TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data

Shreshth Tuli
Imperial College London
London, UK
s.tuli20@imperial.ac.uk

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

Nicholas R. Jennings
Loughborough University
London, UK
n.r.jennings@lboro.ac.uk

ABSTRACT
Efficient anomaly detection and diagnosis in multivariate time-series data is of great importance for modern industrial applications. However, building a system that is able to quickly and accurately pinpoint anomalous observations is a challenging problem. This is due to the lack of anomaly labels, high data volatility and the demands of ultra-low inference times in modern applications. Despite the recent developments of deep learning approaches for anomaly detection, only a few of them can address all of these challenges. In this paper, we propose TranAD, a deep transformer network based anomaly detection and diagnosis model which uses attention-based sequence encoders to swiftly perform inference with the knowledge of the broader temporal trends in the data. TranAD uses focus score-based self-conditioning to enable robust multi-modal feature extraction and adversarial training to gain stability. Additionally, model-agnostic meta learning (MAML) allows us to train the model using limited data. Extensive empirical studies on six publicly available datasets demonstrate that TranAD can outperform state-of-the-art baseline methods in detection and diagnosis performance with data and time-efficient training. Specifically, TranAD increases F1 scores by up to 17%, reducing training times by up to 99% compared to the baselines.

PVLDB Reference Format:
doi:10.14778/3514061.3514067

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/imperial-qore/TranAD.

1 INTRODUCTION
Modern IT operations generate enormous amounts of high dimensional sensor data used for continuous monitoring and proper functioning of large-scale datasets. Traditionally, data mining experts have studied and highlighted data that do not follow usual trends and pinpoint anomalous observations. This is an active research discipline conforming to the expected trends, is an active research discipline referred to as anomaly detection in multivariate time series [11]. Many data-driven industries, including ones related to distributed computing, Internet of Things (IoT), robotics and urban resource management [4, 46] are now adopting machine learning based unsupervised methods for anomaly detection.

Challenges. The problem of anomaly detection is becoming increasingly challenging in large-scale databases due to the increasing data modality [18, 28, 54]. In particular, the increasing number of sensors and devices in contemporary IoT platforms with increasing data volatility creates the requirement for significant amounts of data for accurate inference. However, due to the rising federated learning paradigm with geographically distant clusters, synchronizing databases across devices is expensive, causing limited data availability for training [48, 57]. Further, next-generation applications need ultra-fast inference speeds for quick recovery and optimal Quality of Service (QoS) [6, 50]. Time-series databases are generated using several engineering artifacts (servers, robots, etc.) that interact with the environment, humans or other systems. As a result, the data often displays both stochastic and temporal trends [45]. It thus becomes crucial to distinguish outliers due to stochasticity and only pinpoint observations that do not adhere to the observed temporal trends. Moreover, the lack of labeled data and anomaly diversity makes the problem challenging as we cannot use supervised learning models, which have shown to be effective in other areas of data mining [12]. Finally, it is not only important to detect anomalies but also the root causes, i.e., the specific data sources leading to abnormal behavior [23]. This complicates the problem further as we need to perform multi-class prediction (whether there is an anomaly and from which source if so) [60].

Existing solutions. The above discussed challenges have led to the development of a myriad of unsupervised learning solutions for automated anomaly detection. Researchers have developed reconstruction-based methods that predominantly aim to encapsulate the temporal trends and predict the time-series data in an unsupervised fashion, then use the deviation of the prediction with the ground-truth data as anomaly scores. Based on various extreme value analysis methods, such approaches classify timestamps with high anomaly scores as abnormal [4, 10, 14, 20, 28, 29, 45, 60, 62]. The way prior works generate a predicted time-series from a given one varies from one work to another. Traditional approaches, like SAND [10], use clustering and statistical analysis to...
new insights. As noted above, recurrent models based on prior methods are not only slow and computationally expensive, but are also unable to model long-term trends effectively [4, 14, 62]. This is because, at each timestamp, a recurrent model needs to first perform inference for all previous timestamps before proceeding further. Recent developments of the transformer models allow single-shot inference with the complete input series using position encoding [51]. Using transformers allows much faster detection compared to recurrent methods by parallelizing inference on GPUs [19]. However, transformers also provide the benefit of being able to encode large sequences with accuracy and training/inference times nearly agnostic to the sequence length [51]. Thus, we use transformers to grow the temporal context information sent to an anomaly detector without significantly increasing the computational overheads.

Our contributions. This work uses various tools, including Transformer neural networks and model-agnostic meta learning, as building blocks. However, each of these different technologies cannot be directly used and need necessary adaptations to create a generalizable model for anomaly detection. Specifically, we propose a transformer-based anomaly detection model (TranAD), that uses self-conditioning and an adversarial training process. Its architecture makes it fast for training and testing while maintaining stability with large input sequences. Simple transformer-based encoder-decoder networks tend to miss anomalies if the deviation is too small, i.e., it is relatively close to normal data. One of our contributions is to show that this can be alleviated by an adversarial training procedure that can amplify reconstruction errors. Further, using self-conditioning for robust multi-modal feature extraction can help gain training stability and allow generalization [32]. This, with model-agnostic meta learning (MAML) helps keep optimum detection performance even with limited data [15], as we show later in the validation that methods with simple transformers underperform by over 11% compared to TranAD. We perform extensive empirical experiments on publicly available datasets to compare and analyze TranAD against the state-of-the-art methods. Our experiments show that TranAD is able to outperform baselines by increasing prediction scores by up to 17% while reducing training time overheads by up to 99%.

The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 outlines the working of the TranAD model for multivariate anomaly detection and diagnosis. A performance evaluation of the proposed method is shown in Section 4. Section 5 presents additional analysis. Finally, Section 6 concludes.

2 RELATED WORK

Time series anomaly detection is a long-studied problem in the VLDB community. The prior literature works on two types of time-series data: univariate and multivariate. For the former, various methods analyze and detect anomalies in time-series data with a single data source [34], while for the latter multiple time-series together [14, 45, 62].

Classical methods. Such methods for anomaly detection typically model the time-series distribution using various classical techniques like k-Mean clustering, Support Vector Machines (SVMs) or regression models [10, 28, 43, 52]. Other methods use wavelet theory or various signal transformation methods like Hilbert transform [25]. Other classes of methods use Principal Component Analysis (PCA), process regression or hidden Markov chains to model time-series data [41]. The GraphAn technique [9] converts the time-series inputs to graphs and uses graph distance metrics to detect outliers. Another technique, namely isolation forest, uses an ensemble of several isolation trees that recursively partition the feature space for outlier detection [5, 31]. Finally, classical methods use variants of Auto-Regressive Integrated Moving Average (ARIMA) to model and detect anomalous behaviour [56]. However, auto-regression based approaches are rarely used for anomaly detection in high-order multivariate time series due to their inability to efficiently capture volatile time-series [1]. Other methods like SAND [10], CPOD [47] and Elle [28] utilize clustering and database read-write history to detect outliers.

Time-series discord discovery is another recently proposed method for fault prediction [16, 37, 58, 59]. Time series discord refers to the most unusual time series subsequences, i.e., subsequences that are maximally different from all other subsequences in the same time series. A sub-class of methods uses matrix profiling or its variants for anomaly and motif discovery by detecting time series discords [16, 35, 63]. Many advances have been proposed to make matrix profiling techniques data and time efficient [21]. Other efforts aim to make matrix profiling applicable to diverse domains [64]. However, matrix profiling has many more uses than just anomaly detection and is considered to be slower than pure discord discovery algorithms [37]. A recent approach, MERLIN [37], uses a parameter-free version of time series discord discovery by iteratively comparing subsequences of varying length with their immediate neighbors. MERLIN is considered to be the state-of-the-art discord discovery approach with low overheads; hence, is regarded as one of the baselines in our experiments.

Deep Learning based methods. Most contemporary state-of-the-art techniques employ some form of deep neural networks. The LSTM-NDT [20] method relies on an LSTM based deep neural network model that uses the input sequence as training data and, for each input timestamp, forecasts data for the next timestamp. LSTMs are auto-regressive neural networks that learn order dependence in sequential data, where the prediction at each timestamp uses feedback from the output of the previous timestamp. This work also proposes a non-parametric dynamic error thresholding (NDT) strategy to set a threshold for anomaly labeling using moving averages of the error sequence. However, being a recurrent model, such models are slow to train in many cases with long input sequences.
Further, LSTMs are often inefficient in modeling long temporal patterns, especially when the data is noisy [62].

The DAGMM [65] method uses a deep autoencoding Gaussian mixture model for dimension reduction in the feature space and recurrent networks for temporal modeling. This work predicts an output using a mixture of Gaussians, where the parameters of each Gaussian are given by a deep neural model. The autoencoder compresses an input datapoint into a latent space, that is then used by a recurrent estimation network to predict the next datapoint. The decoupled training of both networks allows the model to be more robust; however, it is still slow and unable to explicitly utilize inter-modal correlations [14]. The Omnianomaly [45] uses a stochastic recurrent neural network (similar to an LSTM-Variational Autoencoder [39]) and a planar normalizing flow to generate reconstruction probabilities. It also proposes an adjusted Peak Over Threshold (POT) method for automated anomaly threshold selection that outperforms the previously used NDT approach. This work led to a significant performance leap compared to the prior art, but at the expense of high training times.

The Multi-Scale Convolutional Recursive Encoder-Decoder (MSCRED) [60] converts an input sequence window into a normalized two-dimensional image and then passes it through a ConvLSTM layer. This method is able to capture more complex inter-modal correlations and temporal information, however is unable to generalize to settings with insufficient training data. The MAD-GAN [29] uses an LSTM based GAN model to model the time-series distribution using generators. This work uses not only the prediction error, but also the discriminator loss in the anomaly scores. MTAD-GAT [62] uses a graph-attention network to model both feature and temporal correlations and pass it through a lightweight Gated-Recurrent-Unit (GRU) network that aids detection without severe overheads. Traditionally, attention operations perform input compression using convex combination where the weights are determined using neural networks. GRU is a simplified version of LSTM with a smaller parameter set and can be trained in limited data settings. The CAE-M [61] uses a convolutional autoencoding memory network, similar to MSCRED. It passes the time-series through a CNN with the output being processed by bidirectional LSTMs to capture long-term temporal trends. Such recurrent neural network-based models have been shown to have high computation costs and low scalability for high dimensional datasets [4].

More recent works such as USAD [4], GDN [14] and openGauss [30] do not use resource-hungry recurrent models, but only attention-based network architectures to improve training speeds. The USAD method uses an autoencoder with two decoders with an adversarial game-style training framework. This is one of the first works that focus on low overheads by using a simple autoencoder and can achieve a several-fold reduction in training times compared to the prior art. The Graph Deviation Network (GDN) approach learns a graph of relationships between data modes and uses attention-based forecasting and deviation scoring to output anomaly scores. The openGauss approach uses a tree-based LSTM that has lower memory and computational footprint and allows capturing temporal trends even with noisy data. However, due to the small window as an input and the use of simple or no recurrent models, the latest models are unable to capture long-term dependencies effectively.

The recently proposed HitAnomaly [19] method uses vanilla transformers as encoder-decoder networks, but is only applicable to natural-language log data and not appropriate for generic continuous time-series data as inputs. In our experiments, we compare TranAD against the state-of-the-art methods MERLIN, LSTM-NDT, DAGMM, OmniAnomaly, MSCRED, MAD-GAN, USAD, MTAD-GAT, CAE-M and GDN. These methods have shown superiority in anomaly detection and diagnosis, but complement one another in terms of performance across different time-series datasets. Out of these, only USAD aims to reduce training times, but does this to a limited extent. Just like reconstruction based prior work [4, 29, 45, 60, 61], we develop a TranAD model that learns broad level trends using training data to find anomalies in test data. We specifically improve anomaly detection and diagnosis performance with also reducing the training times in this work.

3 METHODOLOGY

3.1 Problem Formulation

We consider a multivariate time-series, which is a timestamped sequence of observations/datapoints of size $T$

$$T = \{x_1, \ldots, x_T\},$$

where each datapoint $x_t$ is collected at a specific timestamp $t$ and $x_t \in \mathbb{R}^m$, $\forall t$. Here, the univariate setting is a particular case where $m = 1$. We now define the two problems of anomaly detection and diagnosis.

**Anomaly Detection:** Given a training input time-series $\hat{T}$, for any unseen test time-series $\hat{T}$ of length $\hat{T}$ and same modality as the training series, we need to predict $\hat{Y} = \{y_1, \ldots, y_{\hat{T}}\}$, where we use $y_t \in \{0, 1\}$ to denote whether the datapoint at the $t$-th timestamp of the test set is anomalous (1 denotes an anomalous datapoint).

**Anomaly Diagnosis:** Given the above training and test time-series, we need to predict $\hat{Y} = \{y_1, \ldots, y_{\hat{T}}\}$, where $y_t \in \{0, 1\}^m$ to denote which of the modes of the datapoint at the $t$-th timestamp are anomalous.

3.2 Data Preprocessing

To make our model more robust, we normalize the data and convert it to time-series windows, both for training and testing. We normalize the time-series as:

$$x_t \leftarrow \frac{x_t - \min(T)}{\max(T) - \min(T) + \epsilon},$$

where $\min(T)$ and $\max(T)$ are the mode-wise minimum and maximum values in the training time-series. $\epsilon$ is a small constant vector to prevent zero-division. Knowing the ranges a-priori, we normalize the data to get it in the range $[0, 1]$.

To model the dependence of a data point $x_t$ at a timestamp $t$, we consider a local contextual window of length $K$ as

$$W_t = \{x_{t-K+1}, \ldots, x_t\}.$$

We use replication padding for $t < K$ and convert an input time series $T$ to a sequence of sliding windows $W = \{W_1, \ldots, W_T\}$. Replication padding, for each $t < K$, appends to the window $W_t$ a constant vector $(x_1, \ldots, x_1)$ of length $K - t$ to maintain the window length of $K$ for each $t$. Instead of using $T$ as training input, we use $W$ for model training and $\hat{W}$ (corresponding to $\hat{T}$) as the test
series. This is a common practice in prior work [4, 45] as it allows us to give a datapoint with its local context instead of a standalone vector, and hence is used in our model. We also consider the time slice until the current timestamp \( t \) of a series \( T \) and denote it as \( C_t \).

Now, instead of directly predicting the anomaly label \( y_t \) for each input window \( W_t \), we shall first predict an anomaly score \( s_t \) for this window. Using anomaly scores for the past input windows, we calculate a threshold value \( D \) above which we label the input window as anomalous, thus \( y_t = 1( s_t \geq D) \). To calculate the anomaly score \( s_t \), we reconstruct the input window as \( O_t \) and use the deviation between \( W_t \) and \( O_t \). For the sake of simplicity and without loss of generality, we shall use \( W, C, O \) and \( s \) for the rest of the discussion.

### 3.3 Transformer Model

Transformers are popular deep learning models that have been used in various natural language and vision processing tasks [51]. However, we use insightful refactoring of the transformer architecture for the task of anomaly detection in time-series data. Just like other encoder-decoder models, in a transformer, an input sequence undergoes several attention-based transformations. Figure 1 shows the architecture of the neural network used in TranAD. The encoder encodes the complete sequence until the current timestamp \( C \) with a focus score (more details later). The window encoder uses this to create an encoded representation of the input window \( W \), which is then passed to two decoders to create its reconstruction.

We now provide details on the working of TranAD. A multivariate sequence like \( W \) or \( C \) is transformed first into a matrix form with modality \( m \). We define scaled-dot product attention [51] of three matrices \( Q \) (query), \( K \) (key) and \( V \) (value):

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{m}} \right) V.
\]

Here, the softmax forms the convex combination weights for the values in \( V \), allowing us to compress the matrix \( V \) into a smaller representative embedding for simplified inference in the downstream neural network operations. Unlike traditional attention operation, the scaled-dot product attention scales the weights by a \( \sqrt{m} \) term to reduce the variance of the weights, facilitating stable training [51]. For input matrices \( Q, K \) and \( V \), we apply Multi-Head Self Attention [51] by first passing it through \( h \) (number of heads) feed-forward layers to get \( Q_1, K_i \) and \( V_i \) for \( i \in \{1, \ldots, h\} \), and then applying scaled-dot product attention as

\[
\text{MultiHeadAtt}(Q, K, V) = \text{Concat}(H_1, \ldots, H_h),
\]

where \( H_i = \text{Attention}(Q_i, K_i, V_i) \). (3)

Multi-Head Attention allows the model to jointly attend to information from different representation sub-spaces at different positions. In addition, we use position encoding of the input matrices as defined in [51].

As GAN models have been shown to perform well in characteristic tasks of whether an input is anomalous or not, we leverage a time-efficient GAN style adversarial training method. Our model consists of two transformer encoders and two decoders (Figure 1). We consider the model inference in two phases. We first take the \( W \) and \( C \) pair as an input and a focus score \( F \) (initially a zero matrix of the dimension of \( W \), more details in the next subsection). We broadcast \( F \) to match the dimension of \( W \), with appropriate zero-padding and concatenate the two. We then apply position encoding and obtain the input for the first encoder, say \( I_1 \). The first encoder performs the following operations

\[
I_1^2 = \text{LayerNorm}(I_1 + \text{MultiHeadAtt}(I_1, I_1, I_1)),
I_2^2 = \text{LayerNorm}(I_1^2 + \text{FeedForward}(I_1^2)).
\]

Here, \( \text{MultiHeadAtt}(I_1, I_1, I_1) \) denotes the multi-head self attention operation for the input matrix \( I_1 \) and \( + \) denotes matrix addition. The above operations generate attention weights using the input time-series windows and the complete sequence to capture temporal trends within the input sequences. These operations enable the model to infer over multiple batches of the time-series windows in parallel as the neural network, at each timestamp, does not depend on the output of a previous timestamp, significantly improving the training time of the proposed method. For the window encoder, we apply position encoding to the input window \( W \) to get \( I_2 \). We modify the self-attention in the window encoder to mask the data at subsequent positions. This is done to prevent the decoder from looking at the datapoints for future timestamp values at the time of training as all data \( W \) and \( C \) is given at once to allow parallel training. The window encoder performs the following operations

\[
I_2^1 = \text{Mask}(\text{MultiHeadAtt}(I_2, I_2, I_2)),
I_2^2 = \text{LayerNorm}(I_2 + I_2^1),
I_2^3 = \text{LayerNorm}(I_2^2 + \text{MultiHeadAtt}(I_2^2, I_2^2, I_2^2)).
\]

![Figure 1: The TranAD Model.](image_url)
The encoding of the complete sequence $I^2_t$ is used as value and keys by the window encoder for the attention operation using the encoded input window as the query matrix. The motivation behind the operations in (5) is similar to the one for (4); however, here we apply masking of the window input to hide the window sequences for future timestamps in the same input batch. As the complete input sequence up to the $t$-th timestamp is given to the model as an input; it allows the model to encapsulate and leverage a larger context compared to a bounded, limited one as in prior art [4, 45, 62]. Finally, we use two identical decoders which perform the operation

$$O_t = \text{Sigmoid}(\text{FeedForward}(I^2_t)),$$

where $i \in \{1, 2\}$ for the first and second decoder respectively. The Sigmoid activation is used to generate an output in the range $[0, 1]$, to match the normalized input window. Thus, the TranAD model takes the inputs $C$ and $W$ to generate two outputs $O_1$ and $O_2$.

### 3.4 Offline Two-Phase Adversarial Training

We now describe the adversarial training process and the two-phase inference approach in the TranAD model, summarized in Algorithm 1.

**Phase 1 - Input Reconstruction.** The Transformer model enables us to predict the reconstruction of each input time-series window. It does this by acting as an encoder-decoder network at each timestamp. However, traditional encoder-decoder models often are unable to capture short-term trends and tend to miss anomalies if the deviations are too small [29]. To tackle this challenge, we develop an auto-regressive inference style that predicts the reconstructed window in two-phases. In the first phase, the model aims to generate an approximate reconstruction of the input window. The deviation from this inference, referred to as the focus score mentioned previously, facilitates the attention network inside the Transformer Encoder to extract temporal trends, focusing on the sub-sequences where the deviations are high. Thus, the output of the second phase is conditioned on the deviations generated from the first phase. Thus, in the first stage, the encoders convert the input window $W \in \mathbb{R}^{K \times m}$ (with focus score $F = [0]_{K \times m}$) to a compressed latent representation $I^2_1$ using context-based attention as in a common transformer model. This compressed representation is then converted to generate outputs $O_1$ and $O_2$ via Eq. (6).

**Phase 2 - Focused Input Reconstruction.** In the second phase, we use the reconstruction loss for the first decoder as a focus score. Having the focus matrix for the second phase $F = \hat{O}_1$, we rerun model inference to obtain the output of the second decoder as $\hat{O}_2$.

The focus score generated in the first phase indicates the deviations of the reconstructed output from the given input. This acts as a prior to modify the attention weights in the second phase and gives higher neural network activation to specific input sub-sequences to extract short-term temporal trends. We refer to this approach as "self-conditioning" in the rest of the paper. This two-phase auto-regressive inference style has a three-fold benefit. First, it amplifies the deviations, as the reconstruction error acts as an activation in the attention part of the Encoder in Figure 1, to generate an anomaly score, simplifying the fault-labeling task (discussed in Section 3.5). Second, it prevents false positives by capturing short-term temporal trends in the Window Encoder in Figure 1. Third, the adversarial style training is known to improve generalizability and make the model robust to diverse input sequences [4].

**Evolving Training Objective.** The above-described model is bound to suffer from similar challenges as in other adversarial training frameworks. One of the critical challenges is maintaining training stability. To tackle this, we design an adversarial training procedure that uses outputs from two separate decoders (Decoders 1 and 2 in Figure 1). Initially, both decoders aim to independently reconstruct the input time-series window. As in [45] and [39], we define the reconstruction loss for each decoder using the L2-norm using the outputs of the first phase:

$$L_1 = \|O_1 - W\|_2,$$

$$L_2 = \|O_2 - W\|_2.$$  

(7)

We now introduce the adversarial loss that uses outputs of the second phase. Here, the second decoder aims to distinguish between the input window and the candidate reconstruction generated by the first decoder in phase 1 (using the focus scores) by maximizing the difference $\|\hat{O}_2 - W\|_2$. On the other hand, the first decoder aims to fool the second decoder by aiming to create a degenerate focus score (a zero vector) by perfectly reconstructing the input (i.e., $O_1 = W$). This pushes the decoder 2, in this phase, to generate the same output as $O_2$ which it aims to match the input in phase 1. This means the training objective is

$$\min_{\text{Decoder1}} \max_{\text{Decoder2}} \|\hat{O}_2 - W\|_2.$$  

(8)

Thus, the objective of the first decoder is to minimize the reconstruction error of this self-conditioned output, whereas the objective of the second one is to maximize the same. We realize this by using the loss as:

$$L_1 = +\|\hat{O}_2 - W\|_2,$$

$$L_2 = -\|\hat{O}_2 - W\|_2.$$  

(9)

Now that we have loss functions for both phases, we need to determine the cumulative loss for each decoder. We thus use an evolutionary loss function that combines the reconstruction and

---

**Algorithm 1** The TranAD training algorithm

**Require:**
- Encoder $E$, Decoders $D_1$ and $D_2$
- Dataset used for training $W$
- Evolutionary hyperparameter $\epsilon$
- Iteration limit $N$

1. Initialize weights $E, D_1, D_2$
2. $n \leftarrow 0$
3. do
4. for ($t = 1$ to $T$)
5. $O_1, O_2 \leftarrow D_1(E(W_t, \hat{O})), D_2(E(W_t, \hat{O}))$
6. $\hat{O}_2 \leftarrow D_2(E(W_t, ||O_1 - W||_2))$
7. $L_1 = \epsilon^{-n}||O_1 - W||_2 + (1 - \epsilon^{-n})||\hat{O}_2 - W||_2$
8. $L_2 = \epsilon^{-n}||O_2 - W||_2 - (1 - \epsilon^{-n})||\hat{O}_2 - W||_2$
9. Update weights of $E, D_1, D_2$ using $L_1, L_2$
10. $n \leftarrow n + 1$
11. Meta-Learn weights $E, D_1, D_2$ using a random batch
12. while $n < N$

---

1205
where $\alpha$ window. With poor reconstructions, the focus scores used in the construction loss is high. This is to ensure stable training when the weight given to the re-

updated weights $\theta$ of each epoch we perform meta-learning step as

$\text{Meta Learning.}$ Finally, our training loop uses model-agnostic meta learning (MAML), a few-shot learning model for fast adaptation of neural networks [15]. This helps our TranAD model learn temporal trends in the input training time-series with limited data. In each training epoch, a gradient update for neural network weights (without loss in generality assume $\theta$) can be simply written

$$\theta' \leftarrow \theta - \alpha \nabla_{\theta} L(f(\theta)), \quad (11)$$

where $\alpha, f(\cdot)$ and $L(\cdot)$ are learning rate, abstract representation of the neural network and loss function respectively. Now, at the end of each epoch we perform meta-learning step as

$$\theta \leftarrow \theta - \beta \nabla_{\theta} L(f(\theta')), \quad (12)$$

The meta-optimization is performed with a meta step-size $\beta$, over the model weights $\theta$ where the objective is evaluated using the updated weights $\theta'$. Prior work has shown that this allows models to be trained quickly with limited data [15]. We encapsulate this in a single line in Algorithm 1 (line 11).
4 EXPERIMENTS

We compare TranAD with state-of-the-art models for multivariate time-series anomaly detection, including MERLIN [37], LSTM-NDT [20] (with autoencoder implementation from openGauss [30]), DAGMM [65], OmniAnomaly [45], MSCRED [60], MAD-GAN [29], USAD [4], MTAD-GAT [62], CAE-M [61] and GDN [14] (with graph embedding implementation from GraphAn [9]). For more details refer Section 2. We also tested the Isolation Forest method, but due to its low F1 scores, do not include the corresponding results in our discussion. Other classical methods have been omitted as deep-learning based approaches have already been shown to outperform them in prior work [4, 14, 62]. We use hyperparameters of the baseline models as presented in their respective papers. We train all models using PyTorch-1.7.1 library.

We use the AdamW [27] optimizer to train our model with an initial learning rate of 0.01 (meta learning rate 0.02) and step-scheduler with step size of 0.5 [42]. We use the following hyperparameter values determined using grid-search.

- Window size = 10.
- Number of layers in transformer encoders = 1
- Number of layers in feed-forward unit of encoders = 2
- Hidden units in encoder layers = 64
- Dropout in encoders = 0.1

The effect of window size on anomaly detection performance is analyzed in Section 5. We choose hyperparameters other than the window size using grid search. For POT parameters, coefficient \(10^{-4}\) for all data sets, low quantile is 0.07 for SMAP, 0.01 for MSL, and 0.001 for others. These were selected as per the implementation of the OmniAnomaly baseline [45]. The only dataset-specific hyperparameter is the number of heads in multi-head attention, which was kept to be the same as the dimension size of the dataset. Other assignments for this hyperparameter give similar broad-level trends.

To train TranAD, we divide the training time-series into 80% training data and 20% validation data. To avoid model over-fitting, we use early-stopping criteria to train TranAD, i.e., we stop the training process once the validation accuracy starts to decrease.

4.1 Datasets

We use seven publicly available datasets in our experiments. We summarize their characteristics in Table 1. The values in parenthesis are the number of sequences in the dataset repository and we report average scores across all sequences in a dataset. For instance, the SMAP dataset has 55 traces with 25 dimensions each. While we share some of the concerns expressed in [55] about the lack of quality benchmark datasets for time series anomaly detection, we use these commonly-used benchmark datasets here to enable direct comparison of our approach to competing methods.

(1) Numenta Anomaly Benchmark (NAB): is a dataset of multiple real-world data traces, including readings from temperature sensors, CPU utilization of cloud machines, service request latencies and taxi demands in New York city [2]. However, this dataset is known to have sequences with incorrect anomaly labels [55] such as the nyc–taxi trace [37], which we exclude in our experiments.

(2) HexagonML (UCR) dataset: is a dataset of multiple univariate time series (included just for completeness) that was used in KDD 2021 cup [13, 26]. We include only the datasets obtained from natural sources (the InternalBleeding and ECG datasets) and ignore the synthetic sequences.

(3) MIT-BIH Supraventricular Arrhythmia Database (MBA): is a collection of electrocardiogram recordings from four patients, containing multiple instances of two different kinds of anomalies (either supraventricular contractions or premature heartbeats) [17, 36]. This is a popular large-scale dataset in the data management community [8, 10].
We eschew comparisons on the Yahoo dataset that has been claimed to suffer from mislabeling and run-to-failure bias [55].

4.2 Evaluation Metrics

4.2.1 Anomaly Detection. We use precision, recall, area under the receiver operating characteristic curve (ROC/AUC) and F1 score to evaluate the detection performance of all models [14, 62]. We also measure the AUC and F1 scores by training all models with 20% of the training data (again using the 80:20 split for validation data and the rest as the test set), and call these AUC* and F1* respectively, to measure the performance of the models with limited data. We train on the five sets of 20% training data and report average results for statistical significance.

4.2.2 Anomaly Diagnosis. We use commonly used metrics to measure the diagnosis performance of all models [62]. HitRate@P% is the measure of how many ground truth dimensions have been included in the top candidates predicted by the model [45]. P% is the percentage of the ground truth dimensions for each timestamp, which we use to consider the top predicted candidates. For instance, if at timestamp t, if 2 dimensions are labeled anomalous in the ground truth, HitRate@100% would consider top 2 dimensions and HitRate@150% would consider 3 dimensions (100 and 150 are chosen based on prior work [62]). We also measure the Normalized Discounted Cumulative Gain (NDCG) [24]. NDCG@P% considers the same number of top predicted candidates as HitRate@P%.

4.3 Results

Anomaly Detection. Tables 2 and 3 provide the precision, recall, AUC, F1, AUC* and F1* scores for TranAD and baseline models for all datasets. On average, the F1 score of the TranAD model is 0.8102 and F1* is 0.8012. TranAD outperforms the baselines (in terms of F1 score) for all datasets except MSL when we consider the complete dataset for model training. TranAD also outperforms baselines for all datasets except the WADI dataset with 20% of the dataset used for training (F1* score). For MSL, the GDN model has the highest F1 score (0.9591) and for the WADI dataset, OmniAnomaly has the highest F1* score (0.1017). Similarly, TranAD outperforms baselines in terms of AUC scores for all datasets except MSDS, where GDN has the highest AUC (0.9105). All models perform relatively poorly on WADI due to its large-scale in terms of sequence lengths and data modality. Specifically, TranAD achieves improvement of up to 17.06% in F1 score, 14.64% in F1* score, 11.69% in AUC and 11.06% in AUC* scores over the state-of-the-art baseline models.

The MERLIN baseline is a parameter free approach that does not require any training data; hence, we report F1* and AUC* as F1 and AUC scores, respectively. MERLIN performs relatively well only on the univariate datasets, i.e. NAB and UCR, and is unable to scale effectively to multivariate data in our traces. The baseline method LSTM-NDT has a good performance on MSL and SMD, but performs poorly on other datasets. This is due to its sensitivity to different scenarios and poor efficiency of the NDT thresholding method [62]. The POT technique used in TranAD and other models like OmniAnomaly helps set more accurate threshold values by also considering the localized peak values in the data sequence. DAGMM model performs very well for short datasets like UCR, MBA and SWaT, but its scores drop significantly for other datasets with longer sequences. This is because it does not map the temporal information explicitly as it does not use sequence windows but only a single GRU model. The window encoder in TranAD, with the encoding of the complete sequence as a self-condition, allows it to perform better even with long high-dimensional sequences. The OmniAnomaly, CAE-M and MSCRED models use sequential observations as input, allowing these methods to retain the temporal information. Such methods perform reconstruction regardless of anomalous data, which prevents them from detecting anomalies close to the normal trends [4]. TranAD tackles this by using adversarial training to amplify errors. Hence, in datasets like SWaT, where anomalous data is not very far from normal data, it can detect even mild anomalies.

Recent models such as USAD, MTAD-GAT and GDN use attention mechanisms to focus on specific modes of the data. Moreover, these models try to capture the long-term trends by adjusting the weights of their neural network and only use a local window as an input for reconstruction. GDN has slightly higher scores for MSL and MSDS datasets than TranAD due to the scalable graph-based inference over the inter-dimensional data correlations [14]. TranAD does this using self-attention and performs better than GDN overall across all datasets. The limitation of seeing only a local contextual window prevents methods such as USAD and MTAD-GAT from classifying long-term anomalies (like in SMD or WADI). However, self-conditioning on an embedding of the complete trace with position encoding aids temporal attention, thanks to the transformer.
architecture in TranAD. This allows TranAD to capture long-term trends more effectively. Further, due to the meta-learning, TranAD also outperforms baselines with limited training data except for OmniAnomaly on the WADI dataset, indicating its high efficacy with limited data. OmniAnomaly performs best among all methods on the WADI dataset due to high noise in this dataset and dedicated stochasticity modeling in OmniAnomaly [45]. TranAD is slightly behind this method in terms of F1* and AUC*; however, it outperforms it when compared across all datasets and also when given the complete WADI dataset.

We perform critical difference analysis to assess the significance of the differences among the performance of the models. Figure 4 depicts the critical difference diagrams for the F1 and AUC scores based on the Wilcoxon pair-wised signed-rank test (with $\alpha = 0.05$) after rejecting the null hypothesis using the Friedman test on all datasets [22]. TranAD achieves the best rank across all models with a significant statistical difference.

**Anomaly Diagnosis.** The anomaly diagnosis results in Table 4 where H and N correspond to HitRate and NDCG (with complete data used for model testing). We only present results on the multi-variate SMD and MSDS datasets for the sake of brevity (TranAD yields better scores for others as well). We also ignore models that do not explicitly output anomaly class outputs for each dimension individually. Multi-head attention in TranAD allows it to attend to multiple modes simultaneously, making it more suitable for more inter-correlated anomalies. This is observed and explained by datasets like MSDS (distributed systems) where anomalous behavior in one mode can lead to a chain of events causing anomalies.
Table 3: Performance comparison of TranAD with baseline methods with 20% of the training dataset. AUC*: AUC with 20% training data, F1*: F1 score with 20% training data. The best F1* and AUC* scores are highlighted in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>NAB</th>
<th>UCR</th>
<th>MBA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUC*</td>
<td>F1*</td>
<td>AUC*</td>
</tr>
<tr>
<td>MERLIN</td>
<td>0.8029</td>
<td>0.7619</td>
<td>0.8984</td>
</tr>
<tr>
<td>LSTM-NDT</td>
<td>0.8013</td>
<td>0.6212</td>
<td>0.8912</td>
</tr>
<tr>
<td>DAGMM</td>
<td>0.7827</td>
<td>0.6125</td>
<td>0.9728</td>
</tr>
<tr>
<td>OmniAnomaly</td>
<td>0.8129</td>
<td>0.6713</td>
<td>0.9728</td>
</tr>
<tr>
<td>MSCRED</td>
<td>0.8299</td>
<td>0.7013</td>
<td>0.9637</td>
</tr>
<tr>
<td>MAD-GAN</td>
<td>0.8194</td>
<td>0.7019</td>
<td>0.9959</td>
</tr>
<tr>
<td>USAD</td>
<td>0.7267</td>
<td>0.6781</td>
<td>0.9967</td>
</tr>
<tr>
<td>MTAD-GAT</td>
<td>0.6956</td>
<td>0.7013</td>
<td>0.9794</td>
</tr>
<tr>
<td>CAE-M</td>
<td>0.7312</td>
<td>0.7126</td>
<td>0.9926</td>
</tr>
<tr>
<td>GDN</td>
<td>0.9217</td>
<td>0.8421</td>
<td>0.9989</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SMAP</th>
<th>MSL</th>
<th>SWAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUC*</td>
<td>F1*</td>
<td>AUC*</td>
</tr>
<tr>
<td>MERLIN</td>
<td>0.7426</td>
<td>0.2725</td>
<td>0.6281</td>
</tr>
<tr>
<td>LSTM-NDT</td>
<td>0.7007</td>
<td>0.5418</td>
<td>0.9520</td>
</tr>
<tr>
<td>DAGMM</td>
<td>0.9879</td>
<td>0.8131</td>
<td>0.9703</td>
</tr>
<tr>
<td>OmniAnomaly</td>
<td>0.9811</td>
<td>0.8050</td>
<td>0.9797</td>
</tr>
<tr>
<td>MSCRED</td>
<td>0.9877</td>
<td>0.8468</td>
<td>0.9649</td>
</tr>
<tr>
<td>MAD-GAN</td>
<td>0.9883</td>
<td>0.8379</td>
<td>0.9649</td>
</tr>
<tr>
<td>USAD</td>
<td>0.9814</td>
<td>0.8225</td>
<td>0.9782</td>
</tr>
<tr>
<td>MTAD-GAT</td>
<td>0.9888</td>
<td>0.8411</td>
<td>0.9414</td>
</tr>
<tr>
<td>CAE-M</td>
<td>0.9885</td>
<td>0.8889</td>
<td>0.9857</td>
</tr>
<tr>
<td>GDN</td>
<td>0.5912</td>
<td>0.1174</td>
<td>0.7158</td>
</tr>
<tr>
<td>TranAD</td>
<td>0.9076</td>
<td>0.6212</td>
<td>0.9637</td>
</tr>
</tbody>
</table>

in other modes (see Figure 5). TranAD is able to leverage the complete trace information with the local window to aid in pinpointing anomalous behavior to specific modes. The table demonstrates that TranAD is able to detect 46.3% − 75.3% root causes for anomalies.

Compared to the baseline methods, TranAD is able to improve diagnosis score by up to 6% for SMD and 30% for MSDS. The average improvement in diagnosis scores is 4.25%.

5 ANALYSES

5.1 Ablation Analysis
To study the relative importance of each component of the model, we exclude every major one and observe how it affects the performance in terms of the F1 scores for each dataset. First, we consider the TranAD model without the transformer-based encoder-decoder architecture but instead with a feed-forward network. Second, we consider the model without the self-conditioning, i.e., we fix the focus score $F = \emptyset$ in phase 2. Third, we study the model without...
Table 4: Diagnosis Performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>SMD</th>
<th>MSDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H@100%</td>
<td>H@150%</td>
</tr>
<tr>
<td>MERLIN</td>
<td>0.5907</td>
<td>0.6177</td>
</tr>
<tr>
<td>LSTM-NDT</td>
<td>0.3808</td>
<td>0.5225</td>
</tr>
<tr>
<td>DAGMM</td>
<td>0.4927</td>
<td>0.6091</td>
</tr>
<tr>
<td>OmniAnomaly</td>
<td>0.4567</td>
<td>0.5676</td>
</tr>
<tr>
<td>MSCRED</td>
<td>0.4272</td>
<td>0.5180</td>
</tr>
<tr>
<td>MAD-GAN</td>
<td>0.4630</td>
<td>0.5785</td>
</tr>
<tr>
<td>USAD</td>
<td>0.4925</td>
<td>0.6055</td>
</tr>
<tr>
<td>MTAD-GAT</td>
<td>0.3497</td>
<td>0.4777</td>
</tr>
<tr>
<td>CAE-M</td>
<td>0.4707</td>
<td>0.5878</td>
</tr>
<tr>
<td>GDN</td>
<td>0.3143</td>
<td>0.4386</td>
</tr>
<tr>
<td>TranAD</td>
<td>0.4981</td>
<td>0.6401</td>
</tr>
</tbody>
</table>

Table 5: Comparison of training times in seconds per epoch.

<table>
<thead>
<tr>
<th>Method</th>
<th>NAB</th>
<th>UCR</th>
<th>MBA</th>
<th>SMA</th>
<th>MSL</th>
<th>SWaT</th>
<th>WADI</th>
<th>SMD</th>
<th>MSDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERLIN</td>
<td>3.28</td>
<td>4.09</td>
<td>20.19</td>
<td>6.89</td>
<td>5.12</td>
<td>10.12</td>
<td>132.69</td>
<td>72.32</td>
<td>42.22</td>
</tr>
<tr>
<td>DAGMM</td>
<td>25.38</td>
<td>20.78</td>
<td>74.62</td>
<td>19.05</td>
<td>16.41</td>
<td>18.51</td>
<td>178.17</td>
<td>204.36</td>
<td>187.54</td>
</tr>
<tr>
<td>OmniAnomaly</td>
<td>38.27</td>
<td>38.27</td>
<td>109.86</td>
<td>27.05</td>
<td>21.31</td>
<td>28.39</td>
<td>212.99</td>
<td>276.97</td>
<td>278.10</td>
</tr>
<tr>
<td>MSCRED</td>
<td>258.86</td>
<td>262.45</td>
<td>592.13</td>
<td>16.13</td>
<td>33.47</td>
<td>183.67</td>
<td>1349.05</td>
<td>237.66</td>
<td>109.63</td>
</tr>
<tr>
<td>MAD-GAN</td>
<td>39.80</td>
<td>25.71</td>
<td>160.29</td>
<td>29.49</td>
<td>26.27</td>
<td>27.79</td>
<td>293.60</td>
<td>314.82</td>
<td>285.25</td>
</tr>
<tr>
<td>USAD</td>
<td>145.00</td>
<td>97.12</td>
<td>233.08</td>
<td>1015.03</td>
<td>1287.42</td>
<td>103.92</td>
<td>9812.13</td>
<td>6564.11</td>
<td>1304.09</td>
</tr>
<tr>
<td>MTAD-GAT</td>
<td>258.86</td>
<td>262.45</td>
<td>592.13</td>
<td>16.13</td>
<td>33.47</td>
<td>183.67</td>
<td>1349.05</td>
<td>237.66</td>
<td>109.63</td>
</tr>
<tr>
<td>CAE-M</td>
<td>39.80</td>
<td>25.71</td>
<td>160.29</td>
<td>29.49</td>
<td>26.27</td>
<td>27.79</td>
<td>293.60</td>
<td>314.82</td>
<td>285.25</td>
</tr>
<tr>
<td>GDN</td>
<td>83.84</td>
<td>58.78</td>
<td>159.01</td>
<td>62.33</td>
<td>96.71</td>
<td>59.40</td>
<td>4063.05</td>
<td>809.94</td>
<td>585.34</td>
</tr>
<tr>
<td>TranAD</td>
<td>1.25</td>
<td>0.84</td>
<td>4.08</td>
<td>3.55</td>
<td>5.27</td>
<td>0.87</td>
<td>115.91</td>
<td>43.56</td>
<td>17.15</td>
</tr>
</tbody>
</table>

Table 6: Ablation Study - F1 and F1* scores for TranAD and its ablated versions.

<table>
<thead>
<tr>
<th>Method</th>
<th>NAB</th>
<th>UCR</th>
<th>MBA</th>
<th>SMA</th>
<th>MSL</th>
<th>SWaT</th>
<th>WADI</th>
<th>SMD</th>
<th>MSDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TranAD</td>
<td>0.9364</td>
<td>0.8421</td>
<td>0.9694</td>
<td>0.9399</td>
<td>0.9780</td>
<td>0.9617</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o transformer</td>
<td>0.8580</td>
<td>0.8019</td>
<td>0.8466</td>
<td>0.5395</td>
<td>0.9749</td>
<td>0.9761</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o self-condition</td>
<td>0.8887</td>
<td>0.8102</td>
<td>0.9191</td>
<td>0.9828</td>
<td>0.9770</td>
<td>0.9617</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o adversarial training</td>
<td>0.9012</td>
<td>0.8102</td>
<td>0.9634</td>
<td>0.9289</td>
<td>0.9752</td>
<td>0.9592</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o MAML</td>
<td>0.9068</td>
<td>0.8210</td>
<td>0.9689</td>
<td>0.9304</td>
<td>0.9756</td>
<td>0.9617</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SMA</th>
<th>MSL</th>
<th>SWaT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TranAD</td>
<td>0.8915</td>
<td>0.8889</td>
<td>0.9494</td>
</tr>
<tr>
<td>w/o transformer</td>
<td>0.8643</td>
<td>0.8147</td>
<td>0.9137</td>
</tr>
<tr>
<td>w/o self-condition</td>
<td>0.8894</td>
<td>0.8153</td>
<td>0.9175</td>
</tr>
<tr>
<td>w/o adversarial training</td>
<td>0.8906</td>
<td>0.8476</td>
<td>0.9455</td>
</tr>
<tr>
<td>w/o MAML</td>
<td>0.8915</td>
<td>0.8899</td>
<td>0.9466</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>WADI</th>
<th>SMD</th>
<th>MSDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TranAD</td>
<td>0.4951</td>
<td>0.0649</td>
<td>0.5605</td>
</tr>
<tr>
<td>w/o transformer</td>
<td>0.2181</td>
<td>0.0337</td>
<td>0.9071</td>
</tr>
<tr>
<td>w/o self-condition</td>
<td>0.3620</td>
<td>0.0631</td>
<td>0.9502</td>
</tr>
<tr>
<td>w/o adversarial training</td>
<td>0.3820</td>
<td>0.0621</td>
<td>0.9177</td>
</tr>
<tr>
<td>w/o MAML</td>
<td>0.4815</td>
<td>0.0553</td>
<td>0.9433</td>
</tr>
</tbody>
</table>

5.2 Overhead Analysis

Table 5 shows the average training times for all models on every dataset in seconds per epoch. For comparison, we report the training time for MERLIN as the time it takes to discover discords in

the adversarial loss, i.e., a single-phase inference and only the reconstruction loss for model training. Finally, we consider the model without meta-learning. The results are summarized in Table 6 and provide the following findings:

- Replacing the transformer-based encoder-decoder has the highest performance drop of nearly 11% in terms of the F1 score. This drop is more pronounced for the WADI dataset (56%), demonstrating the need for the attention-based transformer for large-scale datasets.

- When we remove the self-conditioning, the average drop in F1 scores is 6%, which shows that the focus score aids prediction performance.

- Removing the two-phase adversarial training mainly affects SMD and WADI datasets as these traces have a large proportion of mild anomalies and the adversarial loss helps amplify the anomaly scores. The average drop in F1 score, in this case, is 5%.

- Not having the meta-learning in the model has little effect to the F1 scores (≈1%); however, it leads to a nearly 12% drop in F1*. 

The following sections delve deeper into the results.
the test data. The training time of TranAD is 75% – 99% lower than those of the baselines. This demonstrates the advantage of having a transformer with positional encoding to push the complete sequence as an input instead of inferring over sequential windows.

5.3 Sensitivity Analysis

Sensitivity to the training set size. Figure 6 shows the variation of the F1 and AUC scores of all models averaged for all datasets, and the training time with the ratio of the training data used for model training, ranging from 20% to 100%. We do not report sensitivity results on MERLIN as it does not use training data. Other deep learning based reconstruction models are given the same randomly sampled subsequence of 20% to 100% size as that of the training data. We report 90% confidence bounds in Figure 6. Clearly, as dataset size increases, the prediction performance improves and the training time increases. We observe that for every ratio, the TranAD model has a higher F1 score and a lower training time.

Sensitivity to the window size. We also show the performance of the TranAD model and its variants with different window sizes in Figure 7. The window size has an impact both on the anomaly detection scores and training times. TranAD can detect anomalies faster when we have smaller windows since the inference time is lower for smaller inputs. If the window is too small, it does not represent the local contextual information well. However, if the window is too large, short anomalies may be hidden in a large number of datapoints in such a window (see the drop in F1 score for some models). A window size of 10 gives a reasonable balance between the F1 score and training times and hence is used in our experiments.

6 CONCLUSIONS

We present a transformer based anomaly detection model (TranAD) that can detect and diagnose anomalies for multivariate time-series data. The transformer based encoder-decoder allows quick model training and high detection performance for a variety of datasets considered in this work. TranAD leverages self-conditioning and adversarial training to amplify errors and gain training stability. Moreover, meta-learning allows it to be able to identify data trends even with limited data. Specifically, TranAD achieves an improvement of 17% and 11% for F1 score on complete and limited training data, respectively. It is also able to correctly identify root causes for up to 75% of the detected anomalies, higher than the state-of-the-art models. It is able to achieve this with up to 99% lower training times compared to the baseline methods. This makes TranAD an ideal choice for modern industrial systems where accurate and quick anomaly predictions are required.

For the future, we propose to extend the method with other transformer models like bidirectional neural networks to allow model generalization to diverse temporal trends in data. We also wish to explore the direction of applying cost-benefit analysis for each model component based on the deployment setting to avoid expensive computation.


