real data distribution, we do not resample the data.

Table 4: Statistics of the Experiment Datasets

	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$
Before Re-sampling and Merging					
Original Trace Count	2,341,237	1,396,697	4,504,422	5,479,257	110,599
API Count	71	57	33	98	1487
Root Node API Count	7	10	5	23	403
After Re-sampling and Merging					
Max Node Count	19	16	8	32	32
Max Depth (Including Root Node)	2	2	4	5	10
Time Consumption 90% Percentile (ms)	1,394	1,703	95	209	1,190
Time Consumption 95% Percentile (ms)	1,767	3,539	112	236	2,987
Time Consumption 99% Percentile (ms)	4,474	6,149	150	280	6,800

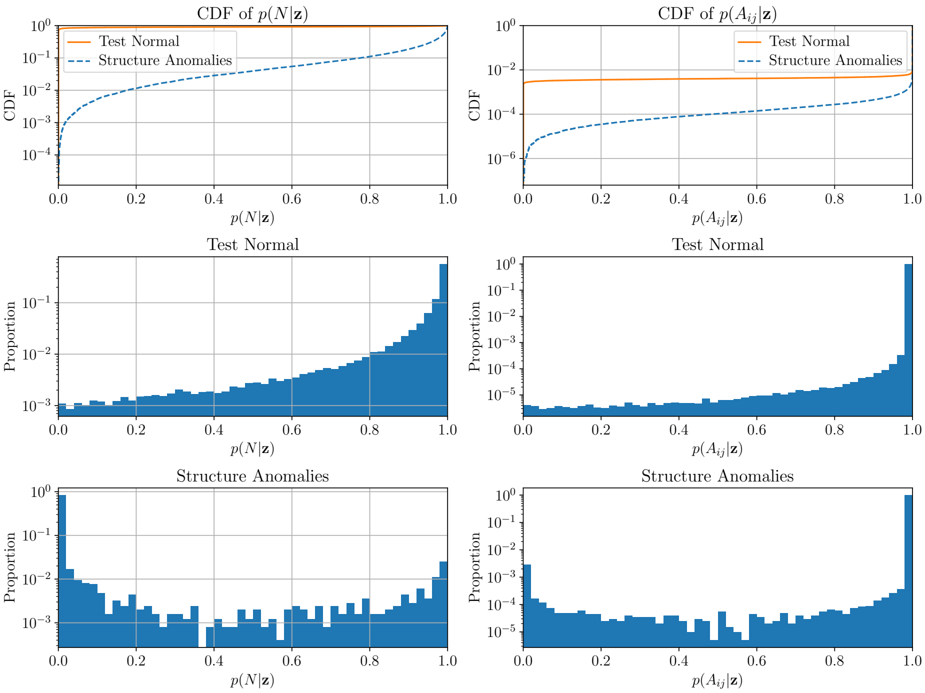


Figure 11: Histograms of $p(N|z)$ and $p(A_{ij}|z)$ of Dataset $\mathcal{A}$

A.5 The Histograms of Reconstruction Probability

Figure 11 shows the histograms of the reconstruction probability of node counts $p(N|\mathbf{z})$ and edges $p(A_{ij}|\mathbf{z})$ on dataset $\mathcal{A}$. The histograms on other datasets are similar.

We can see that almost all $p(N|\mathbf{z})$ are below 0.5 on the structure anomalies, whereas the portion of $p(N|\mathbf{z})$ below 0.5 on the normal traces is just $10^{-2} - 10^{-1}$. This fact motivates us to use categorical scaling on $\log p(N|\mathbf{z})$.

Meanwhile, the portion of $p(A_{ij}|\mathbf{z})$ below 0.5 on the structure anomalies is $10^{-3} - 10^{-2}$, which is much larger than the portion of $p(A_{ij}|\mathbf{z})$ below 0.5 on the normal traces ($\approx 10^{-4}$). This fact motivates us to use Bernoulli scaling on $\log p(A_{ij}|\mathbf{z})$. Note that A is a matrix of fixed size $N_{max} \times N_{max} = 32 \times 32$, so the total number of $p(A_{ij}|\mathbf{z})$ for a given trace is 1024. When structure anomaly happens, some nodes are missing, which means there should be a few (and just a few) A_{ij}, which are supposed to be 1, equal to 0 in the given trace. Given that the maximum number of nodes is $N_{max} = 32$, it means that there should be around $1/1024 - 32/1024$ below 0.5, which is consistent with the observed portion $10^{-3} - 10^{-2}$.