Transportation Research Part C 150 (2023) 104089

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Autonomous anomaly detection on traffic flow time series with reinforcement learning

Dan He [a], Jiwon Kim [a,*], Hua Shi [b], Boyu Ruan [c]

[a] School of Civil Engineering, The University of Queensland, St Lucia QLD 4072, Brisbane, Queensland, Australia
[b] School of Information Technology and Electrical Engineering, The University of Queensland, St Lucia QLD 4072, Brisbane, Queensland, Australia
[c] Shenzhen Institute of Computing Sciences, Shen Zhen, Guangdong, China

ARTICLE INFO

Keywords:
Anomaly detection
Traffic flow data
Reinforcement learning
Deep learning
Kernel density estimation

ABSTRACT

This study develops an autonomous artificial intelligence (AI) agent to detect anomalies in traffic flow time series data, which can learn anomaly patterns from data without supervision, requiring no ground-truth labels for model training or knowledge of a threshold for anomaly definition. Specifically, our model is based on reinforcement learning, where an agent is built by a Long-Short-Term-Memory (LSTM) model and Q-learning algorithm to incorporate sequential information in time series data into policy optimization. The key contribution of our model is the development of a novel unsupervised reward learning algorithm that automatically learns the reward for an action taken by the agent based on the distribution of data, without requiring a manual specification of a reward function. To test the performance of our model, we conduct a comprehensive set of experimental study on both real-world data from Brisbane city, Australia, and synthetic data simulated according to the distribution of real-world data. We compare the performance of our model against three state-of-the-art models, and the experimental results show that our model outperforms the other models in different parameter settings, with around 90% precision, 80% recall, and 85% F1 score.

1. Introduction

The detection of unexpected events or rare patterns on traffic data plays a significant role in intelligent traffic management, which provides crucial alerts of potential incidents or sensor faults. Accurate anomaly detection on traffic data can bring prompt troubleshooting and help timely decision-making for traffic operators. Generally, an anomaly is an observation or a sequence of observations that deviate significantly from the general distribution of the given data, and the amount of deviation is associated with the probability of being an anomaly. In this paper, we study anomaly detection for traffic flow data captured by loop detectors in Brisbane, Australia, where flow data are represented as univariate time series.

In the literature, various methods have been proposed for anomaly detection in univariate time series. A recent survey (Braei and Wagner, 2020) summarizes a comprehensive set of methods divided into three categories: statistics-based, classical machine learning, and deep learning methods. In particular, the statistics-based methods, such as auto-regressive integrated moving average (ARIMA) (Zhang et al., 2005), usually assume that the data could be generated by specific statistical models, where such a generative model is constructed based on the given data, and new data are fit to the model to determine anomaly. Machine learning algorithms such as K-Means Clustering (Nairac et al., 1999; Rebbapragada et al., 2009) and One-Class Support Vector Machine (SVM) (Eskin

* Corresponding author.
E-mail addresses: d.he@uq.edu.au (D. He), jiwon.kim@uq.edu.au (J. Kim), hua.shi@uq.edu.au (H. Shi), ruanboyu@sics.ac.cn (B. Ruan).

https://doi.org/10.1016/j.trc.2023.104089
Received 28 October 2022; Received in revised form 7 February 2023; Accepted 4 March 2023
Available online 15 March 2023

et al., 2002) try to detect the anomaly based on the proximity of data without assuming a specific generative model. More recently, deep learning techniques have been applied in anomaly detection, trying to improve the existing methods by capturing complex, nonlinear correlations in data. A general approach to these solutions (Goodfellow et al., 2016; Buda et al., 2018; Pang et al., 2021) is to predict the next value in a time series based on the data points at the previous time steps and determine an anomaly based on specific rules, e.g., whether the difference between the observed value and predicted value is larger than a pre-defined threshold value.

Challenges: However, the existing solutions are not directly applicable to our traffic anomaly detection problem due to the following challenges. (1) Most of the existing approaches use certain thresholds to determine anomalies—either predefined thresholds or self-adjusted thresholds. The performance of threshold-based methods can be very sensitive to the choice of a threshold. It is challenging to identify threshold values that work consistently and reliably across different times and varying traffic conditions. Thresholds are also highly location-specific since different roads have different flow patterns and data scales. Having to establish separate threshold settings for different locations further compounds the challenges. (2) Many existing solutions are supervised models requiring ground-truth labels classifying anomalies for model training. However, our data do not contain anomaly labels, which is common in real operational environments, and it is difficult to manually label the anomalies from the flow data as the size of the required training set is usually large. (3) We are interested in detecting a continuous sequence of anomalous data points in a flow time series rather than a single anomalous data point, where the anomaly sequences' length is unknown and varies from one to another. However, most of the existing sequence-based anomaly detection methods require a predefined length for an anomalous data unit. Thus, we need a more flexible model to detect flow anomaly sequences with different lengths.

Contributions: To address the above-mentioned research gaps, this study proposes the development of an artificial intelligence (AI) agent for anomaly detection based on Reinforcement Learning (RL), which learns to make a sequence of decisions to accurately detect the start and end of an anomalous segment in traffic time series data, without relying on predefined rules or requiring ground-truth labels. Specifically, there are four major contributions of our work.

- *Unsupervised reward learning:* We develop a novel data-driven reward function that automatically learns the reward based on the distribution of data. This is different from the existing RL-based anomaly detection models (Huang et al., 2018; Yu and Sun, 2020) that define the rewards based on the known anomaly information for flow data, which require ground-truth labels. With the proposed unsupervised reward learning method, our RL model does not require ground-truth labels.
- *Threshold-free:* Generally, most of the existing anomaly detection models require a predefined or self-adjusted threshold to determine anomalies. In contrast, our model is threshold-free, which makes the model more flexible and adaptable.
- *Generalization:* Our model does not require location-specific settings, and it can be easily applied to different locations without much human intervention. To train the model for a new detector location, we simply feed the new flow time series data to our RL model. Then, the RL agent interacts with the data to find the optimal anomaly detection policy for the new data pattern.
- *Performance:* We conduct a comprehensive set of experiments on both real-world data and synthetic data and compare our model with three state-of-the-art models. The results show that our model outperforms the other models by a large margin in different parameter settings. Overall, our model can achieve 90% precision, 80% recall, and 85% F1 score.

This paper is organized as follows. Section 2 reviews the relevant literature on anomaly detection. Section 3 provides mathematical preliminaries, problem statement, and background knowledge of reinforcement learning. Section 4 introduces the detailed methodology of our model. Section 5 presents the experimental studies for evaluating the model performance, and Section 6 presents the conclusion.

2. Literature review

Anomaly detection has been widely studied with various types of data (Chandola et al., 2009; Chalapathy and Chawla, 2019). Our work focuses on anomaly detection on traffic flow time series data. In the literature (Blázquez-García et al., 2021; Chalapathy and Chawla, 2019), studies consider three categories of anomalies in time series data according to different application domains and research perspectives: point anomaly, sequence anomaly, and time series anomaly. A *point anomaly* refers to noise or outliers appearing at a specific time instant and deviating largely from the other values in the time series or to its neighbouring points. A *sequence anomaly* is a sub-sequence of consecutive points in time series whose joint pattern differs from the normal behaviour. A *time series anomaly* refers to the entire time series that is an outlier, which can be detected when the time series of one variable significantly differs from the time series of other variables in multivariate time series data. In our work, we focus on the sequence anomaly in traffic flow data.

Considering the methodology perspective, numerous techniques have been proposed for time series anomaly detection (Blázquez-García et al., 2020; Braei and Wagner, 2020; Lai et al., 2021; Gupta et al., 2013). In this section, we review the existing studies on this topic, divided into four categories according to methodological approaches, namely, *statistics-based* methods, *prediction-based* methods, *similarity-based* methods, and *RL-based* methods.

Statistics-based Methods: Statistics-based methods usually assume that the data are generated by specific statistical models. Thus, they aim to construct a statistical model based on the given data and then examine whether new data fit the model (Box et al., 2015). In the literature, several traditional models are proposed for anomaly detection, including hypothesis testing (Rosner, 1983), wavelet analysis (Lu and Ghorbani, 2008), and auto-regressive integrated moving average (ARIMA) (Zhang et al., 2005). For time series data, Jagadish et al. (1999) propose a histogram-based algorithm to mine the deviant point in time series. Yamanishi et al. (2004) address this problem based on the statistical learning theory, employing an online learning algorithm to learn a probabilistic

model for anomaly detection. In recent years, extreme value theory (De Haan et al., 2006) is applied to develop the Peak Over Threshold (POT) and Streaming Peak over Threshold with drift (DSPOT) methods (Siffer et al., 2017) for univariate time series, where the threshold to determine the anomaly of the data point can be selected automatically. However, this work is only suitable for detecting anomaly values larger than the normal levels, while ignoring the anomalies smaller than the normal values.

Prediction-based Methods: Prediction-based methods are similar to statistics-based methods that first estimate the next value according to the previous sub-sequence, followed by the anomaly examination based on specific rules. For instance, Yu et al. (2014) use a sliding window of the previous data with non-linear weights to predict the next data point. Then, a Prediction Confidence Interval (PCI) is computed based on the predicted value to determine the anomaly. The Deep learning-based Anomaly Detection (DeepAD) framework proposed by Buda et al. (2018) applies the Long Short-Term Memory (LSTM) models to predict the next data point of the time series, in combination with some statistical models. A dynamic threshold setting over a sliding window of data is computed to detect the anomaly based on the prediction of LSTM. Hundman et al. (2018) develop an unsupervised model based on LSTM to predict the telemetry data of spacecraft, which requires no labelled data for training. Then a dynamic error threshold is computed based on historical errors to determine whether the predicted value is an anomaly or not. The deep learning-based anomaly detection approach for time series data (DeepAnT) proposed by Munir et al. (2018) applies a deep Convolutional Neural Network (CNN) as a time series predictor and measures the Euclidean distance between the predicted value and the true value as the anomaly score to determine whether it is an anomaly. Ren et al. (2019) propose an algorithm based on Spectral Residual (SR) (Hou and Zhang, 2007) and CNN, in which SR is to generate the anomaly detection results and the CNN model is applied to the output of SR model to learn a discriminate rule to replace the role of the threshold.

Similarity-based Methods: Similarity-based methods refer to the solutions that consider the distances among the input data, and the ones with larger distances to the other data are usually regarded as anomalies (Angiulli et al., 2005). Generally, the distance measures for time series data include the Euclidean distance, the longest common subsequence (LCSS) (Budalakoti et al., 2006), match count based sequence similarity (Lane et al., 1997), dynamic time warping (DTW) (Berndt and Clifford, 1994), etc. Most of the similarity-based methods are unsupervised algorithms (Goldstein and Uchida, 2016), which require no label on data. The popular similarity-based methods for anomaly detection are the clustering mechanism, such as k-means clustering (Nairac et al., 1999; Rebbapragada et al., 2009), k-medoids (Pan et al., 2010), dynamic clustering (Sequeira and Zaki, 2002), and One-Class SVM (Eskin et al., 2002). Since the computational costs for calculating similarity measures are expensive, some other studies focus more on the efficiency of anomaly detection. Liu et al. (2009) introduce an efficient algorithm to detect the discord in time series, where the discord refers to the sub-sequence that is the most dissimilar with its k nearest neighbours. Angiulli and Fassetti (2007) propose two algorithms: one returns exact anomaly queries with additional space consumption, and another provides an approximate answer with a statistical guarantee, in which a sliding window is applied to extract sub-sequence of time series as the detecting unit.

RL-based Methods: In recent years, with the rapid development of reinforcement learning (RL) algorithms in addressing sequential decision-making problems, a few studies have applied RL techniques in solving time series anomaly detection problems. de La Bourdonnaye et al. (2017) propose a model to learn binocular fixations with little supervised information, in which the informative reward for the RL model is computed by a convolutional autoencoder. Huang et al. (2018) build a value-based time series anomaly detector with Deep-Q-Network (DQN), which is a supervised algorithm with a reward measured by whether an anomaly is detected correctly. The policy-based time series anomaly detector (PTAD) proposed by Yu and Sun (2020) is a policy-based RL framework for discovering anomalous behaviours in time series data, where the reward is computed based on the confusion matrix of detected results. The RLAD model proposed by Wu and Ortiz (2021) combines the DQN algorithm with active learning for anomaly detection which is a semi-supervised solution requiring experts to label the ground truth during the process of anomaly detection. However, all these RL-based solutions partially or fully rely on labelled data for model training.

In summary, anomaly detection models generally require ground-truth labels for model training and threshold settings for determining anomalies. Since we do not have ground-truth data and we aim to avoid cumbersome threshold settings, the existing models cannot be directly applied to our problem. As such, we propose a novel RL-based anomaly detection model that can learn anomaly detection capabilities in an unsupervised and data-driven way without requiring ground-truth labels or specific thresholds.

3. Problem statement and preliminaries

In this section, we first introduce the problem statement of our work. Then, we present an overview of Reinforcement Learning (RL) and Long Short-Term Memory (LSTM) networks, based on which we build our anomaly detection model.

3.1. Problem statement

We aim to detect anomaly patterns in a sequence of traffic flow data observed during a period of time on a given link of a road network.

Definition 1 (*Traffic Flow Sequence*)**.** We define a traffic flow sequence as a univariate time series of link volume (the number of vehicles per unit time, e.g., veh/h) for a specific link, denoted by $X = \{f_{t_1}, f_{t_2}, \dots, f_{t_n}\}$, where f_{t_i}, $i = 1, \dots, n$ represents a flow value at time interval t_i.

Usually, traffic flow data are collected using loop detectors installed on links, and each flow value (f_{t_i}) is measured by aggregating raw vehicle counts over a fixed time interval (e.g., 3 min) and representing it as an hourly volume (veh/h).

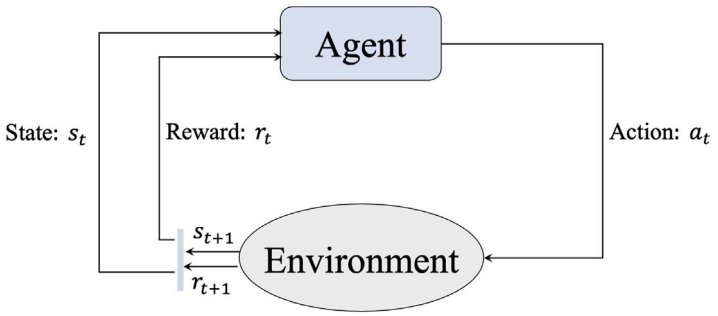


Fig. 1. General framework of RL model.

Definition 2 (*Traffic Flow Anomaly*)**.** We define an anomaly in traffic flow time series as a subsequence $\{f_{t_a}, \ldots, f_{t_b}\}$ with flow values that deviate largely from the normal behaviour, where t_a and t_b are the start time index and end time index of the anomaly time period, respectively.

It is noted that the traffic flow anomaly defined above is a ***sequence anomaly***, which is typically characterized as a sudden and extended drop in traffic flow (i.e., a continuous sequence of anomalous points). Sequence anomalies may reflect a real disruption in traffic flow caused by traffic incidents or indicate an extended period of sensor malfunction and data measurement errors. There is also a *point anomaly* that refers to a single anomalous point in time series, which usually indicates noise created by a data measurement error. While these two types of anomalies both exist in traffic time series data, their characteristics are quite different. Thus, it is challenging to address both types of anomalies simultaneously. Instead, we have decided to focus on sequence anomalies in our initial model for the following reasons. First, actual traffic disruptions are usually represented as sequence anomalies as a full process of traffic breakdown and recovery happens over a period of time rather than at a single point. Automatically detecting actual traffic breakdowns from data is of primary interest for automation in traffic monitoring and congestion management. Second, when anomalies are caused by data measurement errors, the impact of a sequence anomaly can be generally greater than that of a point anomaly due to its extended period of data corruption. Third, based on our analysis of real-world traffic data for this study, we could hardly observe significant point anomalies. In contrast, there were many sequence anomalies with different lengths and patterns, as demonstrated in the case studies in Section 5.2. Finally, a sequence anomaly is more general than a point anomaly, as a point anomaly can be viewed as a special case of a very short sequence anomaly. As such, we intend to focus on tackling the sequence anomaly detection problem first and then extend our model to address point anomaly detection in the future.

Definition 3 (*Traffic Flow Anomaly Detector*)**.** Given a sequence of traffic flow data $X = \{f_{t_1}, f_{t_2}, \ldots, f_{t_n}\}$, the anomaly detector generates an output sequence $Y = \{y_{t_1}, y_{t_2}, \ldots, y_{t_n}\}$, where $y_{t_i} \in \{0, 1\}$ is a binary indicator with $y_{t_i} = 1$ indicating f_{t_i} is anomalous and $y_{t_i} = 0$ indicating f_{t_i} is normal.

It is worth noting that the lengths of anomaly sequences are various, and occurrence time intervals can be different. Some existing sequence-based anomaly detection methods, e.g., Discord (Yankov et al., 2008), apply a fixed-length sliding window to form the data unit for detection so that the length of anomaly sequences is stable. In contrast, our anomaly detector identifies each flow value individually such that anomaly sequences with various lengths and different occurring time intervals can be captured.

3.2. Reinforcement learning

Reinforcement learning (RL) solves a sequential decision-making problem represented as a Markov Decision Process (MDP), where an agent chooses sequential actions by interacting with its environment. Fig. 1 shows a general framework of the reinforcement learning model. This process is formulated by a tuple with five elements: $(\mathcal{A}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \gamma)$. At each time step t, the agent takes an action $a_t \in \mathcal{A}$ according to the current state $s_t \in \mathcal{S}$ in the environment $\mathcal{E}$ based on a policy π. Then, the environment $\mathcal{E}$ presents a reward $r_{t+1} = \mathcal{R}(s_t, a_t)$ to the agent and updates the agent's state to the next state s_{t+1} according to the state transition probability function $\mathcal{P}(s_{t+1}|s_t, a_t)$. The overall objective of the RL agent is to learn an optimal policy π^* that can receive a maximum cumulative reward $G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$, with the discount factor $\gamma \in (0, 1]$.

In general, an RL algorithm can be categorized as either *model-based* or *model-free* depending the availability of a model that describes the environment (a function that predicts state transitions and rewards). A model-based RL algorithm uses a model of the environment within the algorithm, where the model can either be given or learned by the agent. The agent learns the optimal policy by making model-based predictions about the consequences of its actions. The most well-known model-based RL algorithm is Dynamic Programming (Sutton and Barto, 2018). A model-free RL algorithm, on the other hand, does not use a model of the environment. The agent does not have access to a model of the environment but learns the optimal policy by experiencing the consequences of its actions through exploration. To implement this, a model-free RL algorithm estimates either a value function or the policy directly from the agent's experience while it interacts with the environment. As such, model-free algorithms are often

further classified into *policy-based* algorithms and *value-based* algorithms. It is, however, noted that policy-based and value-based methods are not always exclusive of each other, and some algorithms use both approaches (e.g., Actor–Critic methods).

The policy-based algorithms try to directly learn the target policy $\pi(a|s)$ to map the states to the actions, while the value-based algorithms aim to first estimate a value function and then derive the policy from the value function. In this study, we adopt a value-based algorithm to solve the RL for our anomaly detection problem. A value function is a function that predicts the expected cumulative reward at a given state or state–action pair, estimating how good it is for the agent to be in a given state or to perform a given action in a given state. In particular, the action value function, also known as *Q-function*, for a policy π is defined as the expected, discounted cumulative reward for taking action a in the current state s and then forever acting according to policy π, as follows:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s, a_t = a\right] \tag{1}$$

Then, the optimal action value function, $Q^*(s,a)$, is the maximum action value function over all policies as shown below, and we can find an optimal policy, π^*, by maximizing over $Q^*(s,a)$, i.e., by picking action $a = \arg\max_a Q^*(s,a)$.

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{2}$$

Q-learning algorithm (Watkins and Dayan, 1992) is introduced to learn the action value function for policy improvement, with the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{3}$$

where s' and a' are the next state and action, respectively, and α represents the learning rate. Traditional Q-learning algorithms use a *Q-table* to represent the Q-function, which is a simple look-up table to store the Q-function value for every state–action pair. For most real-world problems, however, it is impractical to represent the Q-function as a table due to large state and action spaces. As such, it is common to use a function approximator such as a neural network to estimate the Q-function values, which is called *Deep Q-learning*. One example of deep Q-learning algorithms is the Deep Q-Network (DQN) algorithm developed by DeepMind (Mnih et al., 2015), which uses a deep convolutional neural network to approximate the Q-function. In our work, we use deep Q-learning with a deep recurrent neural network, which is also a commonly used model-free, value-based RL algorithm.

3.3. Long short-term memory networks

Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997) is a variant of recurrent neural network (RNN) that has been widely used to learn patterns in sequential data in deep learning research. LSTM networks were introduced to address limitations of standard RNNs, especially the *vanishing gradient* problem, which arises when the length of a sequence is long because error signals backpropagated across many time steps tend to vanish, making it hard for the networks to update the weights and biases. LSTM network helps alleviate this problem by replacing the recurrent weight matrix in standard RNN models with the identity function and controlling the flow of information by a series of gates, which allow LSTMs to pass relevant information down the long chain of sequences to effectively learn the sequential dependencies across the input sequence. With its capability to learn long sequences of data, LSTM has become a popular choice for time series forecasting, which aims to predict the future values in a time series given a sequence of previous observations as input. In this study, we use LSTM networks as a function approximator to represent the Q-function in the Q-learning framework to solve the RL problem for the time series anomaly detection. The details of our model architecture will be described in Section 4. Here, we give a brief introduction to the mechanism of an LSTM network.

Fig. 2 shows the architecture of one LSTM unit. Generally, there are three layers used in information propagation, namely the input layer, hidden layer, and output layer. The input layer initializes input data for subsequent layers, while the output layer generates the output for the final prediction task (e.g., classification or regression). The hidden layer is designed for encoding the features in the input sequence through a recurrent unit called memory block, which contains a cell to transfer relevant information throughout the sequence and three gates, including a forget gate, an input gate, and an output gate, as shown in Fig. 2. The forget gate is responsible for memorizing and recognizing the information coming inside the network and discarding the irrelevant information. There are two sources of input: one is the information from the previous layers, i.e., h_{t-1}, and the other is the information from the current layer, i.e., X_t. Next, the input gate aims to decide the information's importance by updating the cell state. The input information, i.e., X_t, goes through the *sigmoid* and *tanh* functions in which the *sigmoid* decides the weight of information and *tanh* reduces the bias of the network. Finally, the output gate works to generate the next hidden state, i.e., h_t, of the network, where the information goes through the *sigmoid* function, and then the updated cell information from the cell state goes to the *tanh* function multiplied with the *sigmoid* function of the output state. Each process is formulated as follows.

Forget gate:

$$f_t = \sigma(W_f \times X_t + V_f \times h_{t-1} + b_f) \tag{4}$$

Input gate:

$$i_t = \sigma(W_i \times X_t + V_i \times h_{t-1} + b_i) \tag{5}$$

$$\widetilde{C}_t = \tanh(W_{\widetilde{C}} \times X_t + V_{\widetilde{C}} \times h_t + b_{\widetilde{C}}) \tag{6}$$

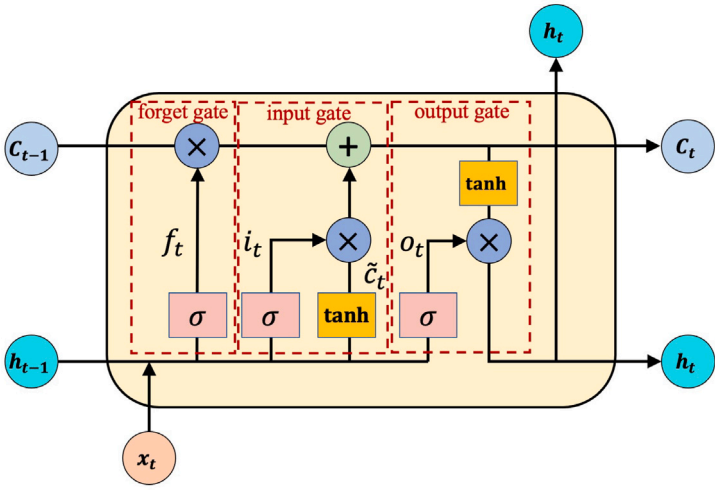


Fig. 2. Schematic diagram of the memory block of LSTM unit.

Cell state update:

$$C_t = f_t \times C_{t-1} + i + t \times \widetilde{C}_t \tag{7}$$

$$o_t = \sigma(W_o \times X_t + V_o \times h_{t-1} + b_o) \tag{8}$$

$$h_t = o_t \times \tanh(C_t) \tag{9}$$

where X_t denotes the input data at time step t and C_t indicates the hidden cell state at t. Variables f, i, C and o represent forget gate, input gate, cell state vectors, and output gate, respectively. Operator $\times$ represents the scalar product of two vectors, $\sigma()$ denotes the standard logistic *sigmoid* function, which scales the inputs into the range of $(0, 1)$, and tanh() is the hyperbolic tangent function, which transforms the input and output into the range of $[-1, 1]$. Parameters $W_j(j = f, i, \widetilde{C}, o)$ and $V_j(j = f, i, \widetilde{C}, o)$ denote the weight matrices in each layer and $b_j(j = f, i, \widetilde{C}, o)$ denotes the bias of each gate.

4. Methodology

In this section, we explain the methodology of our RL-based anomaly detection model. Fig. 3 shows an overview of the proposed model, which contains three modules: input module, reinforcement learning (RL) module, and output module. Each module is discussed in detail below.

4.1. Input module

We consider the operational scenario of monitoring a city-wide traffic network with hundreds or thousands of loop detectors installed on the links across the network. We aim to develop an anomaly detection model that can be flexibly applied to a wide range of links with different flow patterns. Our model is designed and trained for each specific detector by learning patterns in the univariate time series of flow data from the given detector. For instance, Fig. 3's Input Module illustrates traffic flow time series coming from Detectors 1, 2, and 3, where we choose Detector 3 to illustrate further the process of training the proposed RL model for anomaly detection. The traffic flow sequence from a single loop detector (Detector 2 in this example), $X = \{f_{t_1}, f_{t_2}, \ldots, f_{t_n}\}$, is taken as input to the RL module for training. To cover multiple detectors, we can simply create multiple instances of the model (multiple RL agents), each of which will automatically learn to detect anomalies in the associated detector data by adapting to its own local data pattern without supervision. Since there is no location-specific setting or ground-truth labelling required to train our model, it is easy and straightforward to create and train multiple anomaly detectors for other locations.

4.2. RL module

We adopt a reinforcement learning framework to build our anomaly detector agent, which automatically learns from traffic flow data to identify anomalies. Specifically, we formulate the anomaly detection problem as a sequential decision-making problem by defining the state and action spaces for the anomaly detector agent as follows.

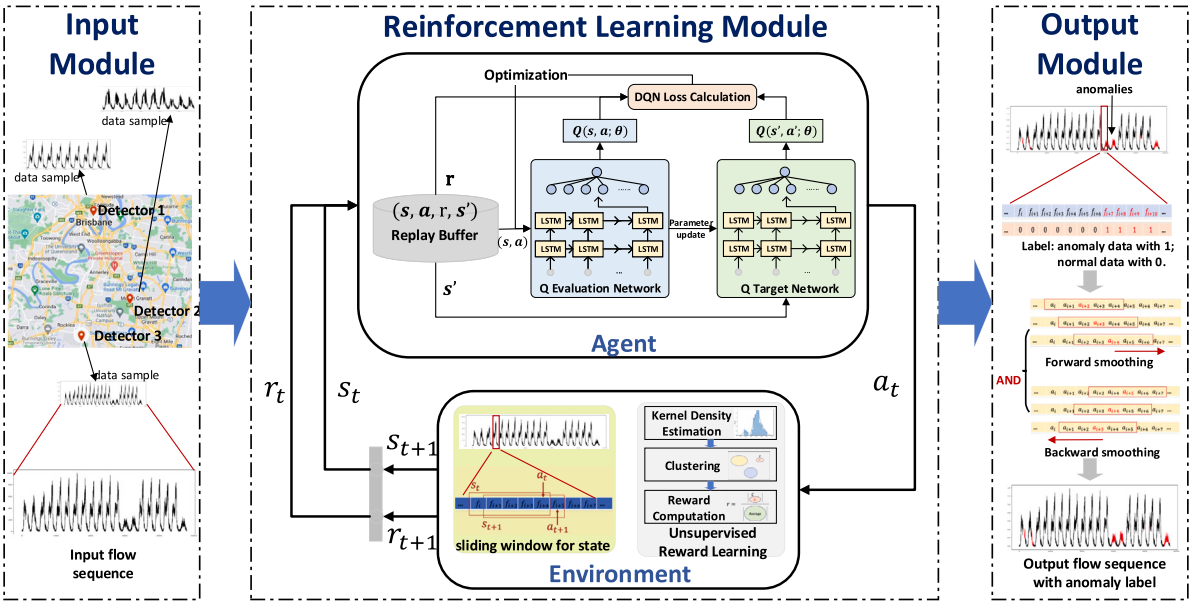


Fig. 3. Model overview.

Definition 4 (*State Space for Anomaly Detector Agent*)**.** The state of an anomaly detector agent at time step t is defined as a sequence of observations for traffic flows and the chosen actions over the past m time steps as follows:

$$s_t = \{o_{t-m+1}, \dots, o_{t-1}, o_t\}$$

where o_i is the observation at time step i represented as a 3-tuple $(f_i, \bar{f}_i, a_i)$ consisting of the current traffic flow at time step i (f_i), the historical mean flow for the same time-of-day period ($\bar{f}_i$), and the chosen action at time step i (a_i).

Definition 5 (*Action Space for Anomaly Detector Agent*)**.** The action space of an anomaly detector agent is defined as $A = \{0, 1\}$. Given the state at time step t, s_t, the anomaly detector agent chooses an action $a_t = 1$ if the current traffic flow f_t is regarded as an anomaly and $a_t = 0$ if f_t is regarded as normal data.

Fig. 4 illustrates an example of the state and action defined above. At each time step t, the anomaly detector agent determines whether the current link flow, f_t, is an anomaly or not by selecting a binary action $a_t \in \{0, 1\}$. Rather than taking only the current flow f_t as the state, we consider a sequence of flows over the past m time steps as the state $(f_{t-m+1}, \dots, f_t)$ so that the agent can take into account the temporal change in the flow time series data. In addition, we include the historical mean of the flows at each time-of-day interval over the corresponding m time steps in the state definition $(\bar{f}_{t-m+1}, \dots, \bar{f}_t)$ to allow the agent to consider the information on how much the current flow value deviates from the typical flow levels measured by the historical mean flows. We assume that historical flow data covering a sufficiently long observation period (e.g., over several months) are available to calculate the historical mean flow for each time step. Finally, the previously chosen actions over the m time steps $(a_{t-m+1}, \dots, a_{t-1})$ are added to the state definition to incorporate the agent's past decisions into the current decision-making. Since the three elements at each time step i, $(f_i, \bar{f}_i, a_i)$, are processed together as a tuple, the full specification of s_t requires the value of a_t, which has not been determined yet. For this, a_t is tentatively set to -1 to specify s_t. It is also noted that we define the state s_t for $t \geq m$ to ensure that all $m-1$ previous flow data are available. For the initial state $s_t = s_m$ $(t = m)$, the first $m-1$ actions are set to 0 by assuming that the first $m-1$ flows are normal. Another thing to note is that the state definition in Definition 4 plays an important role in fulfilling the *Markov* property required in RL settings (Sutton and Barto, 2018). While our problem may not naturally have the Markov property because choosing optimal actions requires information for historical observations in time series, by defining the whole sequence of the m past observations as our 'current' state, we effectively satisfy the Markov property because it allows the agent to make decisions only based on the current state.

Our goal is to make the agent capable of detecting a continuous set of data points that are collectively anomalous, rather than focusing on each individual data point separately. This is a sequential decision-making problem that can be effectively solved by RL because the decisions are interdependent over time. For instance, consider the shaded area (f_{t-2}, f_{t-1}, f_t) in Fig. 4 that represents a continuous anomalous data sequence. When the agent determines the start of this anomalous sequence (a_{t-2}), it should be able to reason about future actions, e.g., whether it is the start of a significant anomalous period that is likely to be followed by a series of consecutive ones $(a = 1)$ or it is just a point anomaly that is likely to be followed by zeros $(a = 0)$. Once the agent detects the start of a true anomalous flow sequence correctly, its subsequent actions will be informed by this decision through the states containing the previously chosen actions and will likely detect the subsequent anomaly points correctly by capturing the continuous anomaly sequence patterns over time.

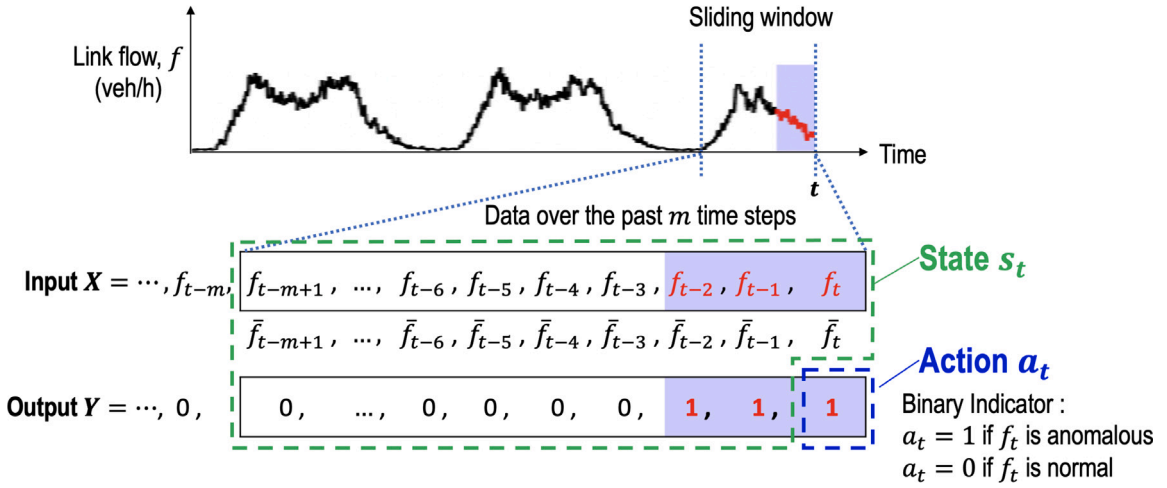


Fig. 4. Example of the state and action for the proposed RL-based anomaly detector.

4.2.1. *Unsupervised reward learning*

Defining a reward function is the most critical step in RL as the agent's learning heavily relies on the reward feedback for its chosen actions. In our anomaly detection problem, the agent should receive a positive reward for correctly detecting an anomaly, while receiving a penalty (e.g., negative reward) for an incorrect detection. However, as there is no ground-truth label available, it is challenging to judge whether the agent makes a correct decision or not. As such, we develop a data-driven reward function that can set the reward value for a given state–action pair automatically based on the data. The basic idea is that we introduce a measurement, namely *normality score*, to quantify how 'normal' the current flow f_t is compared to the 'typical' flow values for the current time-of-day period. Then, we define a reward value based on this normality score such that the agent receives a positive reward for regarding f_t with a *low* normality score as an anomaly while receiving a penalty for regarding f_t with a *high* normality score as an anomaly. To determine the 'typical' flow values for a given time-of-day period, we use historical flow values collected over several months. The detailed procedure to calculate the normality score and reward values is explained below, in conjunction with the illustration of the proposed concept in Fig. 5.

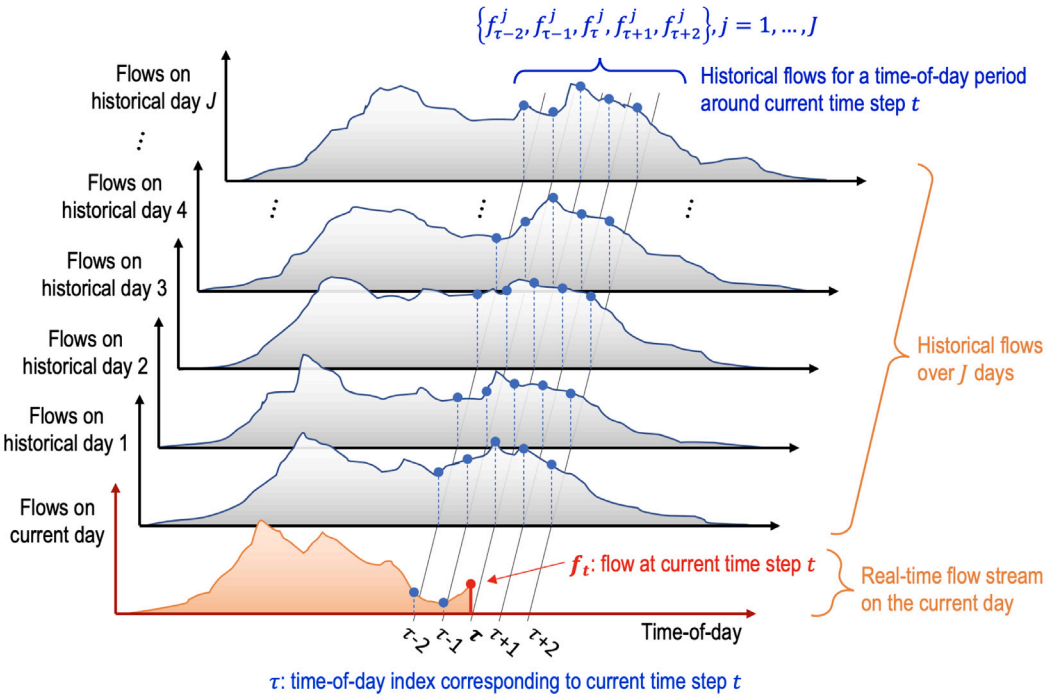
Normality score and reward function. Given a flow value f_t at time step t, we compute the associated *normality score* and define the reward function in the following steps:

1. Let D denote a reference flow database, which is a set of historical daily flow time series over J historical days: $D = \{\mathbf{f}^1, \mathbf{f}^2, \ldots, \mathbf{f}^J\}$, where $\mathbf{f}^j$ is the one-day flow time series for day j, $\mathbf{f}^j = \{f_1^j, f_2^j, \ldots, f_T^j\}$, indexed from 1 to T with T representing the number of time-of-day indices within a day. Given current time step t, which is a 'global' time index that is unique across days and times, we identify the corresponding *time-of-day* index, denoted by τ, which is a 'local' time index within each day. The same step size is used for both t and τ. We then extract the historical flow data around τ from D to capture typical flow patterns in the time-of-day period similar to the current time t. Specifically, we include two time steps before and after τ and extract a set of historical flow data associated with current time t, namely $D^{(t)} = \{f_{\tau-2}^j, f_{\tau-1}^j, f_\tau^j, f_{\tau+1}^j, f_{\tau+2}^j\}$ for $j = 1, \ldots, J$.
2. Let $\mathcal{F}^{(t)}$ denote the set combining f_t and $D^{(t)}$, i.e., $\mathcal{F}^{(t)} = \{f_t\} \cup D^{(t)}$. We then perform clustering on $\mathcal{F}^{(t)}$ using Kernel Density Estimation (KDE), which is a non-parametric way to estimate the probability density function (PDF) of a random variable. KDE produces a smooth curve of PDF based on kernel smoothing and can be used to cluster 1-dimensional data by identifying distinct peaks (local maxima) in the PDF.
3. We denote the cluster containing f_t by $C^{(t)}$ and then calculate the normality score, δ_t, defined as the ratio of the size of $C^{(t)}$ to the average size of all clusters obtained from step 2:

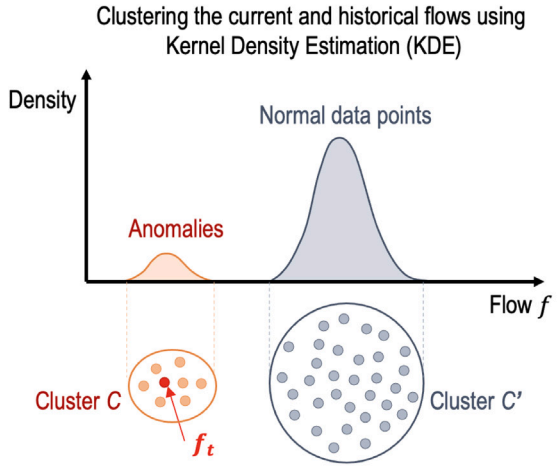
$$\delta_t = \frac{|C^{(t)}|}{|\mathcal{F}^{(t)}|/k_t} \tag{10}$$

   where k_t is the number of clusters found in $\mathcal{F}^{(t)}$ and $|\mathcal{F}^{(t)}|/k_t$ gives the average size of all k_t clusters.
4. Based on δ_t, we define the reward function as follows:

$$r_{t+1} = \begin{cases} 1/\delta_t & \text{for } \delta_t < 1, a_t = 1 \text{ (reward)} \\ \delta_t & \text{for } \delta_t \geq 1, a_t = 0 \text{ (reward)} \\ -1/\delta_t & \text{for } \delta_t < 1, a_t = 0 \text{ (penalty)} \\ -\delta_t & \text{for } \delta_t \geq 1, a_t = 1 \text{ (penalty)} \end{cases} \tag{11}$$



(a) Data extraction (Step 1)



(b) Clustering (Step 2)

Normality Score $\boldsymbol{\delta} = \dfrac{\text{Size of Cluster containing } f_t}{\text{Average Cluster Size}}$

Model \ Data	$\delta < 1$ (anomalous)	$\delta \geq 1$ (normal)
$a = 1$ (anomalous)	$\boldsymbol{r = 1/\delta}$ **(reward)**	$\boldsymbol{r = -\delta}$ **(penalty)**
$a = 0$ (normal)	$\boldsymbol{r = -1/\delta}$ **(penalty)**	$\boldsymbol{r = \delta}$ **(reward)**

(c) Reward computation (Step 3 and 4)

Fig. 5. Example for unsupervised reward learning.

where r_{t+1} is the reward given to the RL anomaly detector after observing the state s_t (reflected in δ_t) and action a_t at the current time step t.

Given that $\mathcal{F}^{(t)}$ contains the flows from a similar time-of-day period as illustrated in Fig. 5(a), the basic assumption of the proposed normality score, δ_t, in Eq. (10) is that normal flow data in $\mathcal{F}^{(t)}$ are similar to each other (clustered together) and anomalies form distinct clusters with much smaller sizes than normal data clusters. This makes the normality score, δ, low when the target flow value f_t is an anomaly because f_t would belong to a small cluster of anomalies (i.e., a small $|C_t|$), whereas δ would be high when f_t is normal because f_t would belong to a large cluster of normal data points (i.e., a large $|C_t|$). Fig. 5(b) illustrates an example of the former case, where cluster C_t to which f_t belongs is a small anomalous cluster. Based on this intuition, we consider two categories of the normality score: $\delta_t < 1$, where f_t is likely to be an anomaly, and $\delta_t \geq 1$, where f_t is likely to be a normal data point. The reward, r_{t+1}, in Eq. (11) then gives a positive reward when the model prediction (a_t) is *consistent* with the data (δ_t), i.e., the RL agent labels f_t as anomalous ($a_t = 1$) when the data classifies it as anomalous ($\delta_t < 1$) or the RL agent labels f_t as normal ($a_t = 0$) when

the data classifies it as normal ($\delta_t \geq 1$). The reward gives a negative reward (penalty) when the model prediction (a_t) is *inconsistent* with the data (δ_t), i.e., the RL agent labels f_t as normal ($a_t = 0$) when the data classifies it as anomalous ($\delta_t < 1$) or the RL agent labels f_t as anomalous ($a_t = 1$) when the data classifies it as normal ($\delta_t \geq 1$). Fig. 5(c) provides a summary table of this rule.

4.2.2. Deep Q-network

Using the reward defined above, we train our RL model to learn the optimal policy for anomaly detection. We adopt a Deep Q-Network (DQN) algorithm (Mnih et al., 2015) to solve the RL problem. The DQN is a Q-learning method that uses a deep neural network to approximate the Q-value function, where the neural network takes the state as input and produces the Q-value for each action. According to Definition 4, our state is a sequence of 3-tuples $(f_i, \bar{f}_i, a_i)$. In order to better capture the sequential correlations from the state, we employ the Long-Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997) as the core brain of our agent, which is a sequence-based deep learning model widely used for time series data. More specifically, as shown in Fig. 3, we develop a stacked-LSTM to encode the features from the state and output an intermediate vector representation capturing the sequential correlations. Next, we concatenate a fully-connected neural layer at the end of the stacked LSTM that takes the vector representation as input and generates the output indicating the possibilities of two actions $a = 0$ and $a = 1$.

There are two key techniques applied in the DQN for policy optimization: Q target network and experience replay, which aim to tackle the instability issue in general DQN. This is because when a deep neural network is applied to represent the value function, it might be unable to converge on the optimal value function due to the high correlation between actions and states.

Experience Replay. We first define the experience tuple as (s, a, r, s'), where s, a, r refer to the current state, action, reward, and s' indicates the next state. During the interaction between the agent and the environment, the sequence of experience tuples generated step by step could be highly correlated, which might induce the risk of getting swayed when learning from each of these experiences tuples in sequential order. To avoid such issues, we employ a replay buffer with a fixed size to store the historical experience tuples. At each time step, we randomly sample a small batch of experience tuples from the replay buffer, from which we update the parameters of the DQN. We update the replay buffer by replacing the eldest experience tuple with a new one.

Q Target Network. Based on the update rule of Q-learning (Eq. (3)), we are trying to update $Q(s, a)$ with $Q(s', a')$ step by step. However, since these two states are close to each other and are usually very similar, which makes it hard for a neural network to distinguish between them. Thus when we update the parameters aiming at a $Q(s, a)$ closer to the desired result, the value of $Q(s', a')$ could be affected, resulting in very unstable training performance. To prevent this situation, we develop another neural network, denoted by 'Q target network', with exactly the same architecture as the major network, named 'Q evaluation network'. As illustrated in Fig. 3, at each time step, the Q evaluation network estimates the $Q(s, a)$, while the Q target network works on the representation of $Q(s', a')$. The predicted Q-value from the Q target network is used to back-propagate through and train the Q evaluation network. However, we do not update the parameters for the target network synchronously. Instead, we periodically copy the parameters from the Q evaluation network to replace the ones for the Q target network, which is able to improve the stability of the training.

To obtain a trade-off between the exploration and exploitation, we employ an ϵ-greedy strategy, where a greedy factor ϵ is used to decide whether our agent takes actions based on the Q function or is randomly picked from the action space. In this case, the agent can have more opportunities to encounter different situations for better options. We initialize the ϵ with a value in the range $[0, 1]$ and gradually decrease it step by step until it reaches the predefined lower bound. Similar to Huang et al. (2018), we design the loss function of our model by measuring the mean squared error between the Q-value obtained from the Q target network and the one computed based on Eq. (3), which is formulated by Eq. (12), where N indicates the batch size for training.

$$Loss = \frac{1}{N}\sum_{i=1}^{N}\left[\left(r + \gamma \max_{a'} Q(s', a'|\theta_{i-1}) - Q(s, a|\theta_i)\right)^2\right] \tag{12}$$

The pseudocode of DQN algorithm is shown in Algorithm 1, which demonstrates the execution flow of the deep-Q-network with ϵ-greedy strategy, experience replay and target network applied. Firstly, we initialize the replay buffer, the action-value function Q with random parameter θ, and the action-value function $\hat{Q}$ with random parameter $\hat{\theta} = \theta$ for the target network. Then we perform P episodes for model learning. For each episode, we initialize the first state s_1. And then for each time step t, we select an action a_t following the ϵ-greedy policy that we have ϵ probability to random choose an action from the action space, and alternatively we choose based on the output of the neural network. Based on this action, we obtain the reward r_t according to our reward learning approach and get the next state s_{t+1} from the environment. Then we update the parameters by backpropagation with a minibatch of transitions randomly sampled from the reply buffer. Note that, the parameter $\hat{\theta}$ in the target network remains stable and will be replaced with parameter θ in every C step. The network is optimized iteration by iteration to achieve a better approximation of the action-value function. The algorithm terminates after P episodes and the actions in the Pth episode are regarded as the final output of the model.

4.3. Output module

From the RL module, we obtain a sequence of actions, which can be regarded as labels for each flow value based on Definition 5. We observe from some case studies that the detected result from our Rl module contains some false alarms, where some sequences with a single value or very few flow values are regarded as an anomaly. This might be caused by the randomness of action taken based on the ϵ-greedy strategy in the RL module. To eliminate such false alarms, we conduct post-processing to smooth the whole

Algorithm 1 DQN Algorithm

Require: traffic flow time series X
Ensure: Q action-value function (from which we obtain a policy and select actions)
initialize replay buffer M
initialize action-value function Q with random weight θ
initialize target action-value function $\hat{Q}$ with random weight $\hat{\theta} = \theta$
for episode = 1 to P **do**
Initialize state s_1 from X
for $t = 1$ to T **do**
following ϵ-greedy policy, select $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$
execute action a_i in agent network and obtain reward r_t based on our reward learning
get s_{t+1} from X
store (s_t, a_t, r_t, s_{t+1}) in M
sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from M ▷ experience replay
set $y_i = \begin{cases} r_j & \text{if episode terminates st step} j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \hat{\theta}) & \text{otherwise} \end{cases}$
perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ w.r.t. the network parameter θ
in every C steps, reset $\hat{Q} = Q$, i.e. $\hat{\theta} = \theta$ ▷ periodic update of target network
end for
end for

action sequence with a bidirectional sliding window, including forward smoothing and backward smoothing, as shown in Fig. 3. In particular, given the output action sequence, notated by $\{a_1, a_2, \dots, a_n\}$, the forward smoothing starts from a_1 with a size $2l+1$ sliding window moving from a_1 to a_n. For instance, the first sub-sequence of actions is $\{a_1, a_2, \dots, a_{2l+1}\}$, and the second sub-sequence is $\{a_2, a_3, \dots, a_{2l+2}\}$. For each sub-sequence $\{a_{i-l}, a_{i-l+1}, \dots, a_i, \dots, a_{i+l-1}, a_{i+l}\}$, we determine the value a_i based on the majority of all the other actions by the following rule. (1) If there are more than l actions are 1, then we set $a_i = 1$. (2) If there are more than l actions equal to 0, we set $a_i = 0$. Similarly, the backward smoothing starts from a_n and performs the same strategy as the forward smoothing. In the end, we conduct an AND operation on the results of forward smoothing and backward smoothing, such that for each action it is set to 1 only if it is 1 after smoothing on both sides. In this case, some random false alarms with a single or only a few flow values can be removed as their neighbouring flows are mostly labelled as normal data. To this end, we obtain the final detection result of our model, which is formed by a sequence of flow with the label either 1 or 0 indicating whether or not a flow value is anomalous.

5. Experimental study

In this section, we present an experimental study of our model and a comparison with several existing state-of-the-art models for anomaly detection on univariate time series data. Firstly, we explain the experiment settings and the selected benchmark models. Then, we analyse the performance of the models using both real-world data and synthetic data.

5.1. Experiment settings

5.1.1. Benchmark models

We compare the performance of our model with that of three benchmark models: two deep learning models and one statistical model. As with our model, all three benchmark models are chosen from unsupervised models that do not require ground-truth labels for model training. Next, we give a brief introduction to each benchmark model and its parameter settings used for the experiments.

- **DeepAnt** (Munir et al., 2018) uses a 1D convolutional neural network (CNN) that takes a sub-sequence of previous l_h steps of flows as input and predicts the flows for the next l_p time steps. Then it measures the L2-norm distance between the predicted values and the observed flows and determines whether the observed flows are anomalous or not based on a manually selected threshold. The parameters for this model are as follows: kernel size = 4, pool size = 2×2, batch size = 70, hidden layer = 80, drop out rate = 0.5, $l_h = 20$, and $l_p = 1$.
- **Telemon** (Hundman et al., 2018) applies an LSTM network to take a sub-sequence of previous flows with length l_h as input and predict the flows for the next l_p time steps. Then it calculates the prediction error by comparing the predicted flows with the observed flows and determines whether the observed flows are anomalous or not based on a dynamically and automatically computed threshold according to the distribution of the most recent observed flow sequence. The general idea of Telemon is similar to DeepAnt in that they are both prediction-based methods using deep learning techniques. The main difference is that Telemon does not require a manually-defined threshold, whereas DeepAnt requires such a setting. The parameters for Telemon are as follows: batch size = 50, hidden layer = 80, drop out rate = 0.3, $l_h = 120$, and $l_p = 20$.

Table 1
Configuration of our model.

Parameters	Value	Description
m	20	The number of time steps in the state (LSTM cells)
input_dim	3	The dimension of the input for each LSTM cell
hidden_dim	128	The dimension of hidden layer in each LSTM cell
n_layers	2	The number of stacked LSTM layers
nn input_dim	128	The dimension of the input for the fully connected layer
nn output_dim	2	The dimension of the output for the fully connected layer
γ	0.99	The discount factor
ϵ	0.5	The initial probability for random actions
ϵ_{min}	0.01	The minimum probability for random actions
ϵ_{dec}	5×10^{-6}	The decreasing value for ϵ at each time step
lr	0.0001	Learning rate
epoch	8	The number of running epochs
N	32	The size of each batch for training
mem_size	10000	The size of replay buffer
l	10	The size of the sliding window for smoothing is $2l + 1$

- **BreakoutDetection (BD)** (Vallis et al., 2014; Twitter, 2015) is an open-source R package released by Twitter that automatically detects anomalies in time series data. The model is based on the Seasonal Hybrid ESD (S-H-ESD) algorithm, which is derived from the generalized Extreme Studentized Deviate (ESD) test (Rosner, 1983). It measures the most deviated flow from the flows at the same time-of-day and marks it as an anomaly based on robust statistical metrics.

Since Telemon and DeepAnt are prediction-based models, they require a training process to build their prediction capabilities. This process is done separately from anomaly detection tasks, where the prediction models are trained via supervised learning to minimize a prediction error measured by the root mean square (RMS) between the predicted and actual flow values. We train both Telemon and DeepAnt with three-month flow data so that the models can capture the typical flow patterns, which will be used as the baseline for their prediction-based anomaly detection.

5.1.2. Model configuration

Our model is implemented with Python 3.7 and PyTorch 1.0.0, and the model parameters are listed in Table 1. For the model optimization, we adopt the Adam optimizer (Kingma and Ba, 2014) to perform stochastic optimization for minimizing the loss function in Eq. (12), which is an efficient optimization algorithm widely used in training neural network models. It is worth noting that our model does not require training with labelled data. Thus, the model learns by observing the input time series for multiple epochs (as set in Table 1), and the outputs from the last epoch are regarded as the final results. As for the ϵ-greedy strategy, the initial ϵ is set to 0.5. And in each time step, the ϵ decreases at ϵ_{dec} until it reaches $\epsilon_{min} = 0.01$, which enables the model to explore the data continuously. We select the output from the epoch with the largest accumulative rewards as our final results.

5.2. Evaluation with real data

To evaluate the model performance on real-world flow data, we select two link flow sequences from different loop detectors. Since there are no ground-truth labels available for the real-world flow data, the performance cannot be quantitatively measured, so we investigate each model's anomaly detection results visually. Specifically, we use three-month flow data from two different links in Brisbane, Australia, where hourly flows (veh/h) are measured based on traffic counts collected every 3-minute aggregation interval. Since flow patterns are usually different between weekdays and weekends, here we only consider the flow data from weekdays.

We show sample flow time series and detection results for the two selected links in Fig. 6 (referred to as Case 1) and Fig. 7 (referred to as Case 2), respectively. For each case, the sample flow time series covers a two-week period containing some visible anomalous data points, where each time step in the x-axis represents a 3-minute interval. Case 1 shows the overall drop and gradual decrease in flow between time steps 3000 and 3500, and Case 2 shows very low flows over two consecutive days between time steps 3500 and 4500, both of which are clearly distinguishable by human analysts. The performance of the three benchmark models and our RL model in detecting these anomalous periods is compared, where the anomaly points detected by a model are shown as red curves in blue-shaded areas. Overall, DeepAnt and Telemon show poor performance. From Fig. 6(a) and 7 (a), we observe that DeepAnt produces a large number of false alarms by identifying normal flows as anomalies. DeepAnt also has missed detection in Case 2, where it misses the detection of both days with very low flows. Fig. 6(b) and 7 (b) show that Telemon fails to detect the whole anomalous period in Case 1 and detects only a portion of the anomalous period in Case 2. Telemon also has false alarms on the third day of the time series. BreakoutDetection and our model show a good performance in both cases, where they correctly detect the anomalous sequences with only a few missed detections. BreakoutDetection, however, produces several false alarms throughout the normal periods. Our model does not have any false alarms in these two case studies and demonstrates the ability to detect the whole sequence of an anomalous period in a fully unsupervised manner.

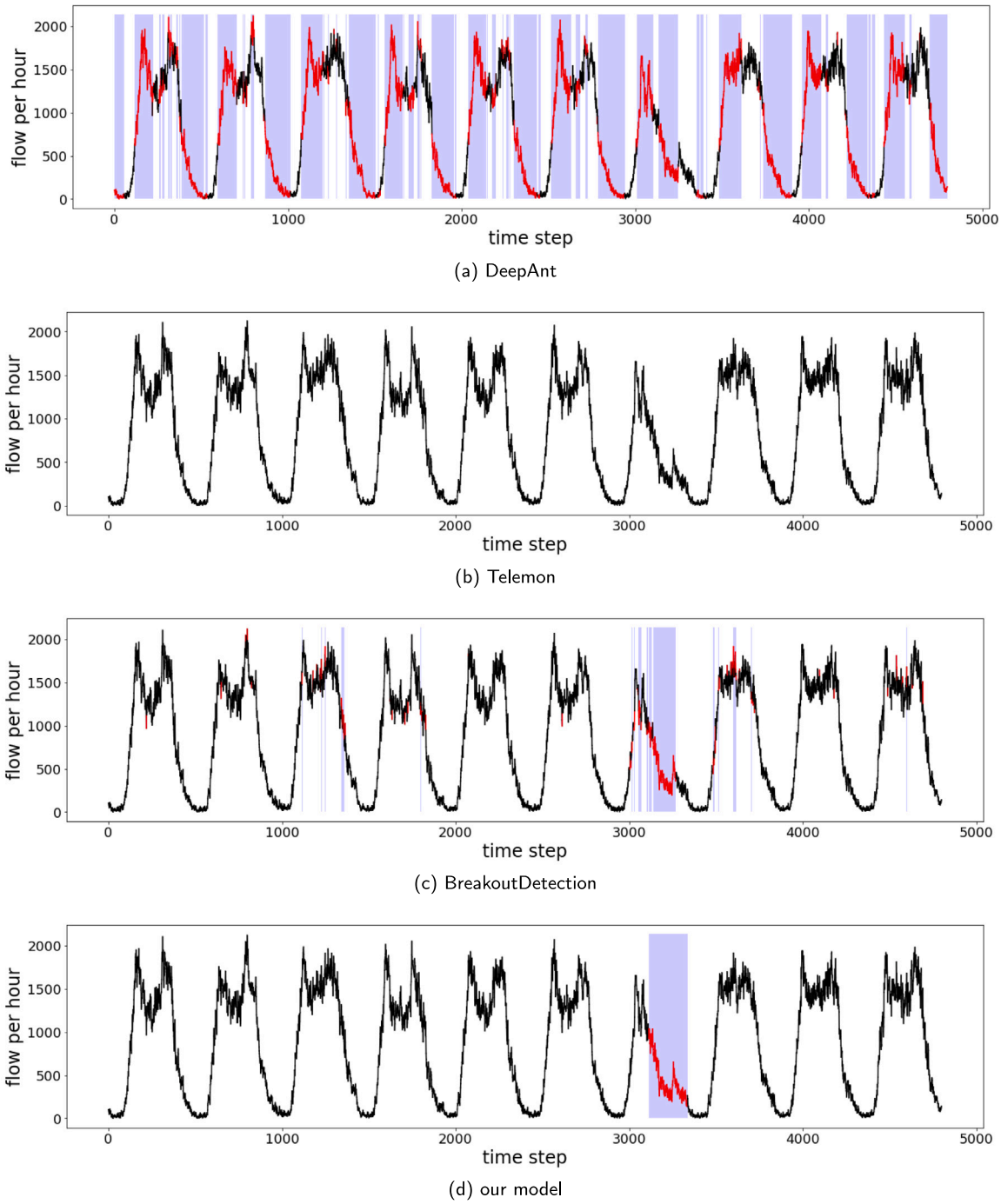


Fig. 6. The comparison of anomaly detection performance for Case 1.

5.3. Evaluation with synthetic data

Since there are no ground-truth labels in our real-world dataset, we generate synthetic flow data with labels of anomalies to evaluate model performance more objectively and quantitatively. and conduct a group of experiments on synthetic data with different settings. We first introduce the method used to generate the synthetic data and then present the evaluation results.

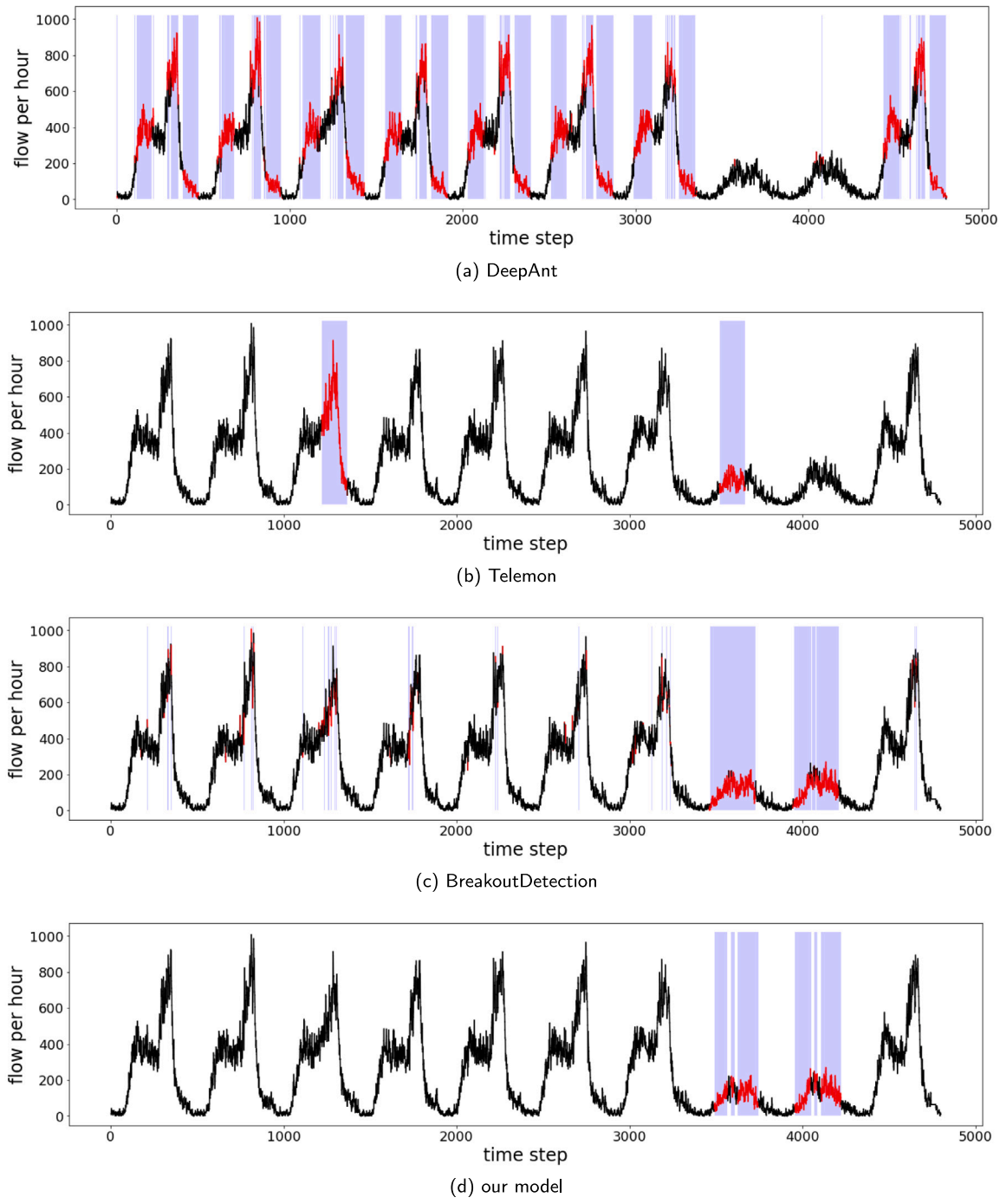


Fig. 7. The comparison of anomaly detection performance comparison for Case 2.

5.3.1. Synthetic data generation

We simulate normal flow time series based on our real-world data combined with synthetically added random fluctuations. Specifically, we extract three-month flow data (weekdays only) from a specific link and group them into 3-minute time-of-day intervals. For each time-of-day interval, we estimate the mean and standard deviation of the corresponding observed flows. We first generate the time series of normal flows by drawing a flow value from the Gaussian distribution with the mean and standard deviation of flows associated with each time-of-day interval. Fig. 8(a) shows an example of the synthetic time series of normal flows

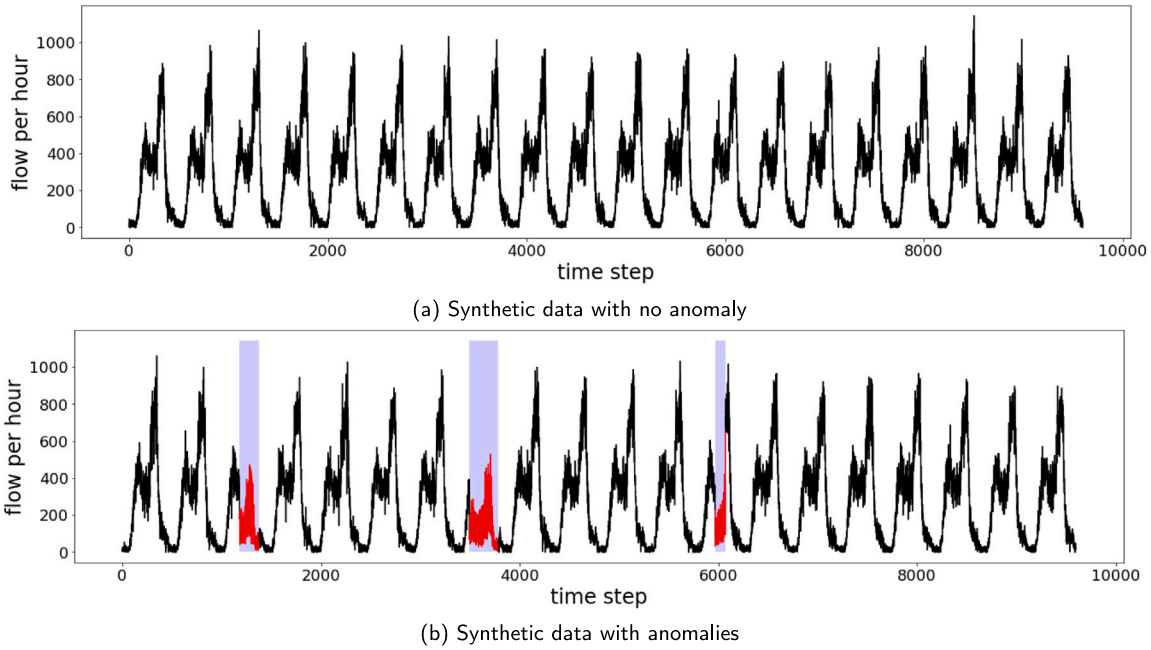


Fig. 8. Example of the synthetic data.

over four weeks (20 days). Then, we randomly add anomaly sub-sequences into the normal time series by first selecting a specific day that will have an anomalous period and then determining the length and severity of the anomaly sequence.

Specifically, we consider four different parameters for synthetic anomaly data generation: dataset size, anomaly density, sequence length, and error rate, as shown in Table 2. The 'dataset size' represents the length of the entire synthetic time series, expressed in months. For instance, a dataset size of 3 indicates a synthetic time series covering 3 months, which includes approximately 28800 time steps (i.e., $28800 = 3$ months$\times 20$ weekdays per month$\times 480$ 3-min time steps per day). The 'anomaly density' indicates the percentage of the days that contain anomaly sub-sequences. For instance, 20% anomaly density for a 3-month time series means that 20% of the 60 weekdays over 3 months, which is 12 days, will have an anomalous sub-sequence. We assume that each selected day will have exactly one anomaly sub-sequence. The 'sequence length' determines the length of an anomaly sub-sequence, expressed as the number of time steps. For instance, a sequence length of 50 will generate an anomaly sub-sequence covering 150 min (i.e., $150 = 50$ time steps $\times$ 3 min per time step). The 'error rate' determines the extent to which an anomalous flow deviates from the normal value in percent, formulated by $(f - \tilde{f})/f \times 100$, where f is the normal flow value and $\tilde{f}$ is the anomalous flow value. For instance, with error rate $e = 40\%$, we replace the original normal flow, f, with the new anomaly flow calculated as $\tilde{f} = (1 - e/100)f = (1 - 0.4)f = 0.6f$. The sequence length and error rate parameters are specified as ranges, where parameter values are randomly sampled from a uniform distribution over a given range, to create variations in the length and severity of an anomaly sub-sequence.

In summary, given the synthetic normal flow time series, which covers a duration specified as 'dataset size', we determine specific days that will have anomalies by randomly assigning them based on 'anomaly density'. For each day with anomalies, we determine the length of the anomalous period by randomly drawing a value from the uniform distribution over the range specified as 'sequence length'. The start time of the anomalous period is selected randomly, and the end time is determined as the sum of the start time and the anomaly sequence length. Since we define the initial state ($s_m = \{o_1, o_2, \ldots, o_m\}$) as normal flows by assuming the first $m-1$ flows are normal and the first $m-1$ actions are zeros as explained in Section 4.2, we prevent the anomalies from occurring during the first $m-1$ time steps of the synthetic series. Once the start and end times of the anomalous time window are determined, we assign an anomalous flow value to each time step within that time window based on the error rate randomly drawn (at each time step) from the uniform distribution over the range specified as 'error rate'. Fig. 8(b) shows an example of the synthetic data with anomalies, where the anomaly sub-sequences are shown as red curves in blue-shaded areas.

We generate a comprehensive set of synthetic time series data by varying each parameter at a time while fixing the other three parameters to default values. Five different values are considered for each parameter. The tested parameter values and default parameter setting are shown in Table 2. With four parameters and five different values for each parameter, we have a total of 20 parameter sets. For each parameter set, we generate five synthetic flow time series under that parameter setting. The performance of a given anomaly detection model is tested using those five time series sets and the average performance is obtained (over the five time series sets) to report the model performance for each parameter set.

Table 2
Parameter settings for synthetic data generation.

Parameters	Values	Unit	Default
Dataset size	1, 3, 6, 9, 12	month	3
Anomaly density	10%, 20%, 30%, 40%, 50%	percentage	20%
Sequence length	[50, 100], [100, 150], [150, 200], [200, 250], [250, 300]	time step	[50, 300]
Error rate	[40, 50], [50, 60], [60, 70], [70, 80], [80, 90]	percentage	[40, 90]

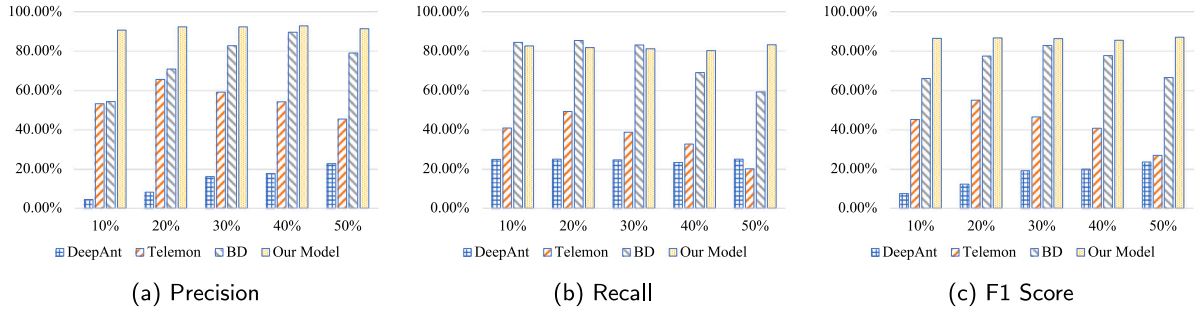


Fig. 9. Performance comparison with different anomaly densities.

5.3.2. Evaluation metrics

The performance of the tested models on the anomaly detection task is evaluated by common metrics used in classification problems (machine learning), including *Precision*, *Recall*, and *F1-score*, which are defined based on the numbers of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) as follows.

- *Precision* is the ratio of correctly detected anomalous flows to all the flows regarded as anomalies. The higher precision indicates the less false alarm.

$$Precision = \frac{TP}{TP + FP}$$

- *Recall* is the ratio of correctly detected anomalous flows to all the flows that are actually anomalous. The higher recall represents the less missed detection.

$$Recall = \frac{TP}{TP + FN}$$

- F1-score combines the precision and recall by computing their harmonic mean. This makes F1-score more comprehensive in terms of evaluating models in classification tasks.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FN + FP}$$

5.3.3. Performance comparison with benchmark models

The performance results for the four tested models on the synthetic data are reported in Figs. 9–12. In general, our model outperforms the other three benchmark models in most of the cases. Our model produces the highest F1 score in 19 out of 20 parameter settings. Among the three benchmark models, the performance of BreakoutDetection is better than that of the two deep learning models (DeepAnt and Telemon). While the prediction accuracy of the two deep learning models was very high, their prediction-based anomaly detection methods did not produce satisfactory results in our experiments.

Considering the anomaly density parameter, when we increase the density, the performance of our model stays stable in all three metrics, as shown in Fig. 9, which demonstrates that our model is robust with the increase in the anomaly portion in the dataset. BreakoutDetection performs well when the anomaly density is within a certain level, e.g., around 30%, where it can achieve better recall when the density is lower and better precision when the density is higher. Telemon has the best performance when the density is around 20%, while all three metrics drop in the other cases. DeepAnt shows an increase in performance with the increase of density, while the overall performance is relatively poor in that the F1 score is only 22% in the best case. In contrast, the precision of our model with various densities is stable at around 90%, and recall is higher than 80%, leading the F1 score to around 85% with different density settings.

In terms of the error rate parameter, as we can see from Fig. 10, the precision is stable with different error rates in our model. The recall is also stable in general but shows a decrease when the error rate is below 50%. This is because small error rates generate anomalies that are less deviated from the normal data, which leads to a greater level of missed detection. When the error rate is greater than or equal to 50%, our model performs well with F1 score higher than 85%. BreakoutDetection achieves a better recall than our model, while the precision is much worse. The performance of Telemon fluctuates with the increase in error rate, while

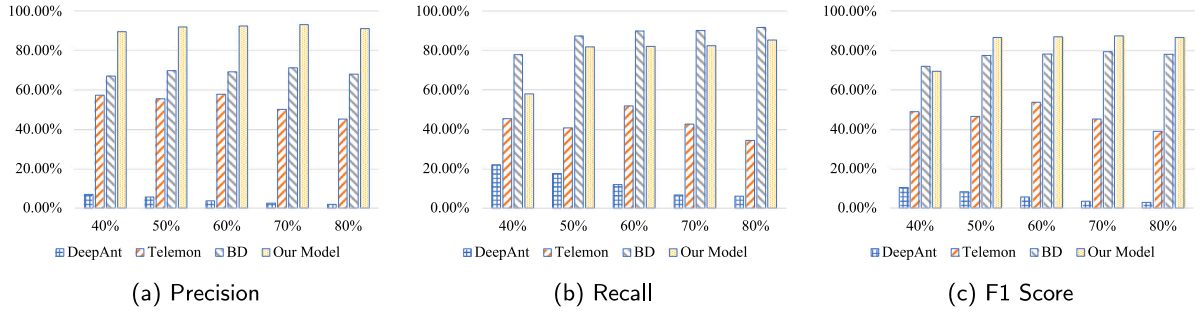


Fig. 10. Performance comparison with different error rates.

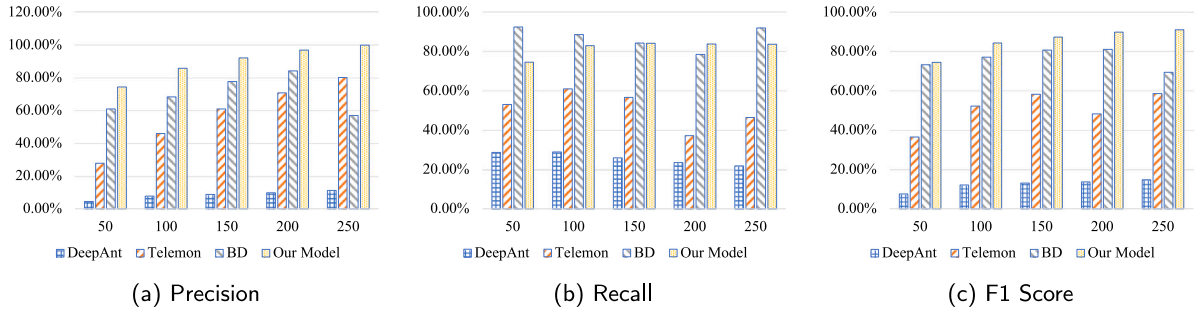


Fig. 11. Performance comparison with different anomaly sequence lengths.

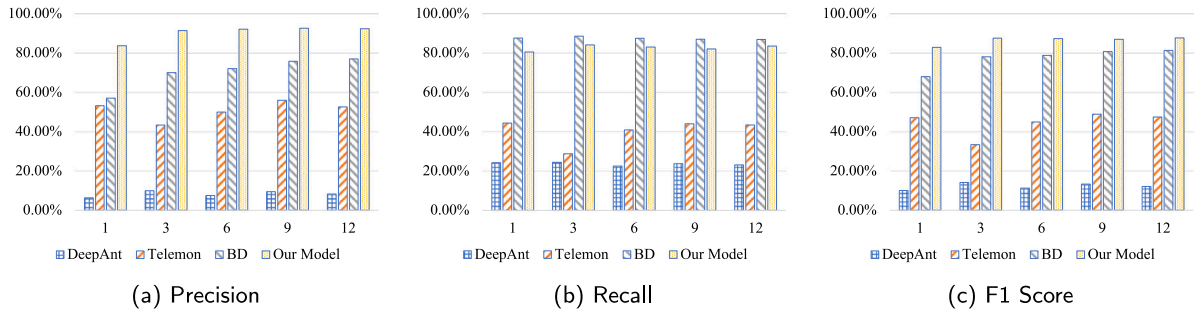


Fig. 12. Performance comparison with different data sizes.

DeepAnt shows a decreasing trend, which is unexpected. The reason might be when the difference becomes larger, the model itself captures such changes and predicts more accurately.

As for the length of the anomaly sequence, the performance of our model increases with the increase in sequence length, as shown in Fig. 11. It is worth noting that, when the sequence length is within [200, 250] points of flow, our model can achieve around 95% of the F1 score, which indicates that our model can capture a long anomaly. Both BreakoutDetection and Telemon have a fluctuating performance with respect to the length of the anomaly sequence, while the DeepAnt shows a slight increase trend with the growth of sequence length. We observe that the BreakoutDetection achieve better recall in some cases, while the precision meanwhile is much worse, which demonstrates the superiority of our model.

Next, we analyse the performance over different data sizes, as shown in Fig. 12. It can be seen from the results that with more input data, the performance is slightly better in our model. But the difference in performance for 3-month data is mild against the one for 12-month data, while the computational cost is much larger for the latter since the running time grows linearly with the size of data. Similarly, BreakoutDetection has a slight increase trend with the increase of data size, while our model outperforms it with respect to F1 score with an increment of around 10%. Again, both Telemon and DeepAnt show a worse performance compared with the other two with less than 50% and 20% F1 scores, respectively.

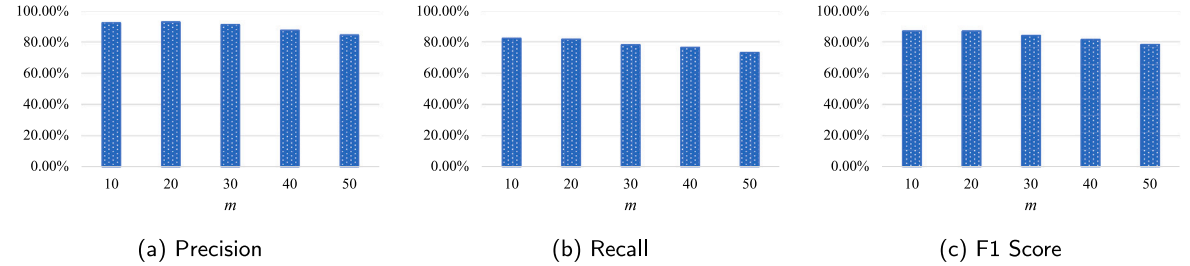


Fig. 13. Performance of our model for different time steps in state.

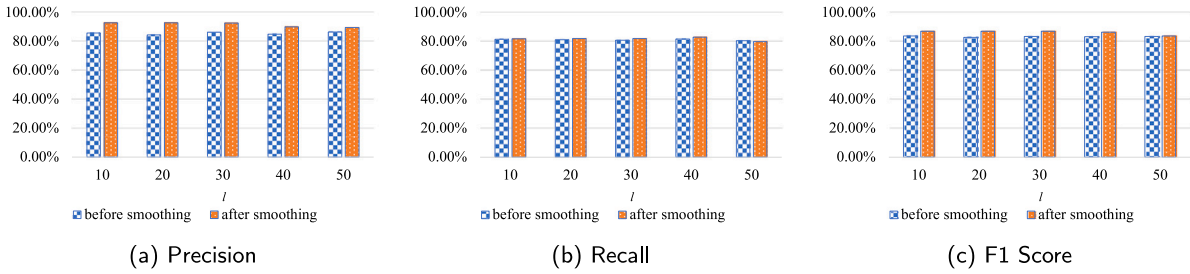


Fig. 14. Performance of our model before and after post-processing smoothing.

5.3.4. *Performance analysis of the proposed RL model*

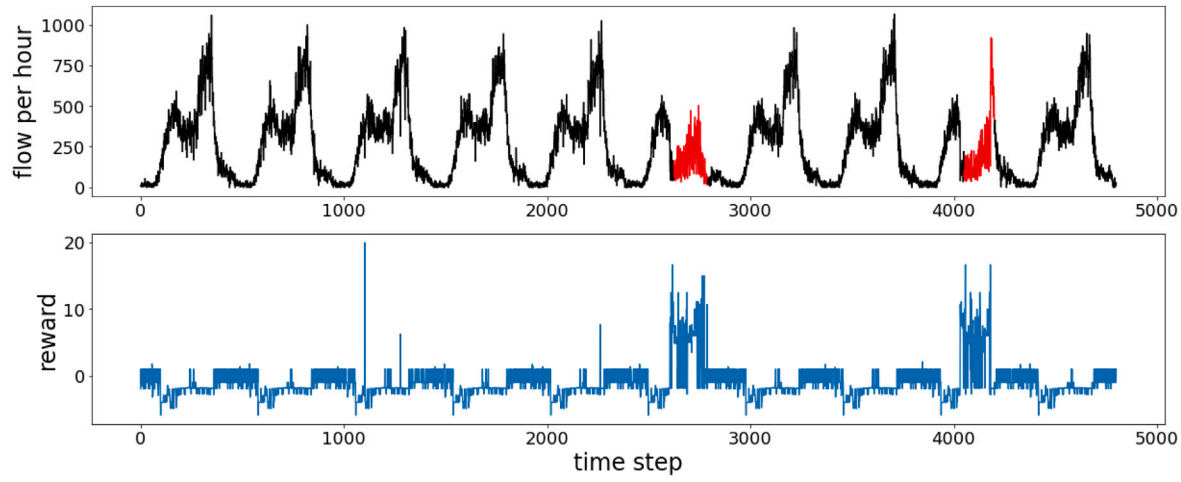
In this set of experiments, we focus on the performance of our own model with different parameter settings. Here, we mainly study the impact of varying two parameters: m and l, which are the number of time steps in the state (s_t) and the size of the sliding window for smoothing in the post-processing, respectively. For each set of experiments, the parameters in Table 2 are set as default.

Firstly, to study how the number of time steps in the state influences the performance of our model, we vary the parameter m from $10, 20, 30, 40,$ to 50. Since each time step refers to a 3-minute aggregation of flow, for the state sequence, we basically consider the flows in the past $30, 60, 90, 120, 150$ minutes. Fig. 13 shows the results of precision, recall, and F1 score of our model. As we can see, all three metrics show a slight decrease in performance as the state includes more time steps. The difference between the performance for $m = 10$ and $m = 20$ is not significant, where in both cases, the F1 scores are around 87%. From the results, we observe that the flow trend for up to 60 minutes is sufficient to indicate whether the current flow is an anomaly or not, and a longer sequence of flows might add irrelevant information, resulting in poorer performance. Consequently, we set the default value of time steps as 20 as shown in Table 2.

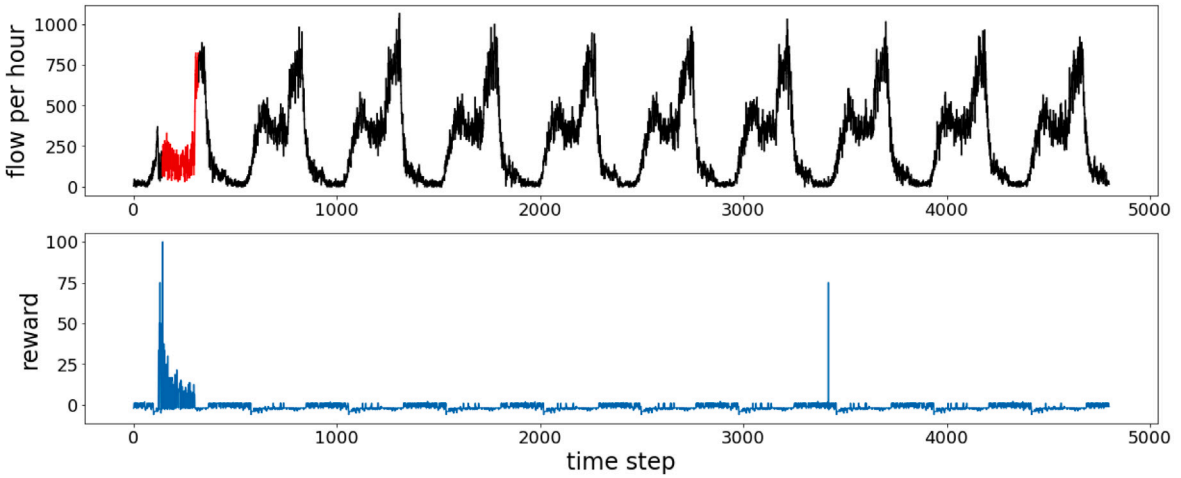
Next, we study the impact of different lengths of the sliding window for the smoothing operation in post-processing. Note that the length of the sliding window is set to be $2l + 1$, and we vary l from $10, 20, 30, 40,$ to 50. Fig. 14 shows the performance results of our model, where the blue bars indicate the performance before the smoothing operation and the other ones represent the performance after the smoothing operation. As we can see from the results, the smoothing operation is helpful to improve the precision and F1 score accordingly, which demonstrates that the smoothing operation is capable of removing false alarms. Regarding different lengths of the sliding window, overall, the influence is not significant, although we can observe a very mild drop in performance in the case $l = 50$.

Next, we investigate the performance of the unsupervised reward learning approach by showing two case studies. We pick two snapshots of flow sequences, where our model detected anomalies correctly, and plot the detected results from our model and the reward values estimated by the proposed unsupervised reward learning. Fig. 15 shows the results of these two cases, where in each case, the top chart shows the flow time series with the anomaly detection results (black for normal points and red for anomalies), and the bottom chart shows the estimated reward for taking action $a = 1$ on each of the flow data points in the top chart (in blue). From the reward function definition in Eq. (11), we know that the reward for labelling a data point as an anomaly ($a = 1$) can be either $1/\delta_t$ (rewarded when the data point is indeed anomalous and its normality score is low, $\delta_t < 1$) or $-\delta_t$ (penalized when the data point is actually normal and its normality score is high, $\delta_t \geq 1$). As such, the bottom chart shows the sequence of the estimated reward value of either $1/\delta_t$ or $-\delta_t$ for each data point, where δ_t is the normality score of each data point reflecting how normal that data point is. The purpose of constructing Fig. 15 is to visually demonstrate how correct and incorrect decisions are captured by the proposed reward function.

From Case A in Fig. 15(a), we observe that there are two anomaly sub-sequences correctly detected by our model (red sections), which are the points at which our model chose action $a = 1$. The reward value for taking action $a = 1$ at each point (blue series) then reveals the clear peaks for those actual anomaly periods, which reward the correct decisions, and lower rewards for all other normal points, which discourage the incorrect decisions. Similarly, Case B in Fig. 15(b) shows one anomaly sub-sequence detected correctly, and we can see a significantly higher reward for that anomaly period. This demonstrates that the proposed reward learning



(a) Case A



(b) Case B

Fig. 15. Estimated rewards for taking action $a = 1$ (detecting as an anomaly) in two synthetic flow time series cases.

Table 3
Performance comparison with threshold-based solution based on Normality Score.

	Precision	Recall	F1 score
$1/\delta \geq 10$	92.71%	18.96%	31.02%
$1/\delta \geq 5$	95.42%	31.80%	41.44%
$1/\delta \geq 1$	14.95%	74.49%	24.91%
Our model	92.94%	82.48%	87.23%

approach indeed creates a data-driven reward in the intended direction such that the model receives the correct rewards or penalties for their actions and effectively identifies anomalies without supervision.

Our final experiment is to analyse the role of RL in our anomaly detection framework and investigate whether we could just use the normality score with some thresholds instead of using RL. From the reward function graphs (blue series) in Fig. 15, we can observe that the positive reward value ($1/\delta_t$) can be used to determine an anomaly as $1/\delta_t$ sharply increases when there are indeed anomalies (red sections). Based on the range of $1/\delta_t$ values observed in Fig. 15, we consider three thresholds for determining an anomaly: $1/\delta_t \geq 10$, $1/\delta_t \geq 5$, and $1/\delta_t \geq 1$, respectively. We then perform experiments by detecting anomalies in synthetic flow time series based only on these thresholds without using RL. For instance, given a threshold of 10, if the calculated normality score at each data point satisfies $1/\delta_t \geq 10$ (equivalently, the normality score is $\delta_t \leq 1/10$), then we mark that point as an anomaly.

Table 3 shows the performance of these threshold-based detection methods and our RL model for comparison. The synthetic data with the default parameters in Table 2 are used for performance evaluation and the performance is averaged over five sets

of synthetic data. The results show that the threshold-based methods perform very poorly, producing low F1 scores ranging from 24.91% to 31.02%, while our RL model produces an F1 score of 87.23%. A larger threshold results in low recall due to more missed detections, and a smaller threshold results in low precision due to more false alarms. This demonstrates the challenge of selecting a suitable threshold. And we observe that the scale of the normality score varies widely across datasets, which makes identifying the right threshold even more difficult. From this experiment, we reaffirm our motivation to pursue a threshold-free anomaly detection model. Our RL model fulfils this need by learning to set reward signals based on data without any external threshold settings.

6. Conclusion

In this work, we propose a Reinforcement Learning model for anomaly detection on traffic flow data, which takes as input a flow time series and outputs the flow sequence with binary indicators showing whether or not the flow value at each time step is an anomaly. We employ the deep reinforcement learning framework, where LSTM networks are used as a function approximator for the Q-function and a deep Q-learning algorithm is applied for policy learning. We propose an unsupervised reward learning approach to define the rewards in a data-driven way, so that the model can learn the criteria for detecting anomalies based on data, without requiring manually specified threshold settings. Overall, our model has three key advantages: (1) it requires no ground-truth labels for model training by an unsupervised reward learning technique; (2) the model can automatically identify the anomalies without any threshold settings as used in most of the existing work; (3) it can detect the anomalies for the input flow sequence with different patterns captured from different types of roads. We conduct a comprehensive set of experiments on both real-world data and synthetic data with a comparison against three outstanding models. The experimental results show that our model outperforms all the tested benchmark models in different parameter settings, which demonstrates the superiority of our model and the automation capability achieved by the reinforcement learning model.

In future work, we consider incorporating multi-source and multi-modal data, e.g., public transport data and vehicle data from different sensors and develop an AI agent based on reinforcement learning to detect the anomalies from multi-modal data, simultaneously. As we can imagine that there could be some associations among the anomalies from multi-modal data in the whole traffic network. Capturing the anomalies in a multi-modal view can help to improve the reliability and confidence of the model. On the other hand, this multi-modal view of anomaly detection can provide significant information on how different traffic networks are influenced by considering various factors such as time of delay.

CRediT authorship contribution statement

Dan He: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft. **Jiwon Kim:** Conceptualization, Methodology, Formal analysis, Investigation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Hua Shi:** Formal analysis, Investigation, Writing – review & editing. **Boyu Ruan:** Methodology, Software, Formal analysis.

Acknowledgements

The authors gratefully acknowledge the support of the Australian Research Council (ARC), Queensland Department of Transport and Main Roads, and Transmax Pty Ltd under the ARC Linkage Project on Real-time Analytics for Traffic Management (LP180100018).

References

Angiulli, F., Basta, S., Pizzuti, C., 2005. Distance-based detection and prediction of outliers. IEEE Trans. Knowl. Data Eng. 18 (2), 145–160.
Angiulli, F., Fassetti, F., 2007. Detecting distance-based outliers in streams of data. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. pp. 811–820.
Berndt, D.J., Clifford, J., 1994. Using dynamic time warping to find patterns in time series. In: KDD Workshop, vol. 10 no. 16. Seattle, WA, USA:, pp. 359–370.
Blázquez-García, A., Conde, A., Mori, U., Lozano, J.A., 2020. A review on outlier/anomaly detection in time series data. arXiv preprint arXiv:2002.04236.
Blázquez-García, A., Conde, A., Mori, U., Lozano, J.A., 2021. A review on outlier/anomaly detection in time series data. ACM Comput. Surv. 54 (3), 1–33.
Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M., 2015. Time series analysis: forecasting and control. John Wiley & Sons.
Braei, M., Wagner, S., 2020. Anomaly detection in univariate time-series: A survey on the state-of-the-art. arXiv preprint arXiv:2004.00433.
Buda, T.S., Caglayan, B., Assem, H., 2018. Deepad: A generic framework based on deep learning for time series anomaly detection. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp. 577–588.
Budalakoti, S., Srivastava, A.N., Akella, R., Turkov, E., 2006. Anomaly detection in large sets of high-dimensional symbol sequences.
Chalapathy, R., Chawla, S., 2019. Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.
Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM Comput. Surv. 41 (3), 1–58.
De Haan, L., Ferreira, A., Ferreira, A., 2006. Extreme Value Theory: An Introduction, vol. 21. Springer.
de La Bourdonnaye, F., Teuliere, C., Chateau, T., Triesch, J., 2017. Learning of binocular fixations using anomaly detection with deep reinforcement learning. In: 2017 International Joint Conference on Neural Networks. IJCNN, IEEE, pp. 760–767.
Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S., 2002. A geometric framework for unsupervised anomaly detection. In: Applications of Data Mining in Computer Security. Springer, pp. 77–101.
Goldstein, M., Uchida, S., 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PLoS One 11 (4), e0152173.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.
Gupta, M., Gao, J., Aggarwal, C.C., Han, J., 2013. Outlier detection for temporal data: A survey. IEEE Trans. Knowl. Data Eng. 26 (9), 2250–2267.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Hou, X., Zhang, L., 2007. Saliency detection: A spectral residual approach. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition. Ieee, pp. 1–8.
Huang, C., Wu, Y., Zuo, Y., Pei, K., Min, G., 2018. Towards experienced anomaly detector through reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence.
Hundman, K., Constantinou, V., Laporte, C., Colwell, I., Soderstrom, T., 2018. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 387–395.
Jagadish, H., Koudas, N., Muthukrishnan, S., 1999. Mining deviants in a time series database. In: Proceedings of the 25th International Conference on Very Large Data Bases. pp. 102–113.
Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
Lai, K.-H., Zha, D., Xu, J., Zhao, Y., Wang, G., Hu, X., 2021. Revisiting time series outlier detection: Definitions and benchmarks.
Lane, T., Brodley, C.E., et al., 1997. Sequence matching and learning in anomaly detection for computer security. In: AAAI Workshop: AI Approaches to Fraud Detection and Risk Management. Providence, Rhode Island, pp. 43–49.
Liu, Y., Chen, X., Wang, F., Yin, J., 2009. Efficient detection of discords for time series stream. In: Advances in Data and Web Management. Springer, pp. 629–634.
Lu, W., Ghorbani, A.A., 2008. Network anomaly detection based on wavelet analysis. EURASIP J. Adv. Signal Process. 2009, 1–16.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.
Munir, M., Siddiqui, S.A., Dengel, A., Ahmed, S., 2018. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. Ieee Access 7, 1991–2005.
Nairac, A., Townsend, N., Carr, R., King, S., Cowley, P., Tarassenko, L., 1999. A system for the analysis of jet engine vibration data. Integr. Comput.-Aided Eng. 6 (1), 53–66.
Pan, X., Tan, J., Kavulya, S., Gandhi, R., Narasimhan, P., 2010. Ganesha: Blackbox diagnosis of mapreduce systems. ACM SIGMETRICS Perform. Eval. Rev. 37 (3), 8–13.
Pang, G., Shen, C., Cao, L., Hengel, A.V.D., 2021. Deep learning for anomaly detection: A review. ACM Comput. Surv. 54 (2), 1–38.
Rebbapragada, U., Protopapas, P., Brodley, C.E., Alcock, C., 2009. Finding anomalous periodic time series. Mach. Learn. 74 (3), 281–313.
Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C., Kou, X., Xing, T., Yang, M., Tong, J., Zhang, Q., 2019. Time-series anomaly detection service at microsoft. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 3009–3017.
Rosner, B., 1983. Percentage points for a generalized ESD many-outlier procedure. Technometrics 25 (2), 165–172.
Sequeira, K., Zaki, M., 2002. Admit: anomaly-based data mining for intrusions. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 386–395.
Siffer, A., Fouque, P.-A., Termier, A., Largouet, C., 2017. Anomaly detection in streams with extreme value theory. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1067–1075.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.
Twitter, 2015. Introducing practical and robust anomaly detection in a time series. https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.
Vallis, O., Hochenbaum, J., Kejariwal, A., 2014. A novel technique for {long-term} anomaly detection in the cloud. In: 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14).
Watkins, C.J., Dayan, P., 1992. Q-learning. Mach. Learn. 8 (3), 279–292.
Wu, T., Ortiz, J., 2021. RLAD: Time series anomaly detection through reinforcement learning and active learning. arXiv preprint arXiv:2104.00543.
Yamanishi, K., Takeuchi, J.-I., Williams, G., Milne, P., 2004. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Min. Knowl. Discov. 8 (3), 275–300.
Yankov, D., Keogh, E., Rebbapragada, U., 2008. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl. Inf. Syst. 17, 241–262.
Yu, M., Sun, S., 2020. Policy-based reinforcement learning for time series anomaly detection. Eng. Appl. Artif. Intell. 95, 103919.
Yu, Y., Zhu, Y., Li, S., Wan, D., 2014. Time series outlier detection based on sliding window prediction. Math. Probl. Eng. 2014.
Zhang, Y., Ge, Z., Greenberg, A., Roughan, M., 2005. Network anomography. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement. pp. 30–30.