國立高雄大學資訊工程學系

碩士論文

應用元強化學習於雲端應用服務快速在線異常檢測

Applying Meta-Reinforcement Learning to Cloud Application Services for Fast Online Anomaly Detection

研 究 生： 陳冠儒

指導教授： 張保榮 教授

中華民國一百一十二年一月

國立高雄大學研究生學位論文審定書

本校資訊工程學系

研究生 陳冠儒 M1095513 所提論文

應用元強化學習於雲端應用服務快速在線異常檢測

Applying Meta-Reinforcement Learning to Cloud Application Services for Fast Online Anomaly Detection

經本委員會審查並舉行口試，符合碩/博士學位論文標準。

學位考試委員簽章：

（召集人）王隆仁　　沈志隆

張保榮

指導教授 張保榮

系主任/所長 資訊工程學系系主任張保榮

應用元強化學習於雲端應用服務快速在線異常檢測

指導教授：張保榮 教授
國立高雄大學資訊工程學系

研究生：陳冠儒
國立高雄大學資訊工程學系碩士班

摘要

雲端運算與網路服務已成為現在科技服務的主流，劇增的用戶對雲端負載增加壓力，導致資料中心發生非預期故障。雖然已有監測工具即時檢測，但大多是以被動式方法判定故障。檢測雲端服務的異常並不容易，使用者的行為會隨時變動，導致故障檢測模型在短時間內失去作用。因此，檢測模型除了具備準確性，也需要快速適應當下的資料分佈。本研究使用深度強化學習結合模型無關的元學習訓練框架生成在線預測模型，對伺服器資源使用率之時間序列進行異常檢測。模型無關的元學習建立多個子任務學習不同異常行為間的隱含表徵。每個子任務包含一個強化學習環境與一個代理者執行決策，最終損失透過置信區域策略優化方法優化並更新參數得到一個初始模型。在最終佈署時，我們加入目標裝置的資料進行少次數的梯度更新微調模型參數，即可快速適應當前的資料，減少管理人員更新模型參數的額外成本。本方法在準確度方面提升了 1.4 倍，並且減少了 4 倍的模型佈署時間。

關鍵字：雲端運算、非預期故障、深度強化學習、元學習、模型無關的元學習、置信區域策略優化。

Applying Meta-Reinforcement Learning to Cloud Application Services for Fast Online Anomaly Detection

Advisor: Dr. Bao-Rong Chang
Department of Computer Science and Information Engineering
National University of Kaohsiung

Student: Guan-Ru Chen
Department of Computer Science and Information Engineering
National University of Kaohsiung

ABSTRACT

Cloud computing and network services have become mainstream science and technology services. As more users put more pressure on the cloud load, unexpected failures occur in the data center. Although there are monitoring tools for real-time detection, most of them use passive methods to determine faults. It is not easy to detect the anomaly of cloud services, and the user's behavior will change at any time, leading to the failure detection model losing its function quickly. Therefore, in addition to accuracy, the detection model must quickly adapt to the current data distribution. In this study, we use Deep Reinforcement Learning combined with the Model Diagnostic Meta-Learning (MAML) training framework to generate an online prediction model to detect exceptions in the time series of server resource utilization. The MAML algorithm builds multiple subtasks to learn the implicit representation of different abnormal behaviors. Each subtask contains a reinforcement learning environment and an agent to execute decisions. The final loss is optimized by Trust Region Policy Optimization (TRPO), and the parameters are updated to obtain an initial model. In the final deployment, we add the data of the target device to update the model parameters by gradient a few times, which can quickly adapt to the current data and reduce the extra cost for managers to update the model parameters. This method improves the accuracy by 1.4 times and reduces the model deployment time by 4 times.

Keywords: Cloud Computing, Unexpected Failures, Deep Reinforcement Learning, Meta Learning, MAML, TRPO.

誌謝

感謝張老師在兩年期間給我的紮實訓練。首先，每年申請計畫時，老師會以過去的經驗引導我加強內容的呈現，提高計畫通過的機會。再來，每年都會任命我為其中一堂課的助教，藉由安排課程、準備實驗以及課程教學方式訓練我們處理事情的能力和紮根基礎知識。相信這些技能在未來職場上一定受用。

在畢業之際，要感謝的是在我研究所的學長姐們郁傑、炯霖、函霖，在我初來乍到時不厭煩的教我電腦裝修與課業知識，讓我能在短時間內適應研究所的生活。也感謝學弟翔宇、易儒、富洋能夠接替我的工作，讓我能在撰寫論文期間無後顧之憂。再來，我由衷的感謝和我從大學到研究所共同打拼的同學佳衛，有你的管理實驗室帳務與報帳工作，讓我很放心處理我份內的事情。當我們忙碌到深夜時你也可以陪我打屁聊天，有事情時你也會主動詢問幫忙和提供建議，很感謝這三年有你在的研究所生活。

最後，很感謝讀研究所期間家裡的支持，不會限制我的選擇。當我壓力大時和你們聊天讓我身心得到釋放，周末回家時也讓我好好充電與鼓勵。有了你們的支持，才會有今天順利畢業的我。

Directory

List of Tables

List of Figures

Chapter 1. Introduction

Cloud computing technology integrates many computing resources, virtualizes them, and has become a medium for providing services for applications. Users can obtain on-demand computing through the Internet. Resource integration management reduces maintenance costs, integrates multiple heterogeneous systems, and provides powerful computing support for particular tasks. As the complexity of the cluster increases, there is always a problem when a crash or downtime occurs. The data center monitors the running status of several servers around the clock, including the logs of processors, memories, disks, and network devices, and issues warnings when abnormal conditions occur. However, the dynamic nature of cloud services makes failures challenging to predict, and many normal behaviors are constantly being redefined. A proactive approach is needed to reduce the impact of failures.

Deep learning has achieved good results in many domains, but it requires a lot of staffing to generate labeled data for training to maintain accuracy. When the data distribution changes, the model will fail. Among them, time series anomaly detection is a big challenge. First, outlier samples are infrequent, often leading to class imbalance, and second, continuous anomalies often occur, making anomalous patterns challenging to discern. In the resource usage sequence of the cloud server system, it can be known that when the user behavior changes, the data distribution changes, making the model unable to adapt immediately. If a high-risk application runs, a failure can have a large impact.

In recent years, the rapid development of reinforcement learning algorithms has dramatically improved the problem of difficult identification of abnormal patterns. Deep reinforcement learning can learn through interaction with the environment and adjust model parameters through experience to form an optimal strategy. However, reinforcement learning is strongly dependent on the environment. When the environment changes, the previous optimal strategy will also be invalid, and it will be challenging to adapt to the complex environment changes in the cloud. For this problem, we incorporate Meta-Learning. The Meta-Learning algorithm is dedicated to the rapid learning of a small number of samples, using past experience to quickly and accurately learn a new and small amount of data. MAML (Model-Agnostic Meta-Learning) is one of the algorithms of Meta-Learning. It is characterized in that it has nothing to do with the model and focuses on learning the common representation between various tasks. We study the application of this algorithm to adapt to abnormal patterns to solve the above problems quickly.

The research object of this paper is company-A's cloud application service, which includes the joint operation and maintenance of multiple virtual hosts. The company uses Zabbix Server to monitor the values of various services and sets a warning to notify the administrator. However, this solution can only notify devices with high usage rates and cannot reflect the overall failure and the actual cause of the abnormality. Therefore, this study first analyzes Zabbix Server monitoring data, uses sliding windows to generate time series data and abnormal labels, and uses TSFEL and PyOD python packages to extract transferable meta-features. Next, we use the MAML-RL algorithm to generate different subtasks for training during the training phase and record the policy loss and network parameters. In the outer loop, we use TRPO (Trust Region Policy Optimization) to find the optimal strategy to maximize the reward of the decision to generate an initial model. Finally, an online prediction model is quickly obtained by using the target data set for small-step training.

Chapter 2. Related Work

2.1 Literature review

The increase in the scale of the cluster system leads to a sharp increase in the failure rate. The operating cost can be significantly reduced if the machine can alert before the failure [22]. Current research is divided into time series forecasting, anomaly detection, and log analysis methods [23]. Many of them use the server's resource usage (CPU, RAM...etc.) as the primary goal. The standard approach of using historical data to train a model offline can fail in dynamic environments where the definition of normal behavior undergoes concept drift over time [6] and may invalidate the model. Therefore, an important topic is how to effectively and long-term adapt to complex time series.

Reinforcement learning has achieved good results in the field of control. Some studies have integrated reinforcement learning into time series anomaly detection. It can be found that compared with general methods, the accuracy has leaped forward. [2] It is confirmed that reinforcement learning has a certain level of ability to identify abnormalities in time series, but online adaptability is still an important issue. Meta-learning has the advantages of few-shot learning and excellent adaptability and has strong potential in anomaly detection. Therefore, some studies have used the MAML method for anomaly detection with small data. [24], It is also confirmed that Few-shot learning using meta-policies is significantly better than other learning frameworks. Meta-ADD[3] proves the transferability of meta-strategy and meta-features in anomaly detection and also adds an active learning method to improve the model's accuracy.

2.2 CUDA

CUDA[25] is the Compute Unified Device Architecture abbreviation launched by NVIDIA in 2006. Acceleration can be performed with current consumer graphics cards and professional graphics cards. Deep learning uses a large number of tensor operations in the training of neural networks. The training time can be significantly reduced if parallel computing technologies such as CUDA are used for acceleration. Currently, mainstream deep learning frameworks such as Tensorflow and Pytorch also use CUDA in large numbers.

2.2 Anaconda

Anaconda is a popular Python data science development platform and an open-source and user-friendly platform. Anaconda currently includes more than 8,000 open-source data science suites and machine learning libraries and is also compatible with Windows, Linux, and other operating systems. Conda, as an open-source package management tool, can not only be used to install additional basic packages but also as a virtual environment management function so that users can isolate different environments to perform experiments, as shown in Figure 1.

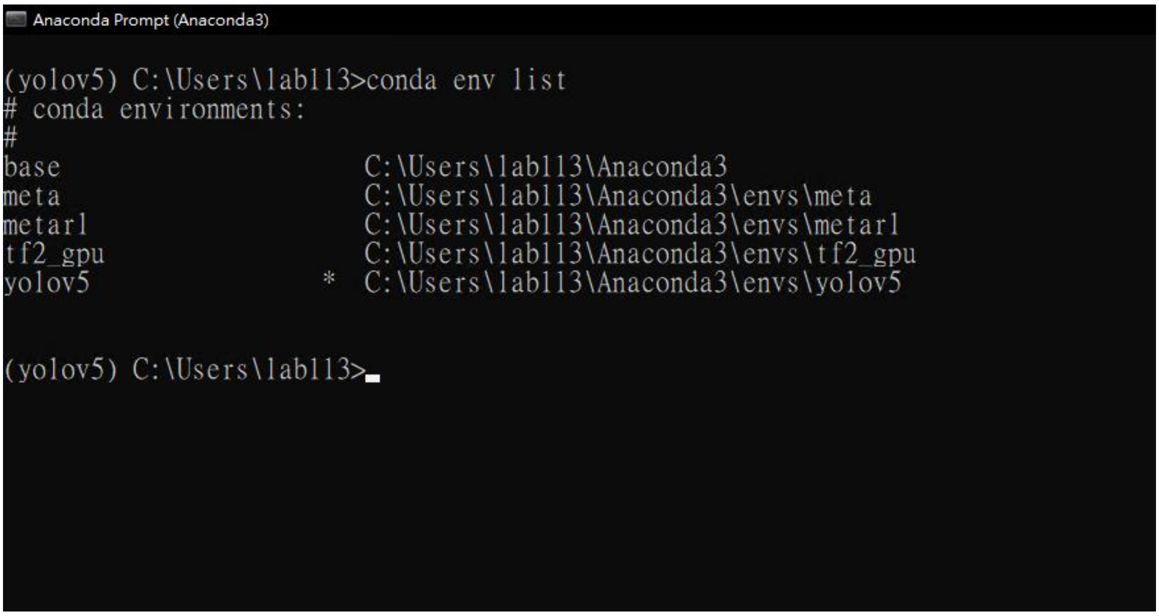


Figure 1. Conda virtual environment management

In addition to package management, the platform can start an external IDE and allow the direct use of a custom virtual environment in the editor. Common code editors such as Jupyter Notebook, Jupyter Lab, and Visual Studio Code are all supported, as shown in Figure 2.

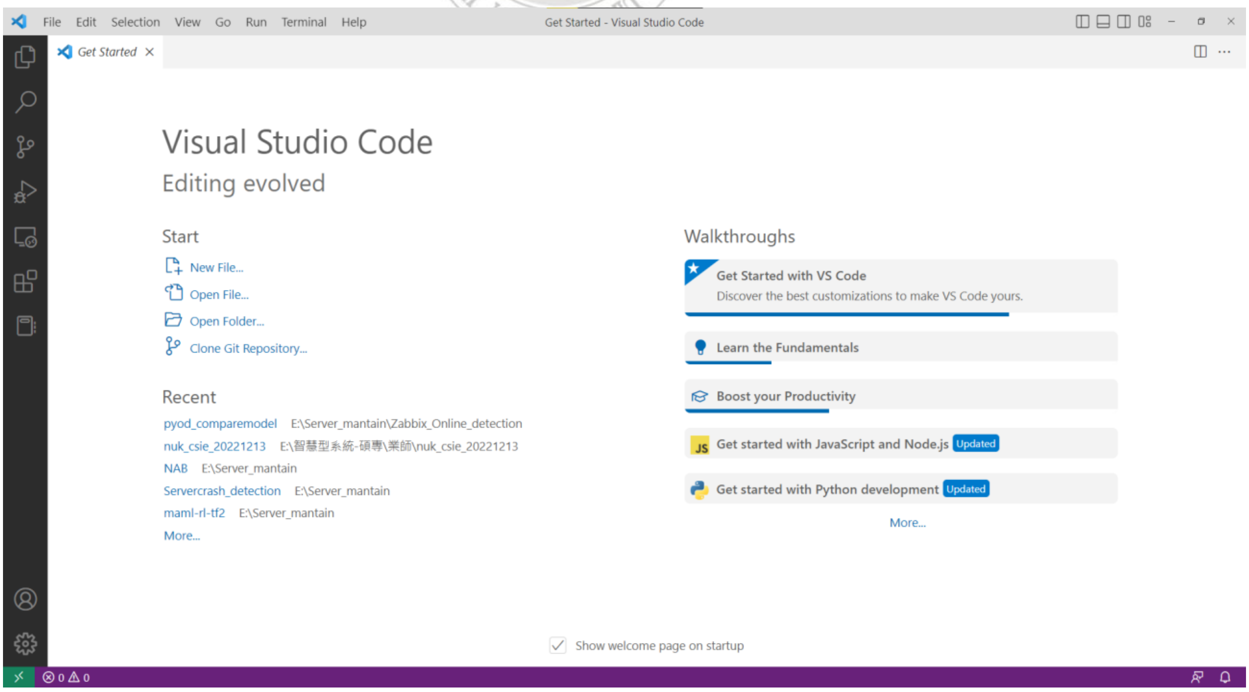


Figure 2. Visual Studio Code

2.3 Pytorch

In early 2017, another set of deep learning frameworks, PyTorch, based on Torch, was open-sourced by Facebook. However, the deep learning framework in the past was more complex and challenging for beginners to adapt. Therefore, in 2017, PyTorch, led by Adam Paszke, Sam Gross and Soumith Chintala, and many Facebook researchers, appeared in the public eye. Pytorch is a Python-first deep learning framework. It is designed for the language feature of Python, so when you use it, you will find that it supports Python very well and can be combined with various Libraries.

2.4 learn2learn

learn2learn[9] is a Python library for meta-learning research. Its bottom layer uses the PyTorch framework as a neural network operation and is compatible with extension functions based on the PyTorch framework, such as torchvision, torchaudio, torchtext, and cherry. This library also provides functions such as data sets, meta-learning algorithms, and optimizers for some famous few-shot tasks. Studies provide meta-learning algorithms for vision, text, and reinforcement learning.

2.5 Time Series Anomaly Detection

Time series is a series of data within a fixed time interval, usually a continuous value. According to the number of variables, it can be divided into univariate time series and multivariate time series [26]. Time series anomaly detection is to identify potential anomalies in a sequence. However, the anomalies of time series are complex and changeable, and sometimes a series of complex continuous time anomalies appear, which makes detection algorithms difficult to identify. Therefore, anomaly detection in time series is challenging, and a huge price will be paid if the prediction is inaccurate. There are many methods used to detect anomalies today. Below we give examples and illustrate the advantages and disadvantages of the methods.

(1) Traditional statistical methods

Anomaly detection based on statistics is the earliest method used. It assumes that the target data is typically distributed. When the observed data exceeds three times the standard deviation, it is judged as abnormal. This method is simple and intuitive but fails when the data are not normally distributed. Furthermore, if the data contains high-dimensional data points, this method cannot identify primarily spatial anomalies.

(2) Supervised learning methods

In recent years, deep learning has achieved good results in different tasks. In anomaly detection, features are usually extracted from sequences to identify anomalies, and binary or multi-category classifications are performed on anomalies. Supervised learning must rely on a large amount of labeled data to train the model. However, obtaining enough data and labels in the real world is complex. Furthermore, the proportion of abnormal and normal data categories is seriously unbalanced, leading to poor performance of the trained classifier.

(3) Semi-supervised learning method

Based on the autoencoder method, which includes a symmetrical network structure and a hidden vector structure, the model learns the target data distribution by restoring the input, as shown in Figure 3. Since the scarcity of positive samples often limits anomaly detection, autoencoders can use model features to learn normal data distributions and determine those with significant restoration errors as anomalies. This method has achieved great success in some fields but will fail in cloud facility maintenance due to rapid environmental changes.

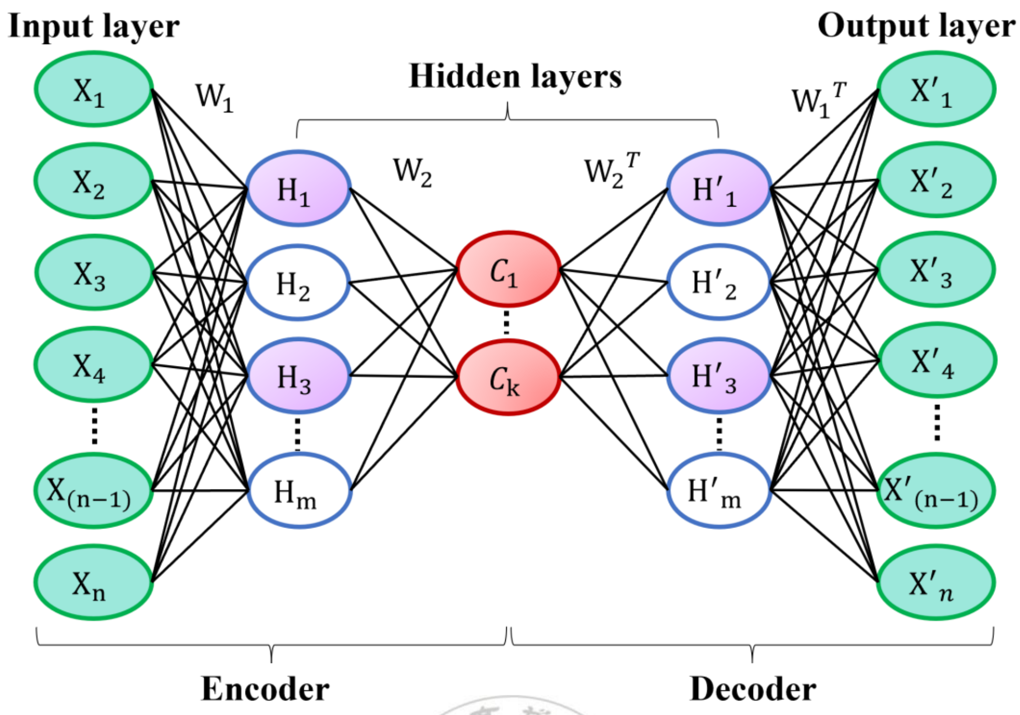


Figure 3. Autoencoder architecture

2.6 Reinforcement Learning

Reinforcement learning is an algorithm that can modify an agent's policy by interacting with the environment. Anomaly detection in time series formulates this problem as a Markov decision process (MDP), which consists of a loop of a policy network, feedback, and environment state, as shown in Figure 4. The agent must maximize reward by learning a control policy. Its self-improvement property solves the problem that the sequence does not have a clear normal pattern. However, the excessive dependence of reinforcement learning on the environment leads to the need to re-simulate the process tens of thousands of times when updating the model, which increases the difficulty.

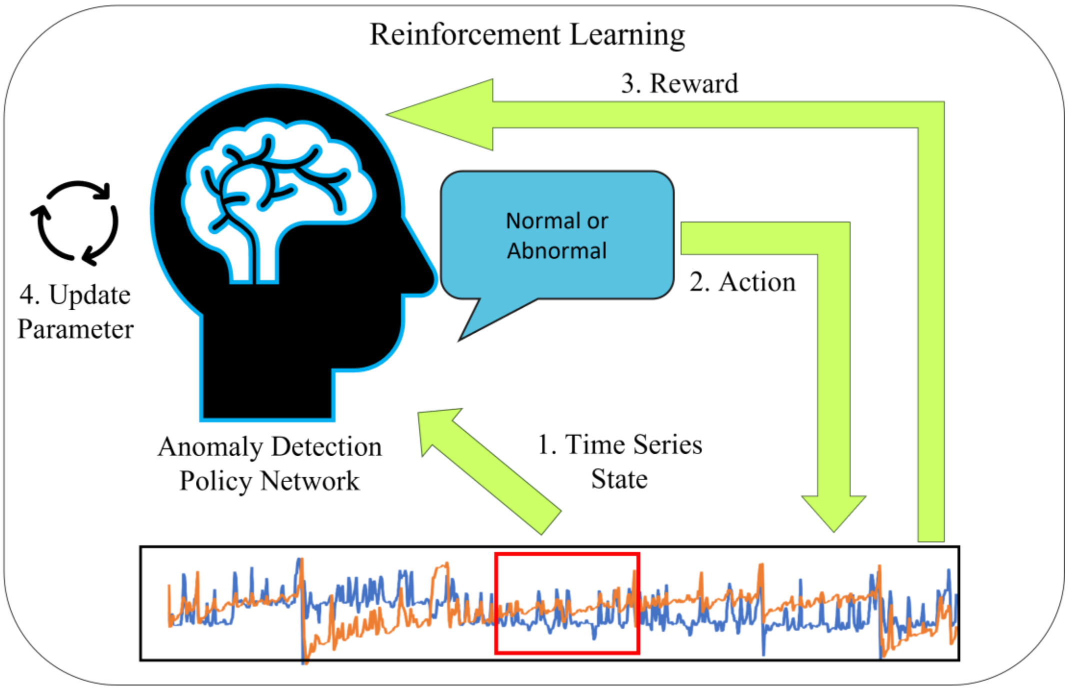


Figure 4. Reinforcement Learning

2.7 Meta-Learning

The biggest problem with deep learning is the demand for data volume, which requires many samples to learn to identify several objects to simulate human learning ability. It is difficult to obtain an accurate model with limited resources. Meta-learning is a technique that has developed rapidly in recent times. It hopes to give the model the ability to “learn how to learn” and quickly learn new tasks from experience. Human beings have had many advantages since birth; they only need to see an object a few times to tell the difference. Therefore, compared to machines, 「Task」 is the central core of this algorithm. Meta-learning can learn through the classification experience of different tasks and quickly adapt to new tasks through prior knowledge. The following section will introduce the classic algorithm Model-Agnostic Meta-Learning (MAML) from meta-learning, which is also the primary training framework used in this study.

2.8 Model-Agnostic Meta-Learning(MAML)

Model-Agnostic Meta-Learning (MAML) [5] is a classic algorithm because it has nothing to do with the model. It can be regarded as a framework for the model to learn by itself. Therefore, it is often used to join other deep learning models, and even reinforcement learning can be seamlessly embedded in it. MAML performs classification training through multiple small subtasks and generates an initial model that can quickly adapt to new tasks, as shown in Figure 5. Because of its fast adaptability and model-independent advantages, we apply it to the cloud application anomaly detection system. In addition to having the self-correction ability of reinforcement learning, it can quickly adapt when the cloud environment changes rapidly.

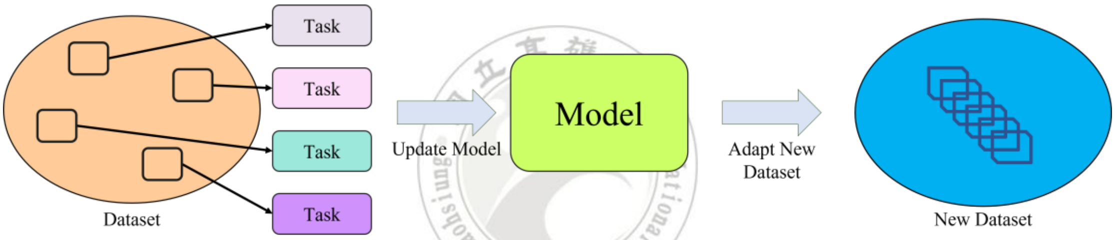


Figure 5. Model-Agnostic Meta-Learning (MAML) algorithm flow

Chapter 3. Research Method

In this study, we combined the cluster of company-A and established an anomaly detection system. The cluster uses Hadoop and Spark frameworks; the data sources are all streaming data captured through HDFS. This system first extracts past hardware resource usage data for sliding window marking, uses PyOD and TSFEL to extract essential features of the sequence, and then conducts Meta-Reinforcement Learning training and testing to obtain an initial model. We update the small step size parameters for the detection object and deploy them to the cluster. Finally, the alarm system will ask whether there is any abnormality in the prediction results of the model at any time. When the system finds an impending failure, it will immediately notify the management personnel to deal with it to reduce unexpected accidents, as shown in Figure 6.

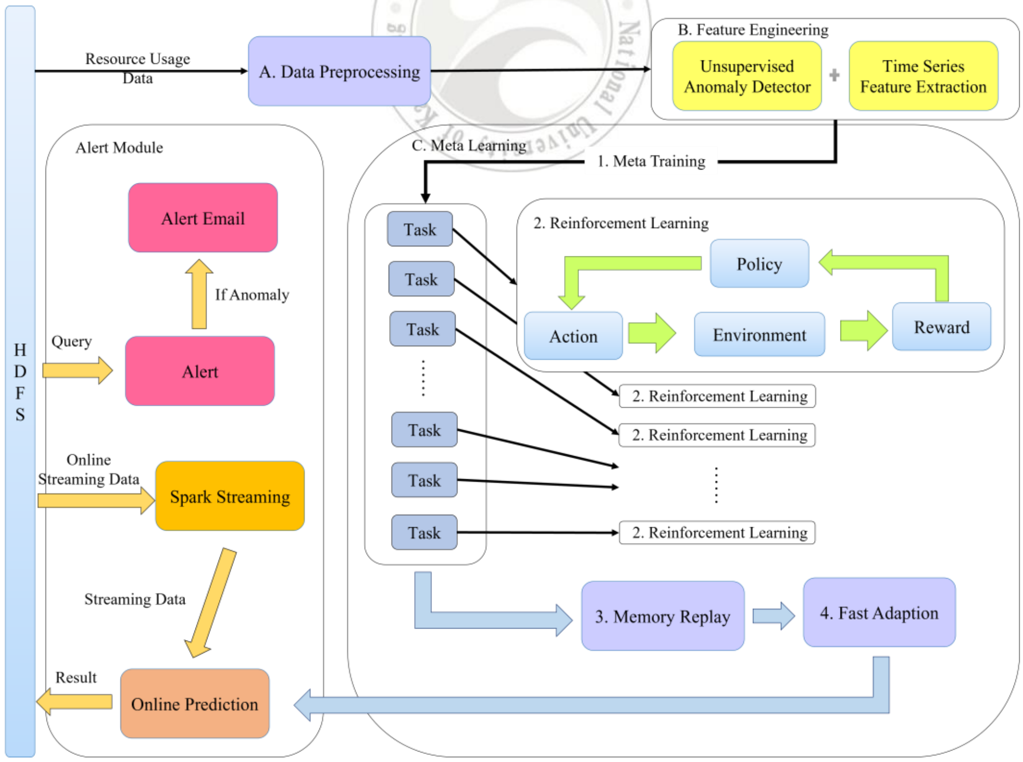
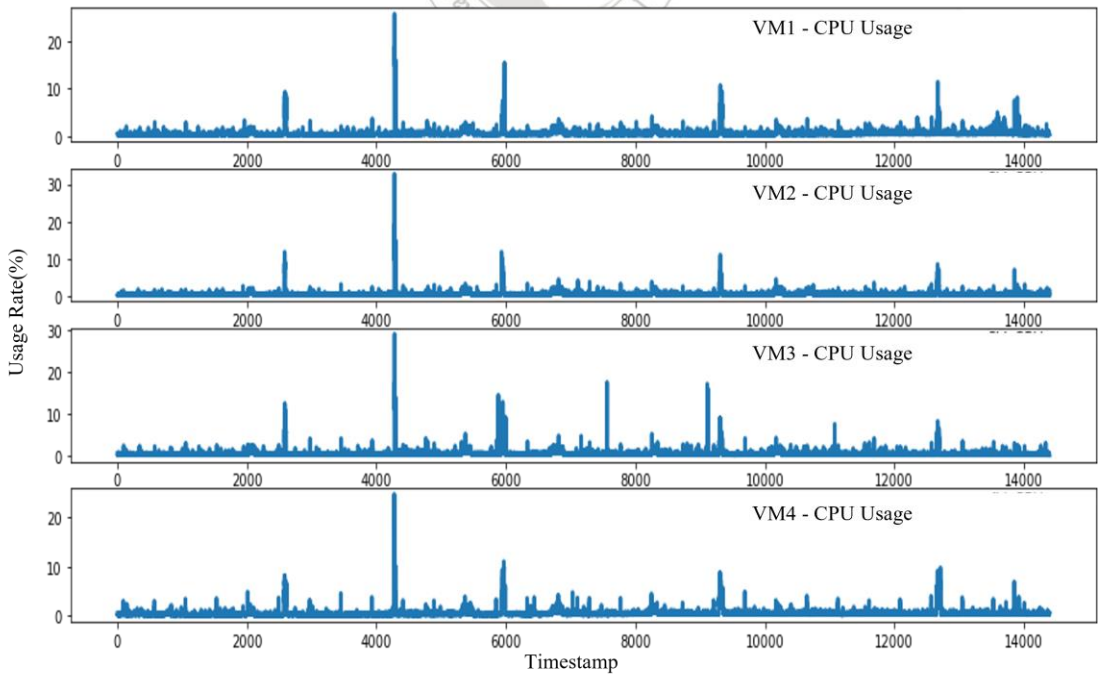


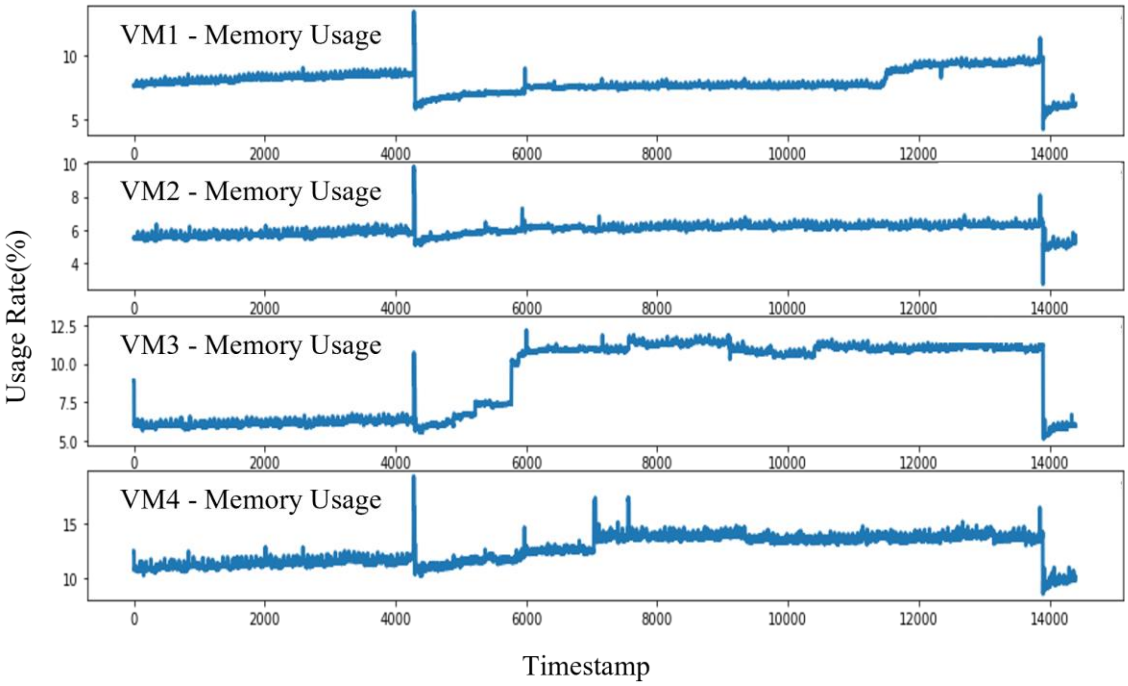
Figure 6. Flow chart of Meta-RL anomaly detection system

3.1 System Abnormal State Analysis

The object of this research is the cluster used by company-A for the upload, search, and calculation of semiconductor packaging and testing production line data. The cluster uses the Zabbix Server tool to monitor various values in the cluster. Zabbix Server is a server monitoring tool that can monitor the utilization rate of various services and resources in the data center and write the monitoring data into CSV files for storage in real-time. First, observe the resource usage data of company-A X-app service from 2021/2/8 to 2021/3/10. X-app is an essential service on the semiconductor production line. This application combines the computing resources of multiple hosts. We found that the memory of the application service will switch once when the utilization rate reaches the peak, and the CPU usage will also reach the peak immediately after the switch. After the switch, the memory will maintain an upward trend for about 20 days, as shown in Figure 7 (a), (b).



(a) CPU usage plot



(b) Memory usage plot

Figure 7. Resource usage plot

The length of the disk queue is also an important indicator. When the disk queue length is greater than 2, it means that the hard disk has too many tasks waiting to be processed. When the CPU usage of the X-app service is high and low, there is a long disk queue length, as shown in Figure 8. Therefore, it can be seen that it is challenging to use thresholds to judge abnormalities in complex trends.

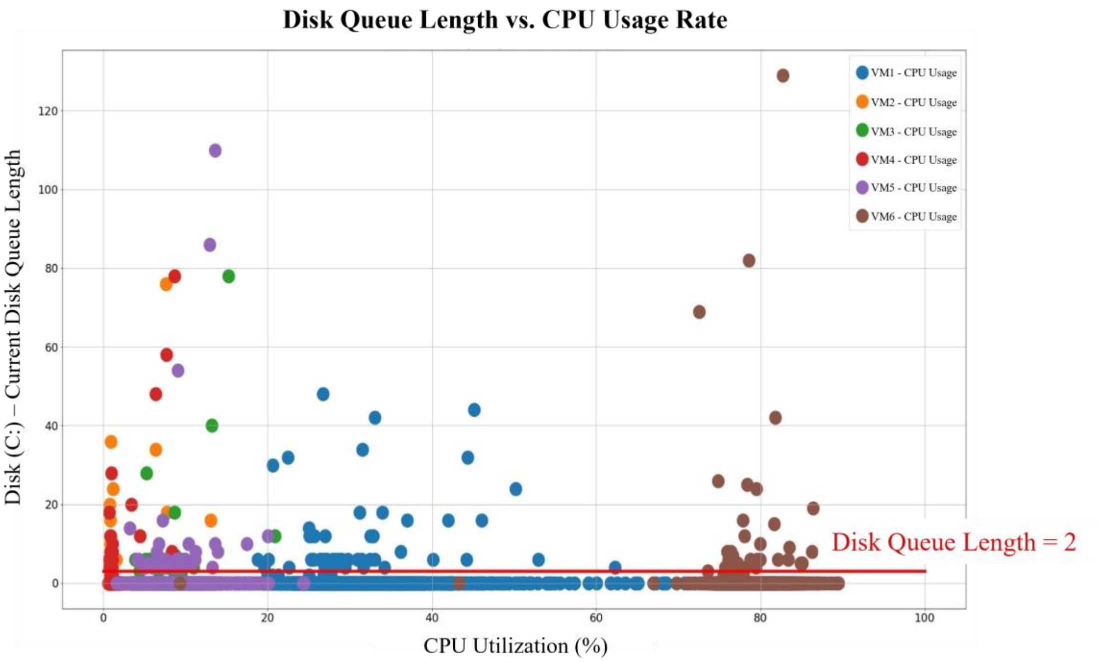


Figure 8. Disk queue and usage plot

3.2 Abnormal data labeling

The data set is to receive data every 3 minutes. It includes information such as CPU, Memory, Disk Queue, and virtual machines of different numbers (codenamed VM1, VM2, VM3, VM4...etc.). We mainly observe the system's CPU and Memory usage data as the main features, as shown in Figure 9.

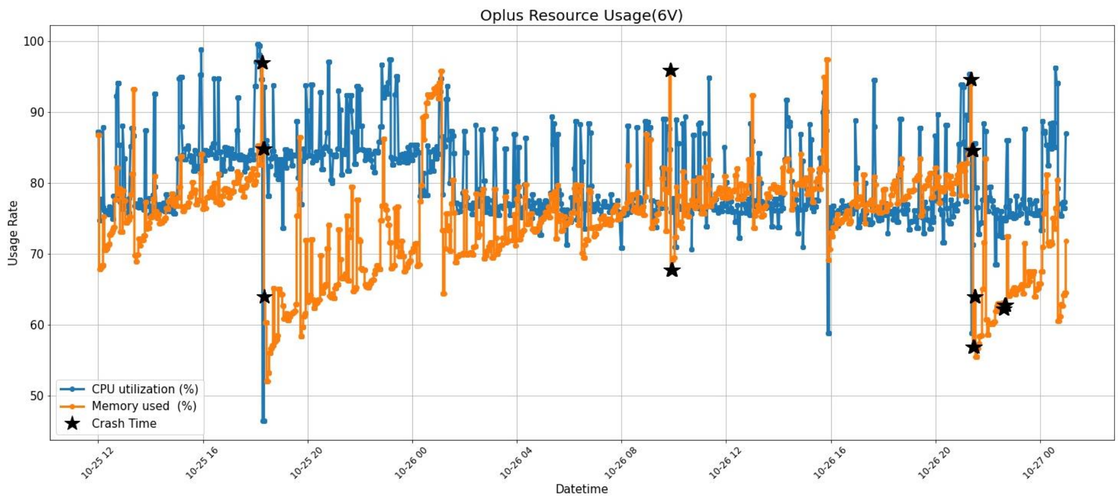


Figure 9. X-app resource usage and exception time stamp

A window marks the real abnormal time with a fixed width, and we use the local normalization method[6] for the data in the sliding window to show sudden data changes, as shown in Figure 10.

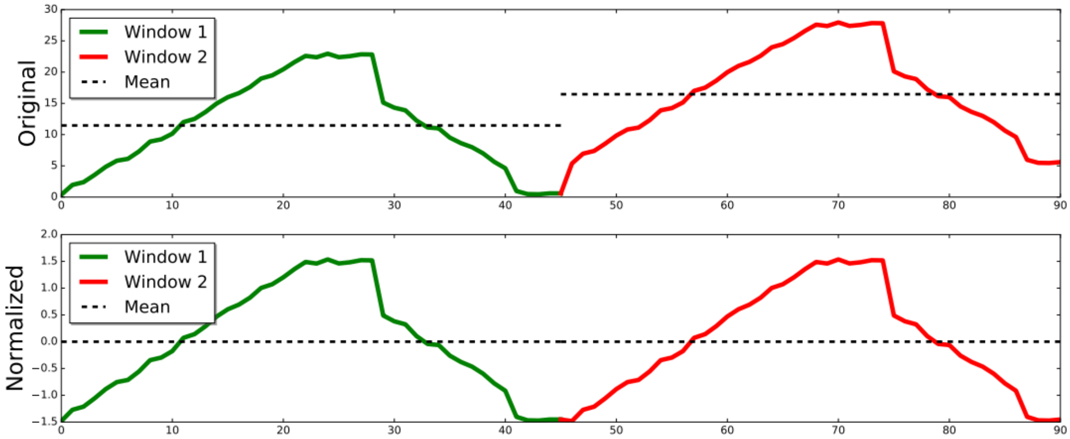


Figure 10. Influence plot of mean drift [6]

We want to warn the detection model when it encounters a precursor to an anomaly on actual binary digit tokens. Therefore, the data set is labeled as 1 in the abnormal interval, and we mark the rest of the normal data as 0 so that as long as there is an abnormal point in the window, it can be determined that the window may be abnormal, as shown in Figure 11.

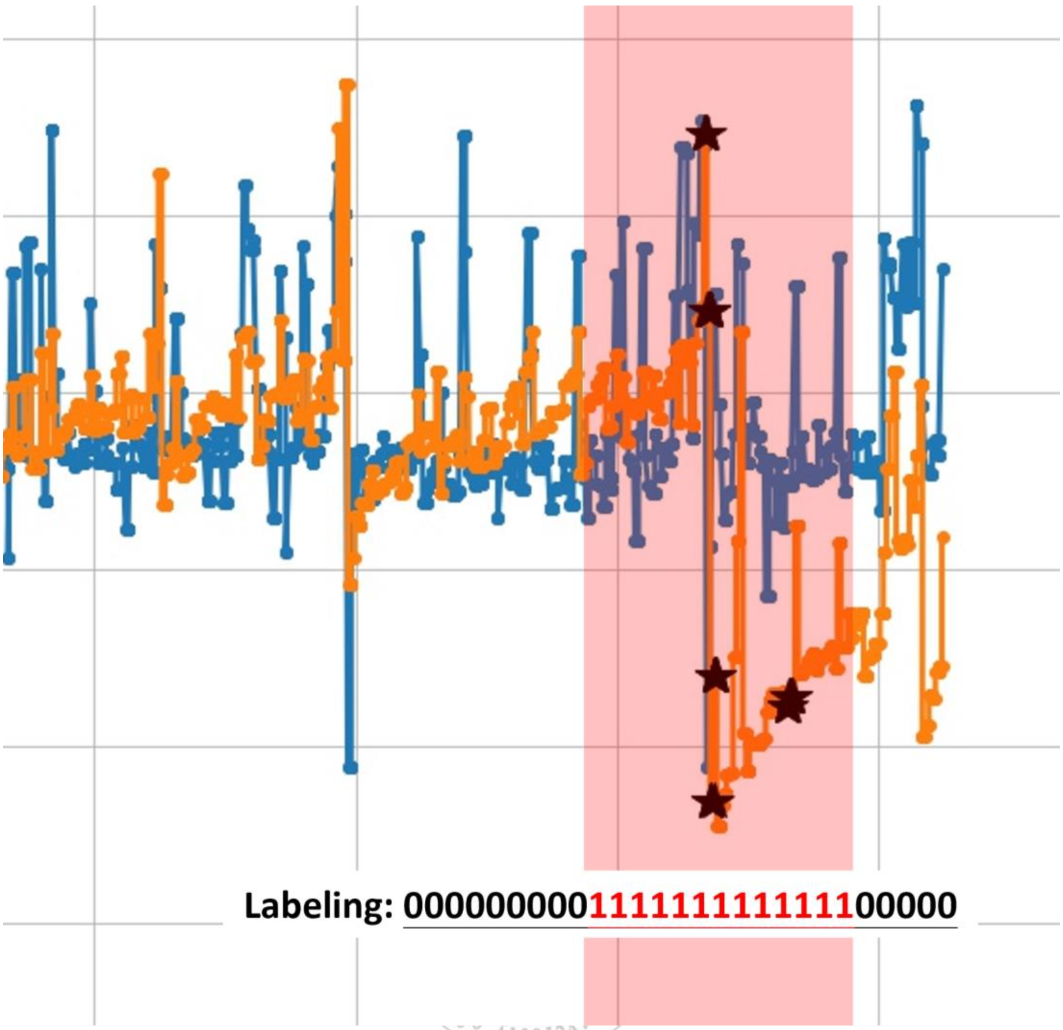


Figure 11. Labeling method of exception window

3.3 Meta feature extraction

This study conducts further feature extraction for time series. First, the original time series contains a complex feature space, which leads to the poor training effect of reinforcement learning. Therefore, we use additional tools to extract time series transferable meta-features. We use TSFEL (Time Series Feature Extraction Library) [7] for analysis. Read the CPU and Memory usage data of virtual machines with different numbers (codenamed VM1, VM2, VM3, VM4...etc.) through Python Pandas. Next, use the TSFEL package to extract statistical domain features, such as maximum and minimum values, gradients, etc., and observe various values and line graphs, as shown in Figure 12.

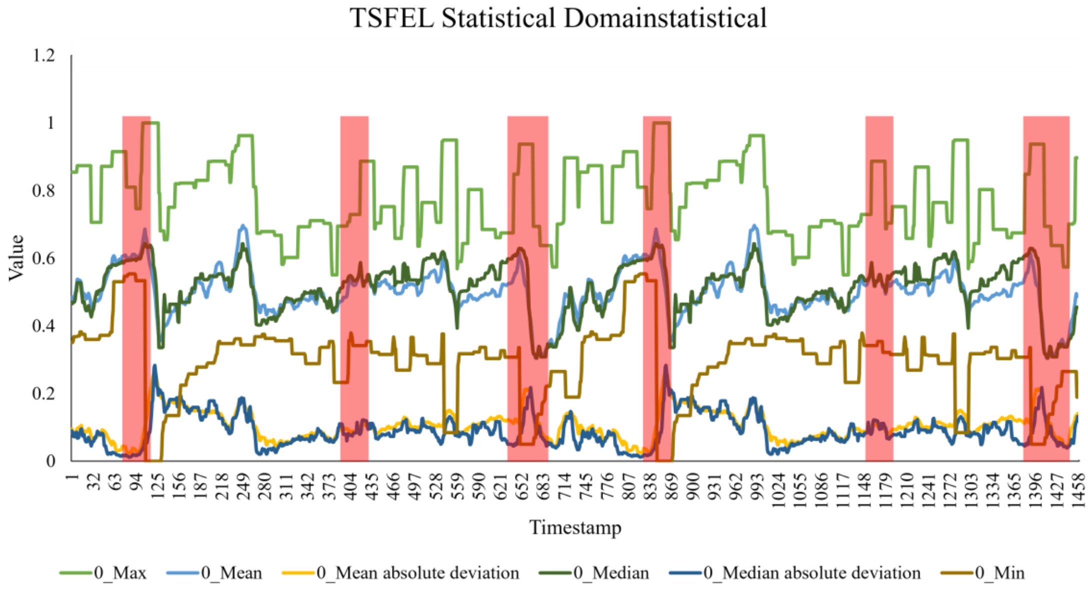


Figure 12. TSFEL statistical domain characteristics

Since the meta-learning process spans data sets with different attributes, the features we need to extract can rely on the meta-data set to prevent feature failure when the data changes. We integrated several unsupervised learning models as pre-processing modules to extract a transferable meta-feature from different data and more accurately evaluate the degree of abnormal data. This step uses the PyOD suite to integrate OCSVM (OneClass SVM), iForest (Isolation Forest), and LODA (Lightweight online detector of anomalies) [10] unsupervised anomaly detection model, using the anomaly score output by the model as the state map of the data set. In addition, we also added the distance between the sample and the abnormal data as a feature. After meta-feature extraction, save it as a CSV file for meta-learning training.

3.4 Build a deep learning training environment

This article uses Anaconda[19] to build a virtual environment, and uses CUDA[20], cudnn[21], and RTX3080 to accelerate the training process. We use Pytorch as the main deep-learning framework and Visual Studio Code as the primary code editor. The hardware settings and package versions as shown in Table 1.

Table 1. Open-source packages version

Package	Version
Python	3.7.10
torch	1.8.1
CUDA	11.1
cudnn	8.1.0

3.5 Create meta-reinforcement learning tasks

This study uses the RLAD[1] method to establish a reinforcement learning environment. First, we use a fixed sliding window to capture data as the environment state (State) in the time series and limit the action space to two discrete states: 0 (normal) and 1 (abnormal). Enter the environment state as the input data into the Policy Network $Eval_N$ to output the decision of the agent, and there is a probability ε to determine whether the agent follows the suggestion made by the policy network. The output decision will calculate the reward with the label of the data set. Here we set the reward function as $\{TP, TN, FN, FP\}$. Finally, there will be a memory module (Replay Memory) to store the transition state$< s, a, r, s' >$, as shown in Figure 13.

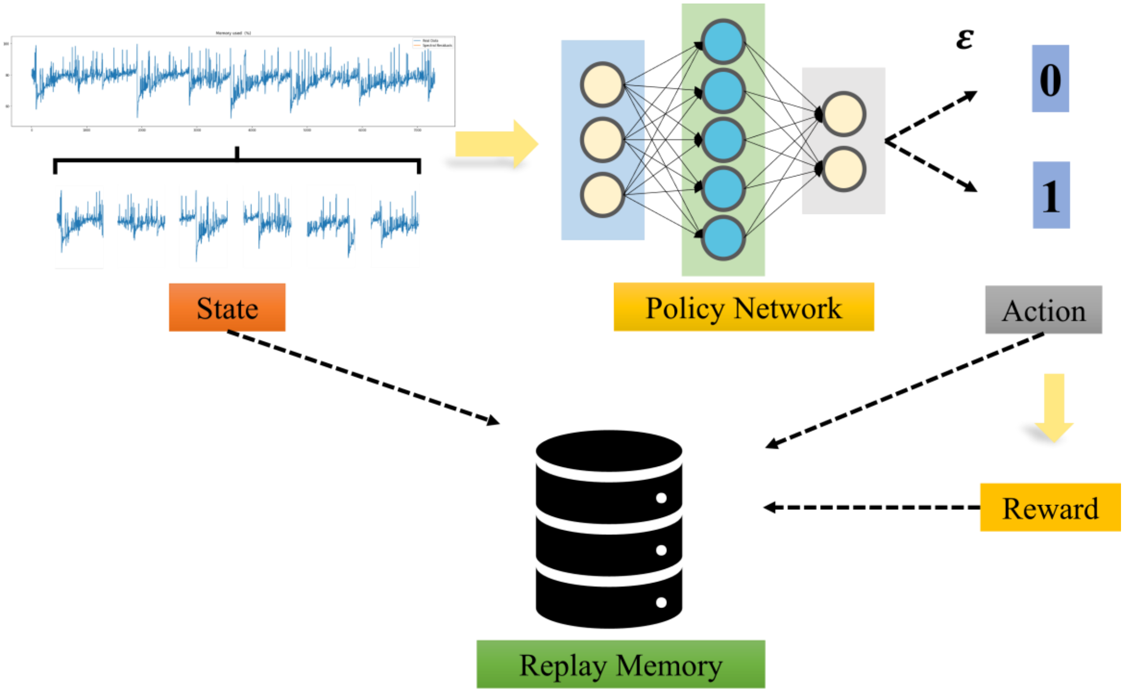


Figure 13. Reinforcement Learning Flowchart

For the Meta reinforcement learning, we regard Environment as a small classification task and slightly modify the reinforcement learning environment as a subtask of meta-learning. The original static reinforcement learning environment was changed to only one interaction, and multiple environments were established for training through multiple execution threads. Action is to judge whether the agent in reinforcement learning is abnormal. The reward is set so that when the meta-strategy selects correct and abnormal data, we give him a reward of 1, and if there is a misjudgment, a small negative reward of -0.1 will be given. However, when the system predicts correct normal data, we give 0 rewards, and we encourage the system to be able to find correct abnormal instances a little more. We use DNN to establish a Policy Network. The Policy Network will receive the environment state, output a probability distribution, and then sample an Action from the distribution. The State, Action, and Reward data will be stored in Replay Memory after each Task operation.

3.6 Create MAML Algorithms

This study uses the learn2learn[9] suite and python language development. First, our inner loop creates multiple execution threads for simultaneous execution in multiple reinforcement learning environments and creates a Task and policy network (θ) for each environment. Next, the State, Action, Reward, and Loss generated in each environment and the updated policy network parameters (θ') are stored in Iteration Replay. In the Outer Loop section, we choose TRPO (Trust region policy optimization) [8] to find the best strategy. The Trajectory of the TRPO algorithm reuses the strategy to increase the utilization rate of samples and also ensures that the reinforcement learning will not affect the model's learning effect due to the strategy change during the training process. The algorithm also delineates a trusted strategy learning area to Ensure the stability and effectiveness of policy learning, as shown in Figure 14.

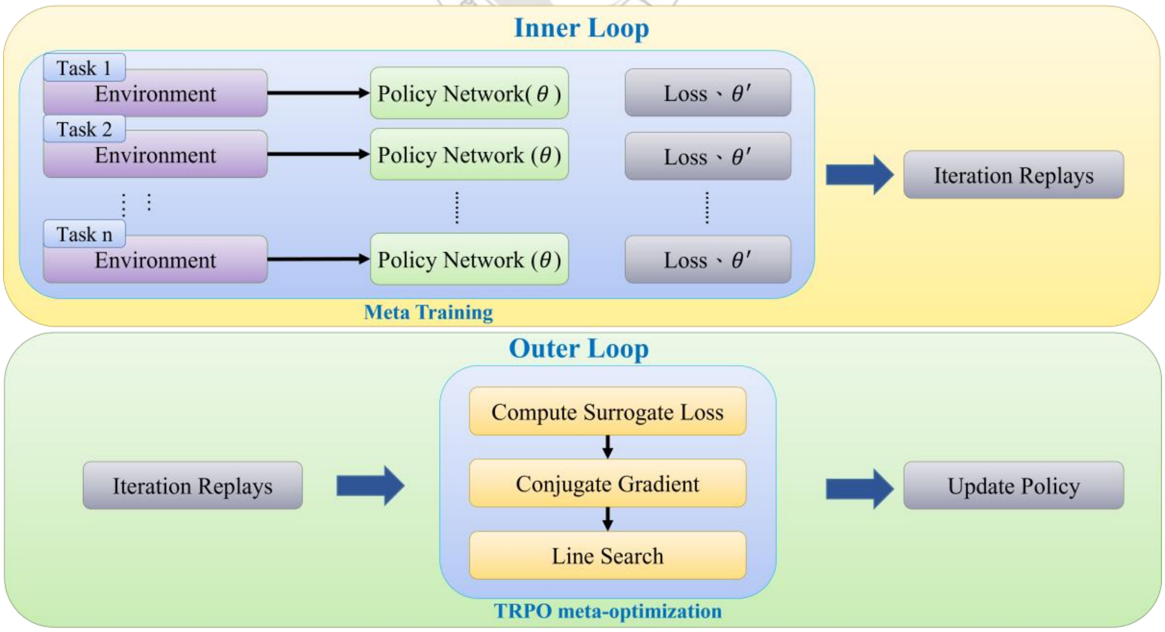


Figure 14. MAML-RL training flow chart

3.7 Training the Meta Reinforcement Model

First, use the OpenAI gym suite to register the resource utilization environment (Step3.5) as this experiment's Environment and set the Inner Loop's hyperparameters as shown in Table 2.

Table 2. MAML hyperparameter settings

Hyperparameter	Value
Hidden layer	[100,100]
Adapt learning rate	0.5
Number of iterations	100
Meta batch size	10
Number of workers	10
cuda	1

Each iteration sets the record file to record the reward and the accuracy of the current final model and sets Tensorboard to observe the trend of the training process, as shown in Figure 15 and Figure 16.

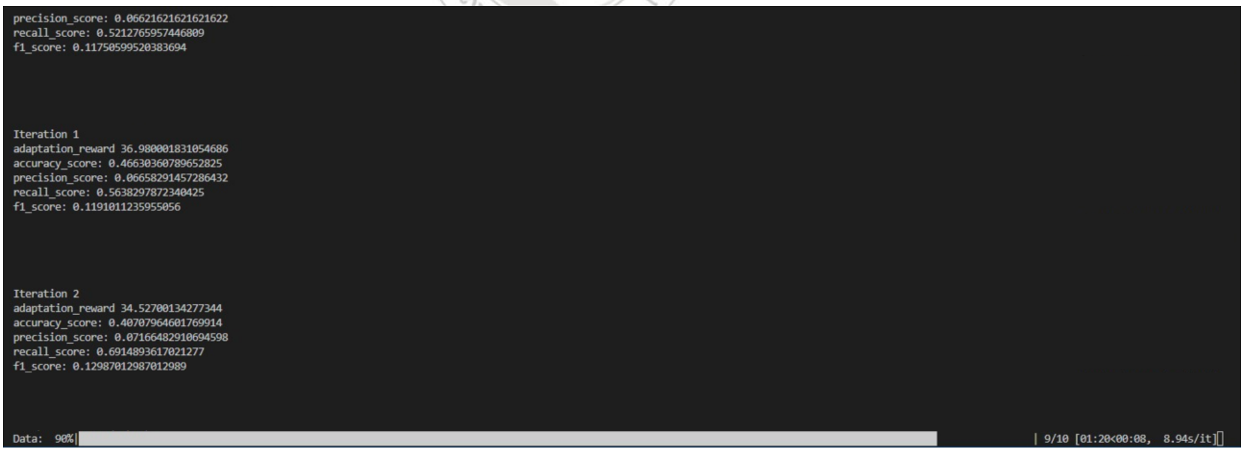


Figure 15. Meta Reinforcement Learning Training

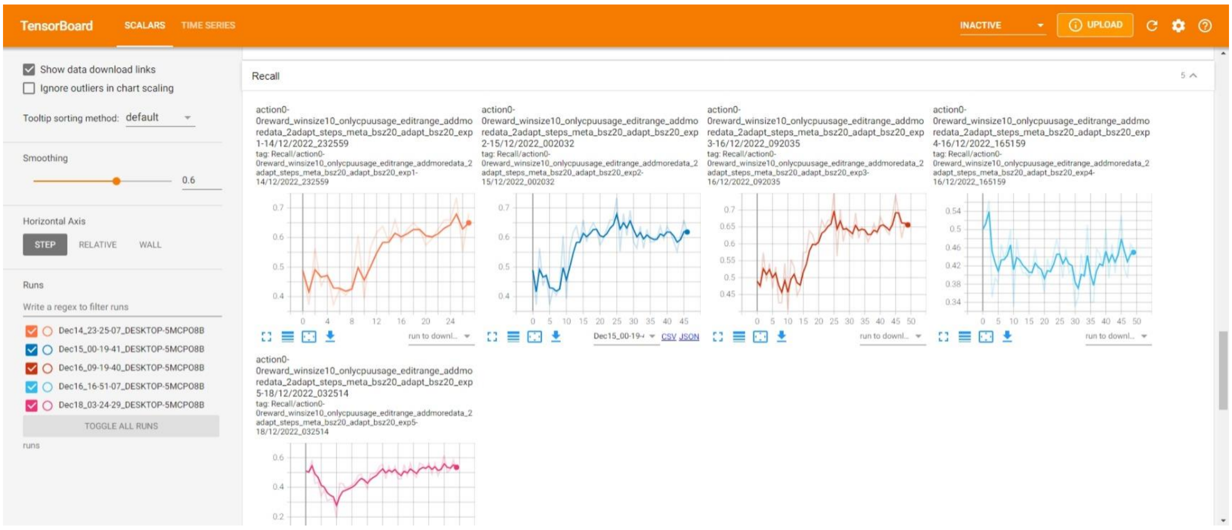


Figure 16. Tensorboard observes the training process

3.8 Policy Network Evaluation

During the training process, we will observe each iteration's accumulated reward value and modify the reward and punishment mechanism of the environment setting through the observed trend, as shown in Figure 17.

Figure 17. Reward trend of reinforcement learning training process

We use Precision, Recall, and F1-score to evaluate whether the model can be deployed. as shown in formulas (1), (2), and (3).

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

The initialization model will first receive the streaming data of the hardware resource utilization rate of the target device offline and perform a short-step gradient update after pre-processing. Finally, evaluate whether the test loss is converged and deployed to the target device.

3.9 Deploy the model to perform online prediction

After evaluation, the models will deploy to the computing cluster of company-A. The company currently uses Hadoop and Spark to build a cluster system and uses Zabbix Server to monitor the resource usage of each virtual machine. The system sets a fixed time to import the data into HDFS (Hadoop Distributed File System). This research uses Spark streaming to convert data into DStream (discretized stream) form and inputs it into the online prediction model for fault prediction, and the detection results will be stored in HDFS immediately. Then the alarm system sends out an alarm to notify the management personnel to deal with it, as shown in Figure 18.

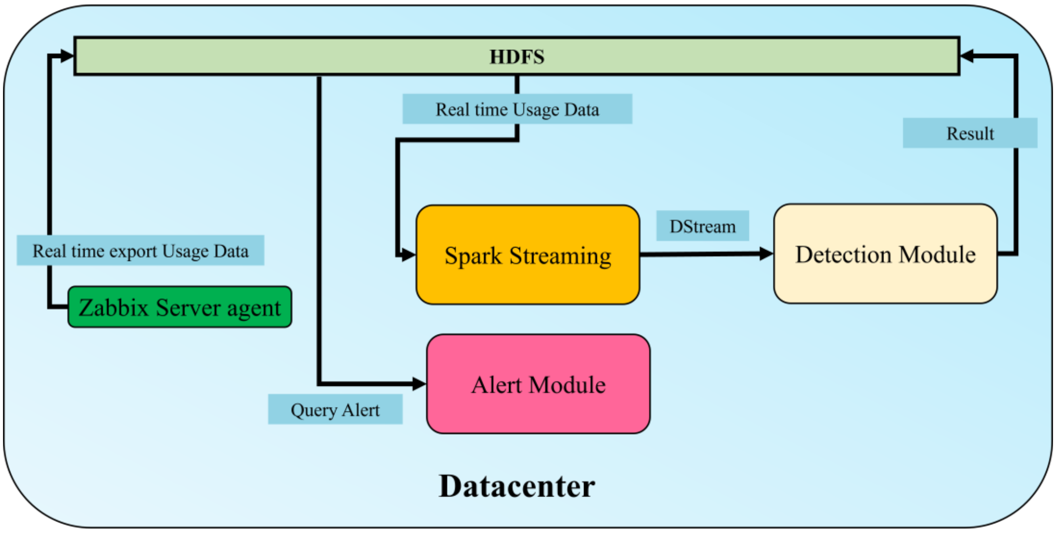


Figure 18. Online detection architecture in the data center

Chapter 4. Experimental Results and Discussion

4.1 Experimental environment

This study uses a deep learning workstation for training, equipped with Intel® Xeon® Silver 4208 and NVIDIA Geforce RTX3080, and has 32GB of DDR4 memory. We use Anaconda to establish a virtual environment for development and the versions of open-source software, as shown in Table 3.

Table 3. Open-source package

Package	Version
conda	22.9.0
Python	3.7.10
Pytorch	1.8.1
CUDA	11.6
cudatoolkit	11.1
cudnn	8.1.0
tensorboard	2.4.1
matplotlib	3.4.2
torchsummary	1.5.1
pyod	1.0.7

The data set part uses X-app, an application service used by company-A to store the semiconductor production line's data. It will be used whenever new data is generated on the machine, or the staff needs to inquire about it. There are frequent exceptions that cause delays when users ask for information. We retrieved the usage rate data from 2021/5 ~ 2021/8 as training data and the 2021/9/1 ~ 2021/9/18 as testing data, as shown in Figure 19.

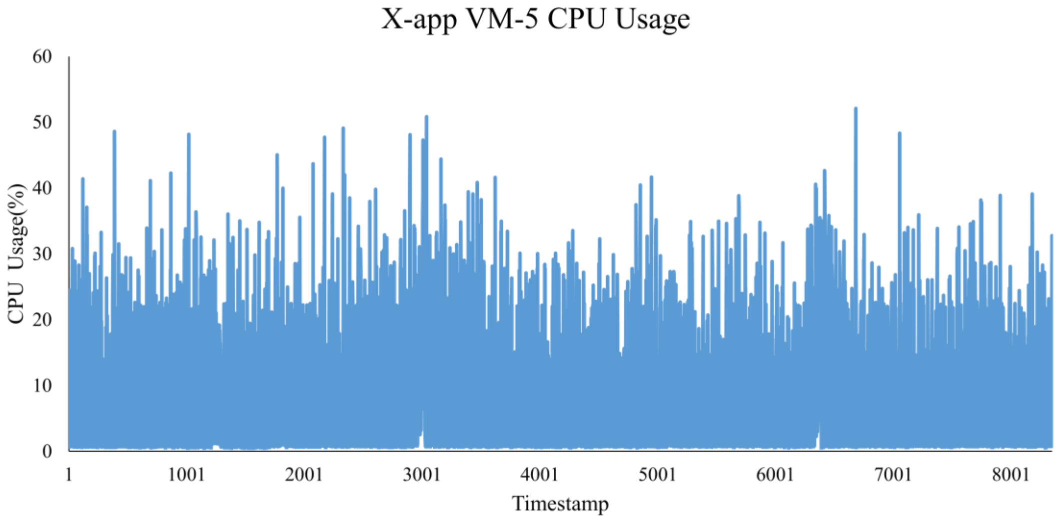


Figure 19. CPU resource usage line chart

In the data pre-processing part, we set the sliding window size to 25, normalize the data to a value in the [0,1] interval, extract the meta-features before training and save it as a CSV file for use. During model training, the data set is split into a training set and a test set at a ratio of 8:2, and 10% of the training set will be extracted as a verification set.

4.2 Experimental design

We conduct two experiments in this section. Experiment 1 will use different adaptation step sizes for training and observe the total reward in the process. Experiment 2 will calculate the time cost comparison of model adaptation, observe the time cost of different retraining models, and evaluate the feasibility of online adaptation. Experiment 3 will compare the anomaly detection performance of our method and the current semi-supervised model in time series.

4.3 Experimental results

4.3.1 Experiment 1

Experiment 1 tests the influence of different adaptation step sizes on reward. We set the learning rate of the Inner Loop to 0.1 and the learning rate of the outer loop to 0.8. The network structure is two layers of neurons with a length of 100, and we test the reward comparison under the adaptation steps 1, 3, and 5, as shown in Figure 20.

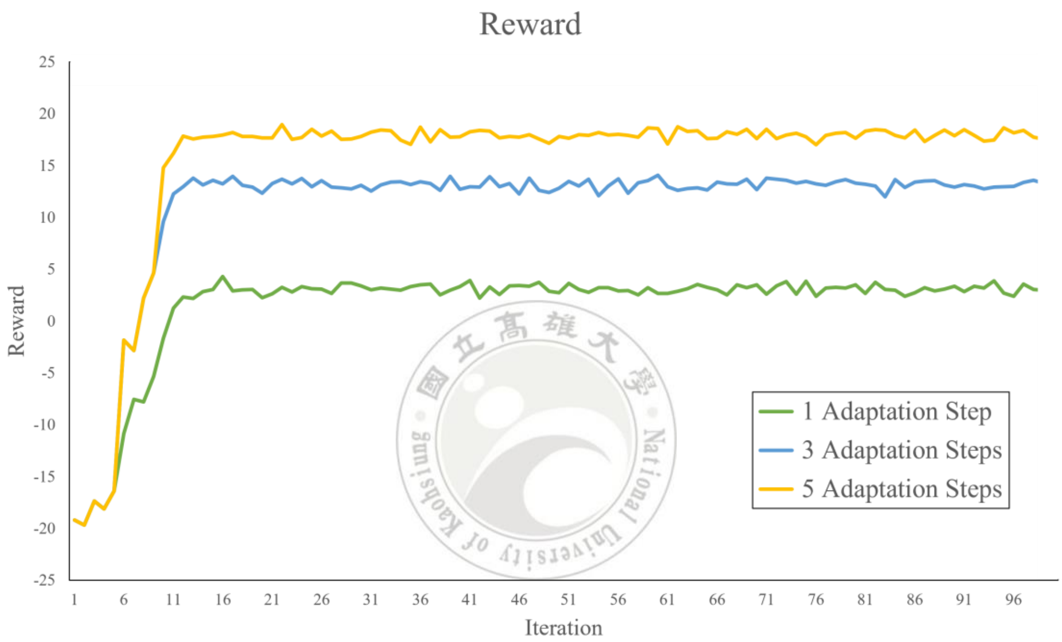


Figure 20. Reward curves under different adaptation steps

4.3.2 Experiment 2

Experiment 2 calculated the time of model training, which was divided into the number of seconds spent training the model for the first time and the time spent re-adapting to new data for the second time. For the first training, a total of 31,920 records of company-A X-app hardware usage data from 2021/5 to 2021/8 were used as prior knowledge for the initialization model. The data from 2021/9/1~2021/9/18 was used as the test data during the second training. The calculation time was based on the time when the F1-score of the model training reached 0.70. The experimental results as shown in Table 4.

Table 4. Adaptive time

Model	First(Ks)	Adaption(Ks)
Autoencoder	0.645	0.0012
Variational Autoencoder	0.815	0.0031
Meta Reinforcement Learning	5.511	0.0003

4.3.2 Experiment 3

Experiment 3 evaluates the F1-score, Recall, and Precision indices of the time series anomaly detection of reinforcement learning in cloud services. We choose Autoencoder, Variational Autoencoder (VAE), and Temporal Convolutional Autoencoder (TCN-AE) [27], which are commonly used semi-supervised learning algorithms for time series anomaly detection, to compare with the reinforcement learning algorithm in this paper, as shown in Table 5. In the abnormal alarm threshold, we use the number of abnormal windows in a fixed interval to judge. If an abnormality exceeding the threshold number occurs in this interval, it is judged as abnormal to prevent excessively sensitive false alarms from occurring.

Table 5. Compare Model Performance

Model	Precision	Recall	F1-score
Autoencoder	0.523	0.6	0.558
Variational Autoencoder	0.401	0.5	0.445
Temporal Convolutional Autoencoder	0.511	0.632	0.565
Meta Reinforcement Learning	0.791	0.772	0.781

4.4 Discussion

In Experiment 1, we know that the adaptation step size can be increased for training when training the model. Still, when the adaptation step size is larger, the reward will not increase in multiples but will reduce the increase. From experiments, it is found that the model will gradually converge at about the 40th iteration number during training, and the number of iterations can be reduced to speed up the training process. In the future, from the different model adaptation times in Experiment 2, although the Meta Reinforcement Learning method in this study has the highest time cost when training the initial model. when a new task is generated or the model needs to be updated, the time for adaptation is at a minimum, it can be shown that our method can quickly adapt to changing datasets. Finally, from Experiment 3, we can see that our model outperforms Temporal Convolutional Autoencoder and improves the performance by about 1.4 times. We do not use an overly complex structure design in the network structure.

Chapter 5. Conclusion

The experimental results show that this study reduces the time and cost of adapting the detection model. Reduce the model failure caused by user behavior changes when detecting cloud application service exceptions. Compared with the semi-supervised learning model, which only learns the data of fixed distribution, the proposed method can quickly adapt to the changing cloud environment, and its accuracy is also improved by 1.4 times. It can be seen that this method has a good performance in the accuracy of identifying anomalies and online adaptability. In this study, we improved the accuracy and adaptability of the model by adding MAML training process, and we also got good results in a small number of positive samples. The method proposed in this study has great potential in maintaining cloud application services, but the model still has room for improvement in terms of current performance. In the future, we can try more advanced methods in data feature extraction and optimizer design to further improve the model's performance. The algorithm can also be more efficiently combined with the actual application cluster. It can speed up the training process by mobilizing the idle resources of different nodes and giving full play to the algorithm's ability.

References

[1] A. Bousdekis, K. Lepenioti, D. Apostolou, and G. Mentzas, “A Review of Data-Driven Decision-Making Methods for Industry 4.0 Maintenance Applications,” *Electronics*, vol. 10, no. 7, 2021.

[2] Wu, Tong, and Jorge Ortiz, “Rlad: Time series anomaly detection through reinforcement learning and active learning,” *arXiv preprint arXiv:2104.00543*, 2021.

[3] D. Zha, K.-H. Lai, M. Wan, and X. Hu, “Meta-AAD: Active Anomaly Detection with Deep Reinforcement Learning,” *CoRR*, vol. abs/2009.07415, 2020.

[4] Vanschoren, Joaquin, “Meta-learning,” *Automated machine learning*. Springer, Cham, pp. 35-61, 2019.

[5] C. Finn, P. Abbeel, and S. Levine, “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks,” *CoRR*, vol. abs/1703.03400, 2017.

[6] S. Saurav *et al.*, “Online Anomaly Detection with Concept Drift Adaptation Using Recurrent Neural Networks,” in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, Goa, India, pp. 78–87, 2018.

[7] M. Barandas *et al.*, “TSFEL: Time Series Feature Extraction Library,” *SoftwareX*, vol. 11, pp. 100456, 2020.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, 2015.

[9] S. M. R. Arnold, P. Mahajan, D. Datta, I. Bunner, and K. S. Zarkias, “learn2learn: A Library for Meta-Learning Research,” *arXiv [cs.LG]*, Aug. 2020.

[10] Y. Zhao, Z. Nasrullah, and Z. Li, “PyOD: A Python Toolbox for Scalable Outlier Detection,” *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1-7, 2019.

[11] Aygun, R. Can, and A. Gokhan Yavuz. “Network Anomaly Detection with Stochastically Improved Autoencoder Based Models,” *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017.

[12] C. Zhou and R. C. Paffenroth, “Anomaly Detection with Robust Deep Autoencoders,” *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[13] C.-W. Tien, T.-Y. Huang, P.-C. Chen, and J.-H. Wang, “Using Autoencoders for Anomaly Detection and Transfer Learning in IoT,” *Computers*, vol. 10, no. 7, ID:. 88, Jul. 2021.

[14] H. Xu, Y. Feng, J. Chen, Z. Wang, H. Qiao, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, and D. Pei, “Unsupervised anomaly detection VIA Variational Auto-Encoder for seasonal KPIs in web applications,” *Proceedings of the 2018*

World Wide Web Conference on World Wide Web - WWW '18, 2018.

[15] J. Alonso, J. Torres and R. Gavaldà, "Predicting Web Server Crashes: A Case Study in Comparing Prediction Algorithms," *2009 Fifth International Conference on Autonomic and Autonomous Systems*, pp. 264-269, 2009.

[16] M. Farshchi, J. Schneider, I. Weber, and J. Grundy, "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis," *Journal of Systems and Software*, vol. 137, pp. 531-549, 2018.

[17] S. Gupta and D. A. Dinesh, "Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks," *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2017.

[18] Z. Zou and J. Ai, "Online Prediction of Server Crash Based on Running Data," *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 7-14, 2020.

[19] "Anaconda Software Distribution", *Anaconda Documentation*. Anaconda Inc., 2020.

[20] Nvidia, P. Vingelmann, and F. H. P. Fitzek, "CUDA, release: 10.2.89," 2020.

[21] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," *CoRR*, vol. abs/1410.0759, 2014.

[22] Z. Xue, X. Dong, S. Ma and W. Dong, "A Survey on Failure Prediction of Large-Scale Server Clusters," *Eighth ACIS International Conference on Software Engineering*, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), 2007, pp. 733–738, doi: 10.1109/SNPD.2007.284.

[23] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis and H. Zhang, "Automated IT system failure prediction: A deep learning approach," *2016 IEEE International Conference on Big Data (Big Data)*, pp. 1291-1300, 2016. doi: 10.1109/BigData.2016.7840733.

[24] S. Zhang, F. Ye, B. Wang and T. G. Habetler, "Few-Shot Bearing Anomaly Detection via Model-Agnostic Meta-Learning," *2020 23rd International Conference on Electrical Machines and Systems (ICEMS)*, pp. 1341-1346, 2020. doi: 10.23919/ICEMS50442.2020.9291099.

[25] Nvidia, P. Vingelmann, and F. H. P. Fitzek, "CUDA, release: 10.2.89," 2020.

[26] H. -S. Wu, "A survey of research on anomaly detection for time series," *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 426-431, 2016. doi: 10.1109/ICCWAMTIP.2016.8079887.

[27] M. Thill, W. Konen, H. Wang, and T. Bäck, "Temporal convolutional autoencoder for unsupervised anomaly detection in time series," *Applied Soft Computing*, vol. 112, pp. 107751, 2021.