Enhanced Training of Response Time Anomaly Detectors Using Diffusion Models

Wenxiang Luo
Department of Computing
Imperial College London
London, UK
wenxiang.luo21@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London
London, UK
g.casale@imperial.ac.uk

***Abstract*—Machine learning (ML) approaches have grown in popularity in recent years as a way to identify anomalies in microservice-based applications. However, training ML models may suffer from time constraints and limited real-world failure data availability. In this paper, we propose a method to mitigate this issue using diffusion models for training data augmentation. Response time data collected using distributed traces is used to train diffusion models leveraging a customized UNet proposed in the paper. The resulting diffusion models can then generate new data to improve the training of response time anomaly detectors. Experiments using the *DeathStarBench* microservices architecture demonstrate that the proposed approach increases the accuracy after training of anomaly detection models by 20%. We further show that the response time data generated by our diffusion models cannot be distinguished by classic discriminators, which confirms that the generated data are of high quality.**

***Index Terms*—Microservice, Anomaly Detection, Machine Learning, Diffusion Model**

I. Introduction

The microservice-based architecture, a modern software development approach, divides applications into small independent microservices [1]. These microservices are frequently updated to remove errors, improve performance, and introduce new features. However, frequent updates increase the likelihood of failure due to insufficient time for exhaustive testing to identify errors in the source code [2]. ML methods are increasingly used to address this problem, but these models may need to be retrained as the updates change the normal behavior of the applications. The retraining requires substantial time and resources, and the frequent updates lead to a data shortage. This issue becomes even worse since the majority of the limited data are normal instances because the applications are typically running under normal conditions. This yields models unable to identify anomalous data accurately, which restricts their effectiveness.

Data augmentation is an effective technique to improve the performance of ML models when there is limited data. By increasing the size and diversity of the training dataset, data augmentation improves classification accuracy and allows the model to generalize well to unseen data. This technique is particularly useful when there is a data shortage or imbalanced datasets since it offers additional data that is similar, but nonetheless non-identical, to the original one and from thus further insights may be learned [3]. Therefore, models that use data augmentation are able to more effectively extract patterns within the dataset, which leads to more reliable and accurate predictions.

Diffusion models are a type of popular generative model in recent years because of their ability to generate realistic data. In this paper, a new method is proposed to use diffusion models for data augmentation in the context of microservice applications, which aims to mitigate the challenges caused by data shortage and imbalanced datasets. Microservice traces are initially collected by running a benchmark, and features, such as the execution time of each span and response time between spans, are extracted from raw traces to create datasets. These datasets are then preprocessed to be suitable for training diffusion models to generate new traces that capture patterns of these datasets. After training, new microservice traces are generated by these diffusion models, which are subsequently augmented into the original datasets to introduce unseen patterns. By using both real and generated data, ML models are able to effectively extract patterns of anomalies, which leads to an increased accuracy in anomaly detection.

Related work. In recent microservice applications, the number of microservices and their interactions have increased significantly. This poses challenges in determining if microservices in applications have failed. TraceAnomaly [4] uses the response times of spans to train a variational autoencoder, which attempts to reconstruct the inputs. TraceGra [5] trains a variational graph autoencoder to reconstruct the inputs. UAC-AD [6] employs contrastive learning for anomaly detection, and trains an autoencoder to reconstruct the input data. Nedelkoski et al. [7] propose a supervised learning method for anomaly detection and classification using an autoencoder and a convolution neural network. Seer [8] uses a deep neural network that comprises convolution layers and LSTM layers to predict the likelihood that a particular microservice is responsible for an anomaly.

The applications of previous methods are limited by their shortcomings as they collect data directly to train ML models, and may suffer from performance issues, as the amount of collected data is limited and highly imbalanced because of the frequent updates of microservices. Some solutions mitigating the problem caused by imbalanced datasets are ineffective in

this case. Software fault injection evaluates the system behavior by deliberately injecting faults to identify errors during development [9], requiring careful selection of fault types and injection points [10]. Over-sampling methods [11] create new training data by interpolating between the minority instances and their neighbors, but the generated data lies within the range of existing minority data, leading to poor representation of underlying patterns in complex data distributions [12]. One-class classification, an anomaly detection technique trained on data from a single class, suffers from performance issues in high dimensions [13].

The method proposed in this paper mitigates these challenges by using data augmentation, which involves developing diffusion models to generate new data. It allows models to effectively extract patterns of anomalies with a small amount of training data, which is beneficial for situations where collecting anomalous data is challenging.

Outline. The rest of this paper is organized as follows. The background knowledge on diffusion models is provided in Section II. Section III presents the problem statement and a motivating example that illustrates the negative effect of imbalanced datasets. Section IV presents the proposed methodology, and the validation follows in Section V. Finally, the conclusion is given in Section VI.

II. Background

Denoising diffusion probabilistic model (DDPM) is a type of diffusion model used in this paper. The training process of DDPMs consists of a forward process where Gaussian noise is gradually added until the data becomes pure noise, and a backward process, where a neural network is trained to learn to remove the noise to restore the input data.

Let $q(x_0)$ be the distribution of the uncorrupted data, $\beta_t \in (0, 1)$ be a hyperparameter that determines how much Gaussian noise is added to the input data, and a training sample be obtained as $x_0 \sim q(x_0)$. The forward process [14] is

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I) \quad \forall t \in \{1, 2, 3, ..., T\} \tag{1}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, T is the number of time steps in the forward process, I is an identity matrix with the same size as the input data, and $\mathcal{N}(x; \mu, \sigma)$ is the normal distribution that outputs x with mean μ and variance σ. If the input data x_0 is provided, the corrupted data x_t is obtained by

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon \tag{2}$$

where $\epsilon \sim \mathcal{N}(0, I)$.

The backward process starts from the pure noise $p_T = \mathcal{N}(x_T; 0, I)$, which is obtained at the end of the forward process. A neural network is trained in this process, and it estimates the noise that needs to be removed to restore the input data. As Gaussian noise is introduced during the forward process, the backward process is also defined by a mean μ_θ and a variance $\beta_t I$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta_t I) \tag{3}$$

where θ represents the parameters of the neural network trained in the backward process. Ho et al. [14] suggest that the model only learns to predict the mean and set the variance dependent on time steps. The suggestion is subsequently validated, and it also found that the mean has a more decisive influence than the variance [15]. By applying this simplification, the relationship between x_{t-1} and x_t is

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon \tag{4}$$

A loss function is required to train the neural network in the backward process, and Ho et al. [14] suggests that it is advantageous to use the loss function below for training

$$L = ||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)||^2 \tag{5}$$

where ϵ_t is Gaussian noise sampled at time step t. Equation (5) indicates that the objective of training the neural network is to minimize the mean squared error (MSE) between the noise introduced during the forward process and the predicted one.

A common architecture to implement a diffusion model is to build a UNet, which is initially designed for image segmentation tasks where the sizes of inputs and outputs are identical. This is similar to diffusion models where the input is corrupted images, and the model predicts the amount of noise that should be removed to restore the original image. The UNet achieves this goal by extracting features using down-sampling layers and reconstructing the input data by up-sampling layers. Residual connections could be introduced to preserve the information between the down-sampling and up-sampling parts. There are studies using UNet for anomaly detection in medical areas [16], but to the best of our knowledge it has not been used in microservices for response time anomaly detection. This contribution is developed in the present paper.

III. Motivation

A. Problem Statement

1) Environment: We consider a microservice application that contains $\boldsymbol{S}$ services packaged into $\boldsymbol{C}$ containers. These containers are assumed to be deployed on $\boldsymbol{N}$ machines with different hardware configurations. The system is monitored via distributed tracing with the goal of detecting anomalies in the microservices. The collected traces from the monitoring toolchain are assumed to be used as an input to a ML model for anomaly detection. If the model detects any deviation from the normal status for one or more microservices, an error message is generated, which enables the maintenance teams to investigate and address the problem.

On top of the above, some additional assumptions used in this paper are as follows. First, some container orchestration tools may dynamically migrate containers within the distributed system, we assume instead them to be pinned upon first deployment. Next, we assume that training and production runs are on similar technical environments, so that training data obtained via testing may be deemed as representative of production. This is critical as the model predictions are severely affected if the testing data are significantly different

from its training data [17]. Lastly, we assume that microservice response time depends only on the application and the technical infrastructure it runs in, not on human factors such as the delay a human imposes for manually supplying data to a process. Hence, the latter delays are not considered as part of the system response time.

2) Feature Extraction: In the paper, we focus on continuous features that describe latencies observed in distributed tracing spans. While in general anomaly detection can rely upon multiple continuous, discrete and categorical features, a broad catalogue of techniques exists from ML for blackbox identification of anomalies. However, response times are fairly different from most other features in that they are closely related to the software architecture and software call inter-dependencies, thus they can additionally benefit from correlation to the graph topology of the microservices-based application. As the latter can therefore leverage a different treatment than other features, we insist specifically only on response time metrics throughout the paper. However, several of the ideas presented may be generalized in principle to embrace a broader spectrum of system metrics.

Response time features for model training are extracted from raw traces collected from the distributed tracing system. Assume span A is the root span with two child spans, B and C. A tree structure is first built to represent the invocation chain in each trace. Based on the invocation chain, two types of features are extracted from the trace: the span execution time (t_A, t_B, and t_C), and the transition time (t_{AB} and t_{BA}) from parent span A to child span B and vice versa. However, the transition times between spans B and C are not included, as there is no direct invocation between these spans.

After the extraction, the execution times and response times from all spans in a trace are used to create a tuple. The tuples from multiple traces are stacked to create a vector for training UNet, represented as $(t_A^{(j)}, t_B^{(j)}, ..., t_{AB}^{(j)}, t_{BA}^{(j)}, ...)$, where the subscripts denote the feature names and superscripts show the index. The features in the resulting vector follow a fixed order, starting with the execution time of the root span and then its child spans. The response times are ordered similarly, starting with the responses between the root span and its children.

3) Model Training: Training ML models typically requires balanced datasets to ensure the model performance is reliable. However, in real-world microservice applications, it is challenging to obtain a balanced dataset that is suitable to train an anomaly detection model with high accuracy.

The anomaly detection model trained by imbalanced datasets can identify normal data accurately as it is the majority class in the dataset. This results in a high overall performance for the models. However, the model is unable to detect anomalies accurately since it is the minority class in the dataset, and it is insufficient for the model to extract patterns of error status. This causes low performance in determining whether the input data is anomalous, although the model is capable of identifying normal inputs. Therefore, the overall performance is high, but the model still lacks the ability to detect anomalies, which makes it less useful.

TABLE I
LOW VS. MIDDLE WORKLOADS: 5% AND 1% ANOMALOUS DATA (IF = ISOLATION FOREST, KM = K-MEANS)

Anomal. (%)	Model	Performance		
		Accuracy	Recall	**F1 Score**
5%	IF	0.980	0.581	**0.735**
	KM	0.970	0.453	**0.624**
	MLP	0.988	0.769	**0.869**
	SVM	0.987	0.732	**0.845**
1%	IF	0.984	0.402	**0.574**
	KM	0.983	0.346	**0.514**
	MLP	0.997	0.647	**0.785**
	SVM	0.995	0.541	**0.702**

B. Illustrative Example

In this section, we illustrate a basic experiment to showcase the reduced effectiveness of microservices anomaly detectors in the presence of imbalanced datasets. In this experiment, four ML models are trained using imbalanced datasets that contain only a fraction of 5% or 1% anomalous data points. The results are displayed in Table I. The inputs to these models are vectors, with rows representing individual traces and columns containing features such as execution and response times. It is clear that the recall is low, particularly for unsupervised learning algorithms, which becomes lower than 0.5. This low recall indicates that a significant number of anomalies are classified as normal data. The reason for this is caused by the highly imbalanced dataset, where the models cannot extract patterns for anomalies effectively. This leads to a low F1 score, which is dominated by recall in this case.

IV. METHODOLOGY

There are five stages in the method. In the first two stages, experiments are performed, and microservice traces are collected. The collected data is then preprocessed, as shown in IV-B, to ensure that it is suitable for training ML models. The preprocessed data is used to train diffusion models in the fourth stage, and new traces are generated in the last stage.

A. Experiments & Trace Collection

Traces are first generated by running the benchmark for a specific period, and the workload is configured by three parameters: number of threads, number of connections, and requests per second. Then, these traces are downloaded from the monitoring tools. To retrieve the desired traces, a URL is constructed with specific parameters to filter the data accordingly. The output of the selected traces varies based on the provided parameters, and these traces are subsequently obtained by sending a request to the configured URL.

B. Data Preparation

1) Outlier Removal: It is critical to remove outliers to ensure the performance of the model is not adversely affected. The outlier removal process involves calculating quartiles. The first and third quartiles, denoted as $\boldsymbol{Q_1}$ and $\boldsymbol{Q_3}$, are obtained, and the interquartile range ($\boldsymbol{IQR}$) is then calculated as the difference between $\boldsymbol{Q_3}$ and $\boldsymbol{Q_1}$. A value is considered an

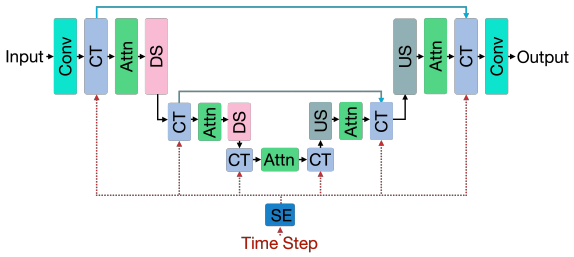


Fig. 1. The architecture of the proposed UNet

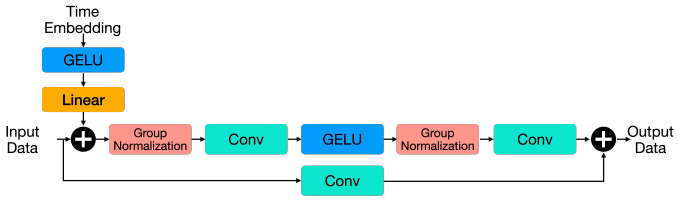


Fig. 2. Convolution Time Block

outlier if it lies beyond three times the $\boldsymbol{IQR}$ from the first or third quartile [18].

2) Feature Selection: The diffusion model developed in this paper is essentially a UNet, comprising several down-sampling and up-sampling layers. These layers perform the operations of division and multiplication by two. Down-sampling layers can handle an odd number of features as input, while a size mismatch occurs during up-sampling, leading to exceptions in model training. Therefore, it is necessary to ensure the number of input features is even to guarantee the down-sampling and up-sampling operations. To achieve this, several features need to be removed from the dataset before training the model to ensure an even number of features after multiple divisions by two. Principal Component Analysis is used to select the most important features by evaluating their influence on the principal components. The least significant features are removed to ensure that an even number of features are input into the model.

3) Data Standardization: The ranges of values for features extracted from raw traces vary considerably, which causes the model to be biased towards the features with larger values. To prevent this bias, the data is standardized by subtracting the mean from each feature and scaling it to unit variance.

C. Model Training

1) Architecture: The neural network built in this paper is a UNet [19] with several modifications, and its basic structure is shown in Figure 1, where $\boldsymbol{SE}$ is a sinusoidal embedding block, $\boldsymbol{Conv}$ represents convolution layers, $\boldsymbol{CT}$ denotes convolution time blocks, $\boldsymbol{Attn}$ is attention layers, and $\boldsymbol{DS}$ and $\boldsymbol{US}$ represent down-sampling layers and up-sampling layers, respectively.

a) Sinusoidal Embedding Block: Sinusoidal embedding has been used in transformers [20] to encode positional information to address the challenge of encoding sequence order. It is also employed in diffusion models since the time steps are critical for the model to understand the amount of noise introduced into the input data. By using sinusoidal embeddings, diffusion models can effectively capture the progression of noise introduced into the input data at each time step, which enables the model to understand the noise introduction in the forward process and the noise removal in the backward process.

b) Convolution Time Block: The structure of this block is displayed in Figure 2, and it receives the time embeddings $\boldsymbol{TE}$ from the sinusoidal embedding block and the input data $\boldsymbol{IN}$. The time embedding is first passed to a $\boldsymbol{GELU}$ activation function and a linear layer to match the size of the input data. The embedding is then reshaped and added to the input data as,

$$H = IN + rearrange(MLP(TE)) \tag{6}$$

where $\boldsymbol{H}$ is a matrix representing the latent representation of the output from the hidden layers. This allows the model to process both time and spatial information effectively. Group normalization is employed in the model to normalize the outputs from an activation function. Unlike batch normalization, which presents poor performance when the batch size is small, group normalization performs well for small batches, which is the case in this paper. It divides channels into groups and computes the mean and variance of each group, which ensures effective normalization despite the small batch size [21]. The model extracts patterns from the sum of the input data and the time embeddings by passing them to group normalization layers and convolution layers,

$$H = Conv(GN(GELU(Conv(GN(H))))) \tag{7}$$

where $\boldsymbol{GN}$ and $\boldsymbol{Conv}$ represent group normalization layers and convolution layers, respectively. Additionally, there is a residual connection between the input and the output in this block. It is a powerful technique in neural networks to address the problems of vanishing gradients and improve training efficiency [22]. This connection allows the input data to bypass multiple layers and be added directly to the output, which facilitates the preservation of the original information. The residual connection in this block is expressed as,

$$OUT = Conv(IN) + H \tag{8}$$

where OUT represents the output of this block.

c) Down-Sampling and Up-Sampling: The size of the input data is divided by two in down sampling layers and the size is doubled in up sampling layers. The reason for performing down-sampling and up-sampling is to enable the model to learn to preserve the most significant information and subsequently use the information to restore the input.

2) Training: Hyperparameters are configured at the start of the training, and the training process is shown in [14]. The input data is treated as a square image for two reasons. First, the size of the image should match the number of features in the dataset. Second, the blocks in the neural network, including

convolution, down-sampling, and up-sampling, are designed to handle square images by default. In line two, the *DataLoader* collects samples from the dataset, and they are reshaped to the required input size. This ensures that the input size is compatible with the architecture of the neural network. In line 3, several timesteps are then sampled uniformly, and the corresponding noise is obtained in line four to corrupt the input images. In line five, the neural network predicts the noise added based on the corrupted images and the time step, and its performance is improved by minimizing the MSE loss between real noise and the predicted noise.

D. Data Generation

New traces are then generated by the trained diffusion models using the process shown in [14]. The generation starts by sampling Gaussian noise, as shown in line one. The diffusion models predict the amount of noise that should be removed from the sampled Gaussian noise in line eight. After iterating all timesteps, the generated traces x_0 are obtained, and the values within x_0 are scaled back to normal ranges using the method described in IV-B3.

V. Validation

A. Environment

The datasets used in this paper are collected from a *Social-Network* application, which is one of the end-to-end applications offered by the open-source benchmark *DeathStarBench* [23]. Each microservice is packaged into separate Docker containers, and *Jaeger* is integrated into the application for monitoring, which simplifies the process of data preparation.

The benchmark is deployed on three machines to simulate the operations of a social network application under different conditions, and the hardware configurations of three machines are shown in Table II. Two of the machines are used to execute tasks related to the social network application, while the third machine is dedicated to running *Jaeger* for network trace collection and storage. The container of each service is pinned to a specific machine to ensure that it is always executed on the same machine throughout the experiments. This setup ensures data consistency, as containers are always executed in the same environment, avoiding issues caused by being moved between different machines.

In order to introduce variations into traces collected from the benchmark, three levels of workload, low, medium, and high, are defined, which corresponds to 20%, 40%, and 60% resource usage. These levels are used to simulate different operational states of the benchmark, allowing for a more comprehensive analysis of performance under varying conditions.

B. Evaluation

1) Metrics: The evaluation metrics selected for this paper are accuracy, precision, recall, and F1 score. We use these metrics to determine whether a trace is normal or anomalous, as it is a binary classification task. Accuracy shows the overall performance of baseline models, and it is calculated as $\boldsymbol{accuracy} = (\boldsymbol{TP} + \boldsymbol{TN})/(\boldsymbol{TP} + \boldsymbol{TN} + \boldsymbol{FP} + \boldsymbol{FN})$

TABLE II
Hardware Configuration of Machines

No.	*CPU*	*Memory*	*Disk*
1	Intel(R) Xeon(R) CPU E5-2470 v2 @ 2.40GHz	32 GB	931.5 GB
2	Intel(R) Xeon(R) CPU E5-2440 v2 @ 1.90GHz	32 GB	931.5 GB
3	Intel(R) Xeon(R) CPU E3-1230 v5 @ 3.40GHz	16 GB	931.5 GB

where $\boldsymbol{TP}$ and $\boldsymbol{TN}$ represent true positive and true negative predictions, while false positive and false negative predictions are denoted as $\boldsymbol{FP}$ and $\boldsymbol{FN}$, respectively. However, accuracy is insufficient to reveal detailed insights as the training datasets in this paper are highly imbalanced. Therefore, precision and recall are also included to assess the models' ability to identify true normal traces, and they are expressed as, $\boldsymbol{precision} = \boldsymbol{TP}/(\boldsymbol{TP} + \boldsymbol{FP})$, and $\boldsymbol{recall} = \boldsymbol{TP}/(\boldsymbol{TP} + \boldsymbol{FN})$.

In addition, the F1 score is selected to provide more comprehensive results of the models performance in anomaly detection, as it combines both precision and recall into one metric. It is computed as $\boldsymbol{F1} = (2 \cdot \boldsymbol{precision} \cdot \boldsymbol{recall})/(\boldsymbol{precision} + \boldsymbol{recall})$

2) Baselines: Four anomaly detection methods, DeepSVDD [24], DevNet [25], DeepSAD [26], and TranAD [27], are selected as the baseline models for anomaly detection, and their configurations are shown in Table IV. There are both supervised and unsupervised learning algorithms among these methods, and the reason for this selection is that traces collected from real-world applications are unlabeled, making it challenging to apply supervised learning methods. Unsupervised learning algorithms detect anomalies without requiring labeled data, which is suitable for this case. On the other hand, when labeled datasets are available, supervised learning algorithms are expected to provide more accurate performance as they are trained with labeled data, which is beneficial for distinguishing normal and anomalous data.

In order to evaluate whether diffusion models are able to effectively mitigate the problems caused by imbalanced datasets, diffusion models are trained using the hyperparameters shown in Table III, and traces are generated using these models during the validation phase. After the generation, these traces are augmented into the original dataset at various percentages to form augmented datasets. 80% of data points in the augmented datasets are used to train baseline models and the rest of 20% data points are then used to determine if their performance in anomaly detection improved. The effectiveness of the augmentation is assessed by comparing the evaluation metrics before and after the augmentation.

C. Performance: 10% Anomalous Data

Normal traces are classified as the negative class, while anomalous traces are defined as the positive class. In order to evaluate the impact of augmenting traces generated by diffusion models, two imbalanced datasets, each containing 10% high or middle workload, are used to train the baseline

TABLE III
HYPERPARAMETER CONFIGURATION

Hyperparameter	*Value*
Number of Time Steps (T)	1000
Beta Start (β_1)	1e-4
Beta End (β_T)	0.02
Beta Schedule Method	Linear
Number of epochs (n_{epochs})	500
Optimizer	Adam
Learning Rate	1e-3
Batch Size	10
Time Embedding Dimension	128

TABLE IV
HYPERPARAMETER CONFIGURATION FOR BASELINE MODELS

Model	*Hyperparameter*
DeepSVDD	epochs=100, activation function=Tanh, layers=10, learning rate=1e-3
DevNet	epochs=120, activation function=Tanh, layers=10, learning rate=1e-3
DeepSAD	epochs=80, activation function=ReLU, layers=15, learning rate=1e-3
TranAD	epochs=50, sequence length=10, batch size=32, learning rate=1e-4

models. The corresponding results are presented in Figure 3 and Figure 4, respectively. In these figures, the X-axis represents the augmentation percentage, from 0% to 50%, while the Y-axis represents the evaluation metrics. The baseline models trained by original datasets are represented by 0% augmentations. The performance at this point shows the ability of the baseline models to identify normal and anomalous traces, and it would be compared with models trained with data augmentation.

The majority of training data points are normal traces, the baseline models should be able to accurately identify these data. Therefore, precision is not used in this case. The recall demonstrates that these models can identify anomalous traces to some extent, suggesting that they may classify some of anomalies as normal instances. Therefore, the accuracy is around 96% or higher in these results. In addition, the F1 score provides a harmonic mean of the precision and recall, but it is mainly affected by the recall in this case.

After augmentation, the performance of baseline models improves, and most of them achieve the highest performance when 25% of traces are generated by diffusion models. It is clear that the recall for all models increases, which suggests that the number of false negative predictions decreases, meaning that the models are capable of identifying anomalous traces more accurately. This increase in recall leads to an increase in the F1 score, which is heavily influenced by recall in this case. In addition, the influence of augmenting generated traces on the accuracy is small, as the anomalous traces are the minority class in the datasets, and the majority class dominates the accuracy. Therefore, augmenting generated traces results in an increase in recall and F1 score.

Although performance increases from 0% to 25% augmentation, it decreases when the augmentation percentage increases from 25% to 50%. The potential reason for this decrease is that augmentation may change the original data distribution. The baseline models learn patterns that do not accurately reflect the original distribution, which results in a decrease in the effectiveness of data augmentation and impacts the performance of the baseline models.

D. Sensitivity Analysis

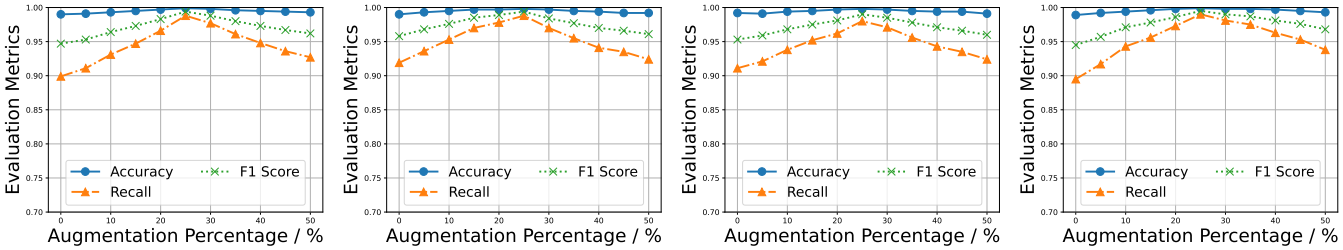
In order to further assess the robustness of the proposed approach, more experiments are performed in order to validate the effectiveness of augmenting generated traces in extreme cases. The results shown in Table V and Table VI provide the performance of baseline models in identifying high and middle workloads. The baseline models are trained using datasets with 5% or 1% anomalous data. For each baseline model in these cases, the table presents its performance at 0% augmentation and its highest performance after augmentation. In addition, the augmentation column specifies the percentage at which each baseline model obtains its peak performance after augmenting generated traces, providing insights into the optimal augmentation levels for enhancing anomaly detection in different workload scenarios.

As shown in figures 3 and 4, the performance for all models increases after augmentation. Most models achieve the highest performance when 25% of data is generated by diffusion models. Recall for all these cases increases, indicating that the number of false negative predictions reduces, and the models become more capable of identifying anomalous traces. This increase in recall results in a slight improvement in accuracy, as the total number of correct predictions increases, but it is still largely determined by normal traces, which is sufficiently accurate without augmentation. The impact of increased recall is also shown in the F1 score, which improves as the models become more effective at identifying anomalies. Thus, augmentation leads to an increase in recall and F1 score.

E. Discriminators

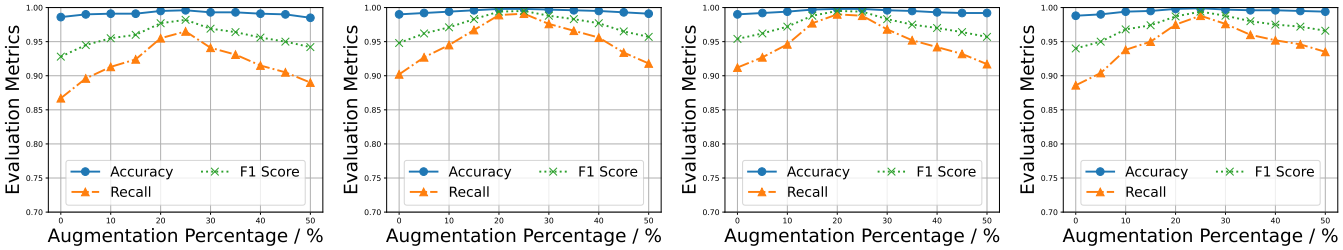
The traces generated by diffusion models are analyzed from another angle, i.e., where discriminators are trained using balanced datasets containing real and generated data, and the results are displayed in Figure 5. Unlike previous experiments that focus on using generated traces to augment original datasets to improve the performance of baseline models. The goal of this experiment is to evaluate if the baseline models are able to identify whether their inputs are real or generated. This provides insight into the quality of the generated traces.

To evaluate the performance of discriminators, the generated data is defined as positive class, and real data is defined as negative class. It is evident from Figure 5 that accuracy for all models is around 0.6, with even lower performance when DeepSVDD is used to distinguish low workload traces. This suggests that these models are not able to predict the inputs correctly. On the other hand, supervised learning models show better performance, achieving around 0.65, when they are predicting high workloads. However, their performance is still lower than the results for data augmentation, indicating that



(a) DeepSVDD (b) DevNet (c) DeepSAD (d) TranAD

Fig. 3. Performance: High vs. Low, 10% Anomalous Data



(a) DeepSVDD (b) DevNet (c) DeepSAD (d) TranAD

Fig. 4. Performance: High vs. Mid, 10% Anomalous Data

TABLE V
LOW VS. HIGH WORKLOADS: 5% AND 1% ANOMALOUS DATA

Anomal. (%)	Model	Augmentation %	Before Augmentation			After Augmentation		
			Accuracy	Recall	**F1 Score**	Accuracy	Recall	**F1 Score**
5%	DeepSVDD	25%	0.991	0.821	**0.901**	0.995	0.924	**0.960**
	DevNet	25%	0.992	0.837	**0.911**	0.996	0.943	**0.971**
	DeepSAD	25%	0.993	0.862	**0.925**	0.998	0.974	**0.987**
	TranAD	25%	0.993	0.865	**0.927**	0.997	0.969	**0.984**
1%	DeepSVDD	20%	0.996	0.733	**0.845**	0.998	0.865	**0.927**
	DevNet	25%	0.997	0.757	**0.862**	0.999	0.893	**0.943**
	DeepSAD	25%	0.997	0.778	**0.875**	0.999	0.889	**0.941**
	TranAD	20%	0.997	0.762	**0.865**	0.998	0.879	**0.936**

although supervised learning models outperform unsupervised learning models, they still have difficulties in identifying whether their input is real or generated.

Moreover, precision and recall for all unsupervised learning models are low, which suggests that they struggle to predict either generated or real traces accurately. When supervised learning models are used for low and high workloads, precision is high, but the recall is low, leading to a low F1 score. This indicates that these models may perform well in predicting real traces, whereas their performance is low for generated traces. This imbalance shows that these models perform poorly and cannot distinguish real and generated traces, which suggests that the traces generated by diffusion models are of high quality and are highly realistic.

VI. CONCLUSION

In this paper, we demonstrate the effectiveness of diffusion models for data augmentation of response time anomaly detectors. We find that the performance of baseline models improves by 20%, indicating that using the diffusion models to generate traces can enhance the model performance in anomaly detection for microservices. In addition, discriminators are trained using both real and generated traces. The results show that the generated traces and the real traces cannot be identified by these discriminators, which suggests that the traces generated by diffusion models are realistic and of high quality.

REFERENCES

[1] I. K. Aksakalli, *et al.*, "Deployment and communication patterns in microservice architectures: A systematic literature review," *JSS*, vol. 180, p. 111014, 2021.

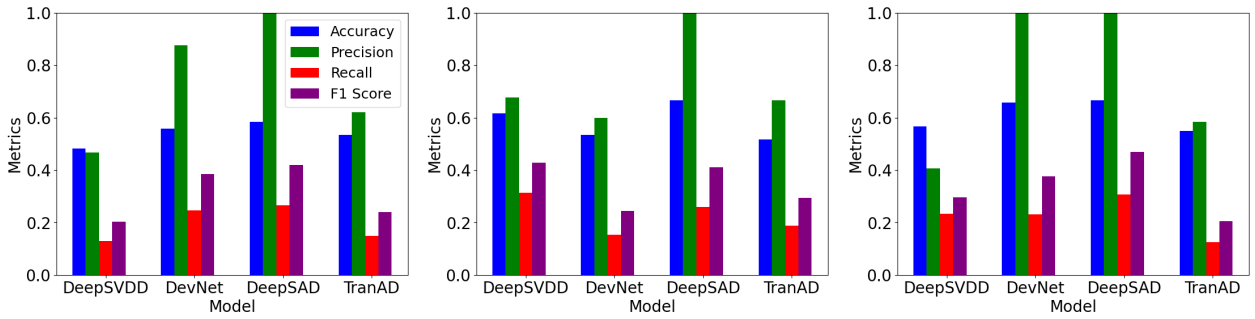
[2] M. Waseem, *et al.*, "Testing microservices architecture-based applications: A systematic mapping study," in *IEEE APSEC*. 2020, 119–128.

[3] X. Jiang and Z. Ge, "Data augmentation classifier for imbalanced fault classification," *IEEE T-ASE*, 18.(3):1206–1217, 2020.

[4] P. Liu, *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE ISSRE*, 2020, 48–58.

TABLE VI
LOW VS. MIDDLE WORKLOADS: 5% AND 1% ANOMALOUS DATA

Anomal. (%)	Model	Augmentation %	Before Augmentation			After Augmentation		
			Accuracy	Recall	**F1 Score**	Accuracy	Recall	**F1 Score**
5%	DeepSVDD	20%	0.987	0.795	**0.886**	0.994	0.889	**0.941**
	DevNet	30%	0.991	0.820	**0.901**	0.995	0.931	**0.964**
	DeepSAD	25%	0.993	0.857	**0.923**	0.997	0.962	**0.981**
	TranAD	20%	0.992	0.831	**0.908**	0.995	0.926	**0.962**
1%	DeepSVDD	25%	0.996	0.700	**0.823**	0.998	0.833	**0.909**
	DevNet	25%	0.996	0.727	**0.842**	0.998	0.875	**0.933**
	DeepSAD	20%	0.997	0.750	**0.857**	0.999	0.882	**0.937**
	TranAD	25%	0.996	0.744	**0.853**	0.998	0.851	**0.920**



(a) Low Workload (b) Middle Workload (c) High Workload

Fig. 5. Discriminator results

[5] J. Chen, *et al.*, "Tracegra: A trace-based anomaly detection for microservice using graph deep learning," *Computer Communications*, vol. 204, 109–117, 2023.

[6] H. Liu, *et al.*, "Uac-ad: Unsupervised adversarial contrastive learning for anomaly detection on multi-modal data in microservice systems," *IEEE TSC*, 2024.

[7] S. Nedelkoski, *et al.*, "Anomaly detection and classification using distributed tracing and deep learning," in *Proc. CCGRID*, 2019, 241–250.

[8] Y. Gan, *et al.*, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proc. ASPLOS*, 2019, 19–33.

[9] R. Natella, "Achieving representative faultloads in software fault injection." Ph.D. dissertation, University of Naples Federico II, Italy, 2011.

[10] R. Natella, *et al.*, "On fault representativeness of software fault injection," *IEEE TSE*, 39(1):80–96, 2012.

[11] N. V. Chawla, *et al.*, "Smote: synthetic minority oversampling technique," *JAIR*, vol. 16, 321–357, 2002.

[12] W. Dai, *et al.*, "Generative oversampling with a contrastive variational autoencoder," in *Proc. ICDM*, 2019, 101–109.

[13] N. Seliya, *et al.*, "A literature review on one-class classification and its potential applications in big data," *Journal of Big Data*, vol. 8, 1–31, 2021.

[14] J. Ho, *et al.*, "Denoising diffusion probabilistic models," *NeurIPS*, vol. 33, 6840–6851, 2020.

[15] A. Q. Nichol *et al.*, "Improved denoising diffusion probabilistic models," in *ICML*. PMLR, 2021, 8162–8171.

[16] R. Yousefpour Shahrivar, *et al.*, "Enhancing fetal anomaly detection in ultrasonography images: A review of machine learning-based approaches," *Biomimetics*, 8(7):519, 2023

[17] J. Yang, *et al.*, "Generalized out-of-distribution detection: A survey," *IJCV*, 132(12):5635–5662, 2024.

[18] D. C. Hoaglin, *et al.*, "Performance of some resistant rules for outlier labeling," *JASA*, 81(396): 991–999, 1986.

[19] O. Ronneberger, *et al.*, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI 2015*.

[20] A. Vaswani, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[21] Y. Wu and K. He, "Group normalization," in *Proc. ECCV*, 2018, 3–19.

[22] K. He, *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, 770–778.

[23] Y. Gan, *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proc. ASPLOS*, 2019, 3–18.

[24] L. Ruff, *et al.*, "Deep one-class classification," in *ICML*. PMLR, 2018, 4393–4402.

[25] G. Pang, *et al.*, "Deep anomaly detection with deviation networks," in *Proc. SIGKDD*, 2019, 353–362.

[26] L. Ruff, *et al.*, "Deep semi-supervised anomaly detection," *arXiv preprint arXiv:1906.02694*, 2019.

[27] S. Tuli, *et al.*, "TranAD: Deep transformer networks for anomaly detection in multivariate time series data," *Proc. VLDB*, 2022.