Online Self-Evolving Anomaly Detection for Reliable Cloud Computing

Tianyu Bai[*], *Haili Wang*[*], *Jingda Guo*[*], *Sihai Tang*[*], *Xu Ma*[†], *Mahendra Talasila*[*], *Song Fu*[*] and *Qing Yang*[*]
[*]Department of Computer Science and Engineering, University of North Texas
[†] Department of Electrical and Computer Engineering, Northeastern University
[*]{TianyuBai, HailiWang, JingdaGuo, SihaiTang, MahendraTalasila}@my.unt.edu,
[†]ma.xu1@northeastern.edu, [*]{Song.Fu, Qing.Yang}@unt.edu

***Abstract*—Production cloud computing systems consist of hundreds to thousands of computing and storage nodes. Such a scale, combined with ever-growing system complexity, is causing a key challenge to failure and resource management for dependable cloud computing. Efficient system monitoring and failure detection are crucial for understanding emergent, cloud-wide phenomena and intelligently managing cloud resources for system-level dependability assurance and application-level performance assurance. To detect failures, we need to monitor the cloud execution and collect runtime performance data. These data are usually unlabeled at runtime in real-world systems, and thus a prior failure history is not always available. In this paper, we present a *self-evolving anomaly detection* framework for cloud dependability assurance. Our framework does not require any prior failure history, and it self-evolves by continuously exploring newly verified anomaly records and continuously updating the anomaly detector at runtime without expensive model retraining. A distinct advantage of our framework is that cloud system operators only need to check a small number of detected anomalies (compared with thousands-millions of system/application event records) and their decisions are leveraged to update the detector. Thus, the detector evolves following the upgrade of system hardware, update of software stack, and change of user workloads. Moreover, we design two types of detectors, one for general anomaly detection and the other for type-specific anomaly detection. Leveraging self-evolution and online learning techniques, our detectors can achieve 88.94% of sensitivity and 94.60% of specificity on average, which makes them suitable for real-world deployment.**

***Index Terms*—Cloud Computing, Reliability, Anomaly Detection, Online Learning.**

I. Introduction

Cloud computing is widely used in almost all aspects of our daily life [31], from social media, online shopping, movie streaming, photo storage, document editing to scientific computing, big data processing, and smart cities (e.g., autonomous vehicles, smart transportation and more). Production cloud systems, such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure, are both economically successful and technically popular.

Despite the great efforts on the design of reliable components, the increase of cloud systems' scale and complexity has outpaced the improvement of components' reliability. Failure occurrence as well as its impact on cloud performance and operating costs becomes an increasingly important concern to cloud system operators and cloud service providers [19]. Anomaly detection [24] is an important failure management technology for computer systems. It detects anomalous system/component behaviors and possible failures by analyzing history behaviors and execution states. Anomaly detection in cloud computing systems provides a cost-effective mechanism for resource allocation, virtual machine/container scheduling, and cloud maintenance.

During cloud operations, a large amount of monitoring data is collected to track the cloud's operational status. Software log files, system audit events, and network traffic statistics are examples of such measurements. These data provide valuable information about the cloud's states and health. A failure occurrence scatters its trace in the measurement data and we need to analyze the data to identify a system/component failure. However, cloud measurements usually contain a huge number of attributes and continuous monitoring leads to an overwhelming data volume. It is very difficult, if not impossible, to manually infer the cloud operating status from those measurements. Another challenge of anomaly detection from measurement data originates from the dynamics of cloud computing systems [28]. It is common in those systems that user behaviors and servers' loads are changing. The cloud hardware and software components are also frequently replaced, upgraded, or updated. This requires an anomaly detection mechanism be able to identify new types of anomalies and update its detection model at runtime as the executing environment changes.

The traditional approaches to anomaly detection rely on statistical analysis or learning algorithms to approximate the dependency of failure occurrences on various performance attributes; see [13] for a comprehensive review. The underlying assumption of those methods is that the training datasets are labeled, i.e., for each measurement used to train an anomaly detector, the designer knows if it corresponds to a normal execution state or a failure. However, the labeled data are not always available in real-world cloud computing systems, especially with hardware and/or software upgrades or updates, or workload changes.

Moreover, the existing approaches usually adopt an offline-training-and-online-detection scheme. Specifically, an anomaly detector is trained offline using a vast amount of data. Then, the detector is used for online anomaly detection. Once deployed, the detector is rarely changed unless the detection accuracy is lower than expectation (e.g., below a predefined

accuracy threshold). When the detector becomes inaccurate, an offline retraining is performed and the anomaly detector is replaced by a newly trained model. As a result, the performance of anomaly detection fluctuates, that is the detection accuracy drops as time goes by until a replacement by the retrained detector occurs, which causes an abrupt change of the detection accuracy. After that, the cycle repeats. As model retraining is time-consuming, the obsolete detector remains in service for a longer period of time, which causes more system/component failures undetected and/or false alarms. This fluctuation of the detection performance affects the efficacy of system monitoring and resource management as well.

In this paper, we address these issues by presenting a *self-evolving anomaly detection* framework. It adopts a novel, adaptive anomaly detection approach. Specifically, it does not require a prior failure history. It continuously monitors the cloud execution and collects runtime performance data. To tackle the high dimensionality of cloud performance metrics, our framework extracts the most relevant metrics for anomaly detection. It describes the cloud performance using the relevant metrics and employs online learning models to detect anomalies. As the detections later get verified (either confirmed or rejected) by cloud operators, our framework adapts its anomaly detector by continuously learning from these verified detections (i.e., online self-evolution). The anomaly detectors are updated to refine future detections without service disruption (due to retraining) or abrupt changes (due to model replacement). In addition, the cloud operators can report observed but undetected failure events to the anomaly detector which exploits these events to improve its model and detect new types of anomalies.

The main contributions of this paper are as follows.

- We design a self-evolving anomaly detection framework for cloud computing systems. It does not require any previously labeled failure data and can learn from newly generated records incrementally online. No hardware or software in the existing cloud computing systems needs to be modified to support the proposed framework.
- In pursuit of online learning, we develop an efficient anomaly detector, leveraging a stochastic gradient descent method to build a detection model.
- While existing anomaly detection methods only classify a record as either normal or not, ignoring the difference between anomalies, our framework can detect abnormal behaviors with their different anomaly types.
- By adding a sparsity regularization component to our framework, the anomaly detector can select the most relevant attributes for effective detection.
- Results from extensive experiments show that our self-evolving framework achieves a better performance compared with the existing approaches/models. Specifically, our self-evolving anomaly detector achieves 88.94% of detection sensitivity and 94.60% of detection specificity without expensive retraining, making it practical for real-world cloud computing systems.

The remaining paper is organized as follows. In Section II, we review the related work on cloud computing anomaly detection. Section III presents the pipeline of our self-evolving framework and detail our novel detectors. Section IV presents and discuss the experimental results. Section V concludes the paper with remarks on future research.

II. Related Work

Theories and practices that apply machine learning methods to anomaly detection have been studied for many years. An early practice of learning anomaly detection is to apply Naive Bayes [25] for detecting cloud anomaly [27]. Combined with feature transformation algorithms, the work in [27] achieved a promising performance. Following a similar strategy, a distributed approach [22] used Independent Component Analysis (ICA) [15] instead of Principal Component Analysis (PCA) [16] for feature selection. In addition, a decision tree model was employed to classify anomalies. The general pipeline (composed of two modules: an unsupervised feature transformation module that preprocesses the collected data and a supervised classification module that detects anomalies) has also been used for disk failure analysis, network security, cloud computing detection, etc. Nevertheless, detecting cloud anomalies by simply using a supervised learning algorithm has limitations. The unlabeled data logs and entries make that type of approach impractical for online anomaly detection in real-world cloud computing systems with constant changes (e.g., hardware upgrades, software updates, and workload shifts).

Considering a lack of prior knowledge of failures, Pannu et al. [24] used unsupervised methods (i.e., one class support vector machine [3]) for cloud anomaly detection. Their strategy allows the adaptive system to surpass the previously published works in real-world applications, such as [11], [30]. The success comes from 1) no introduction of prior knowledge in the model learning stage, and 2) a high-capacity of one-class support vector machine (SVM). Analogous to Pannu's work [24] that applied unsupervised learning algorithms, recent studies [9], [14] employed support vector data description [29] to detect ICA/Kernel-PCA transformed data. The results demonstrated that they improved the detection sensitivity by 19.6% over Bayesian predictors and decision trees ensemble [11].

With the advance of machine learning, a trend of using learning algorithms to detect cloud computing anomaly is emerging [20], [23]. The supervised detector learns in a manner of feedback from detections. Hence, adaption is an elastic alternative of unsupervised learning, especially in the field of cloud computing anomaly detection. The anomaly detection framework yields a better cloud computing anomaly detection performance [5], [10], [12]. Aside from being capable of replacing unsupervised detectors, adaptive frameworks are also appropriate for online learning.

In parallel with our work, there are efforts, such as [4], [26], using neural network, especially deep learning, for anomaly detection. In addition to accuracy, online learning would not be an issue as stochastic gradient descent (SGD) optimizer

[17] has been adopted in those methods. Although neural networks achieve good results in computer vision and natural language processing, it may not be suitable for cloud anomaly detection. The key limitations are as follows. 1) Training a neural network requires extensive resources and consumes a considerable amount of time, which is not cost-effective considering the scale and complexity of cloud computing systems. 2) The cloud failure data is highly imbalanced and traditional neural networks cannot handle well. 3) The log data from real-world cloud computing systems usually do not contain a large number of labeled instances for training a complex neural network.

Different from aforementioned works, our proposed framework pursuit a simple yet efficient realization of anomaly detection for real-world cloud computing systems. We customize models for class-agnostic and class-known anomaly detections respectively. The models meet the requirements of classification, online learning, and feature selection. The customized models can be considered as performing both a feature reduction task and an anomaly detection task (e.g. [9], [27]). Unlike PCA and ICA which project the original data to a new subspace and lose the original semantics of features, we evaluate each feature by weights. By leveraging the advantages of the model that can self-evolve incrementally, we can dynamically and automatically select the most important features for future detections. We design a self-evolving framework that feeds verified predictions to the training process to automatically adapt the model. This design allows us to employ our framework and detectors to real-world cloud computing systems without overfitting, performance fluctuation, or service disruption.

III. Self-Evolving Anomaly Detection Framework

To build a self-evolving architecture for cloud anomaly detection, we propose a decoupled framework, leveraging the self-evolving property to adjust our embedded customized detection models over iterations, and improving the performance over time. Different from the existing two-stage detection methods (i.e., using PCA/ICA for dimension reduction, then feeding the processed data to a classifier), we design a series of single-stage detectors for cloud anomaly detection. These detectors identify anomalies, select relevant attributes, and improve models continuously at runtime.

A. Overall Architecture and Key Components

System states and application execution are monitored in a cloud computing system. The collected runtime data is processed by an anomaly detector for classification. The detected anomalies are then examined by cloud operators. The verified events are then used to update/refine the detector at runtime without offline model retraining. The detector selects the most relevant attributes for future detections. Algorithm 1 sketches our self-evolving anomaly detection framework.

When the anomaly detection system is first deployed in a cloud computing environment, we selectively initialize the detector to identify most records as normal. The detector predicts records and sends predicted anomaly records to the cloud operators for verification. These records are identified as either failures or normal states once a single epoch of cloud performance data records is available. Different from other approaches that send all records to the cloud operators, our method significantly reduce the workload of the cloud operators by only checking the predicted anomalies and achieve comparable results (See Section IV). The verified records are used to update the detector. In our experiments, a model achieves satisfactory results after ten epochs with 300 examples in each epoch.

Algorithm 1 Self-Evolving Anomaly Detection

Input: Newly collected cloud performance data $\mathbf{X}$
Output: Updated anomaly detector $\mathfrak{D}$
Initialize $\mathfrak{D}$.
while $\mathbf{X}! = NULL$ **do**
  Predict label $\mathbf{Y} = \mathfrak{D}$
  Find predicted negative labeled data $\mathbf{X}_{neg}$
  Send $\mathbf{X}_{neg}$ to Cloud Operators and get verified label $\mathbf{Y}_{neg}$
  Online updating detector $\mathfrak{D} = update(\mathfrak{D}, \mathbf{X}_{neg}, \mathbf{Y}_{neg})$

The pipeline of our architecture requires an embedded detector. Most current advances in cloud computing anomaly detection have been driven by combinations of dimension reduction methods and powerful basic statistical classifiers, such as [14], [27]. However, such a combination would not be suitable for our self-evolving framework since it is unable to adjust models in an incremental fashion. To address it, we develop new detectors (detailed in the following sections) which can detect anomalies with their abnormal reason. Moreover, by introducing a sparsity regularization component, our detector can also select the most distinguishing attribute from records.

B. Cloud Anomaly Detection

In this section, we present a novel incremental detector for class-agnostic anomaly detection in cloud computing systems.

To ease our discussion, we use the following notations. The monitoring system collects n execution/state records from the cloud, and each record contains d attributes. The set of records is denoted as $\mathbf{X} \in \mathbf{R}^{n \times d}$ in the real valued space. The label y of a certain record is either 1 or -1, representing a normal record or a failure record. Initially, we use a vector $\mathbf{Y} \in \mathbf{R}^n$ to store the labels of $\mathbf{X}$.

For detection, we adopt a classification plane fashion to build our models, which is similar to the Least Squares Estimator (LSE) and Support Vector Machine (SVM). Let w denote a classification plane which is a column vector (that is $w \in \mathbf{R}^d$), the predictions of $\mathbf{X}$ can be expressed by $\mathbf{X}w$. Following statistical analysis methods, we design an incremental binary classifier, called **L1LS**, for cloud anomaly detection. The design of L1LS is described next.

The Least Squares Estimator [8] is a machine learning method which minimizes the residual sum of squared errors

between the true labels and predictions. That is

$$\sum_{i=1}^{n} \left(y_i - x_i * w - w_0\right)^2, \tag{1}$$

where w_0 is the bias. We rewrite the objective function by appending a column vector of 1s and increasing the length of w by 1 as follows.

$$\min \|\mathbf{Y} - \mathbf{X}w\|_2^2. \tag{2}$$

The classification plane w can be considered as the coefficients of attributes. For instance, w_j is the weight for attribute X^j. The greater w_j is, the more X^j contributes to the detection. Hence, to select the most relevant attributes, we should get the elements in w close to 0. Inspired by research on sparsity such as [18], [21], we modify Equation (2) by adding L1-norm regularization, which improves the sparsity of w. That is

$$\min \|\mathbf{Y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_1, \tag{3}$$

where λ is a parameter representing the trade-off between the empirical loss and model sparsity. We name our design using LSE with L1-regularization as L1LS.

We note that L1LS cannot be embedded in our self-evolving framework due to the following two issues. 1) The objective function of L1LS is not convex, which makes it difficult to solve. 2) L1LS is not an incremental method which is required by our framework. To address these issues, we leverage the Stochastic gradient descent (SGD) method to build our model. SGD only needs the first-order derivative of our original model, and it can converge to a local optimal solution. Another advantage of using SGD is that it meets the requirement of incremental learning, as SGD learns one example from anther (or one batch of examples from another batch).

The derivative on both sides of Equation (3) is as follows.

$$\frac{\partial \mathbb{F}}{\partial w} = -2\mathbf{X}'\left(\mathbf{Y} - \mathbf{X}w\right) + \lambda sign\left(w\right), \tag{4}$$

where $sign\left(\cdot\right)$ denotes an element-wise function that returns 1 if the element's value is greater than zero or -1 otherwise. Specially, if an element is zero, we add a small value $1e^{-7}$ to make the function differentiable. Based on our experiments, we set the learning rate of SGD to $1e^{-4}$ for training [1]. After several iterations, the objective function converges to a local minimum and the sub-optimal w can be determined. Algorithm 2 describes how to solve L1LS.

Once we have the output w, we are able to detect a cloud performance record x by comparing the distance of xw to 1 and to -1. If xw is closer to 1, we predict it as a normal record, otherwise it is considered as an anomaly record.

[1]A dynamic selection approach is used to modulate the learning rate. For a quick training process, the initial learning rate is set, e.g., 0.1. Each time when the value of the objective function does not decrease, we divide the learning rate by 10. The minimum learning rate is $1e^{-7}$.

C. Self-Evolving Anomaly Detection

In the previous section, we present the L1LS model for anomaly detection. L1LS is effective to detect anomalies and is capable of selecting the most relevant attributes for detection. However, it is a binary classifier that can only detect whether anomalies exist, but not their types.

To generalize from a binary classifier to a multi-class classifier, we consider two possible ways: 1) "One vs Rest" and 2) "One vs One". The "One vs Rest" strategy trains a single classifier for each class. For C-class detections, we need to train $C-1$ classifiers. In contrast, the "One vs One" strategy trains $\frac{C(C-1)}{2}$ binary classifiers for a C-way multi-class problem. There are drawbacks in either of the strategies, e.g., the training stage is expensive in both time and space. In addition, these two strategies for multi-class classification problems could lose the competition between classes. They ignore the information that contributes to differentiating classes.

To address the preceding issues, we develop a multi-class classifier based on L1LS by leveraging the one-hot encoder. Let x be a performance record that belongs to the c-th class, based on one-hot encoder, label y is denoted by a row of zero valued vector $y \in \mathbf{R}^c$ where the c-th element's value is one [2]. For example, if there are five classes and a record that belongs to the 3-th class, the label can be presented by $[0, 0, 1, 0, 0]$. For a dataset $\mathbf{X} \in \mathbf{R}^{n*d}$, the corresponding labels $\mathbf{Y} \in \mathbf{R}^{n\times c}$ are presented by a n by c matrix.

1) MCL1LS: Different from L1LS in which the classification plane is a one-dimensional vector, the classification plane $\mathbf{W} \in \mathbf{R}^{d\times c}$ of the new multi-class L1-norm Least Squares model (MCL1LS in short) is a c-dimensional matrix. A general MCL1LS model can be expressed as follows.

$$\min \|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_2^2 + \lambda \|\mathbf{W}\|_1 \tag{5}$$

and the related derivative are as follows.

$$\frac{\partial \mathbb{F}}{\partial \mathbf{W}} = -2\mathbf{X}'\left(\mathbf{Y} - \mathbf{X}\mathbf{W}\right) + \lambda sign\left(\mathbf{W}\right) \tag{6}$$

Although Equation (5) is similar to Equation (3), it is noted that $\mathbf{W}$ in MCL1LS is a matrix while w in L1LS is a column vector. This change enables MCL1LS to detect multi-class records.

Once we obtain the desired $\mathbf{W}$, the prediction of a record x can be performed by computing $x\mathbf{W}$ and the index of the max value in vector $x\mathbf{W}$ indicates the predicted class of x.

2) MCL21LS: MCL1LS improves the detection of anomalies from known classes. However, it is not able to select the most relevant attributes. For attribute selection, it requires that most of the rows in $\mathbf{W}$ should be close to zeros. A L1-norm based regularization term is not enough as it cannot deal with row sparsity. In order to handle the row sparsity in $\mathbf{W}$, we design an enhanced model by leveraging the L21-norm regularization term.

[2]In experiments, we find that using +1 and -1 achieves better results than using 0 and +1. This can be explained from two aspects. 1) The model is a distance based model instead of a possibility based model, and the output is not in the range of $[0, 1]$. 2) The distance between -1 and +1 is greater than that between 0 and +1.

Different from the sparsity requirement of the L1-norm regularization term, the L21-norm regularization term requires row sparsity. For a matrix $\mathbf{W} \in \mathbf{R}^{d \times c}$, the L21 norm is defined as

$$\|\mathbf{W}\|_{2,1} = \sum_{i=1}^{d} \sqrt{\sum_{j=1}^{c} w_{ij}^2} = \sum_{i=1}^{n} \left\|w^i\right\|_2, \tag{7}$$

where w_{ij} is the element in the i-th row and j-th column in $\mathbf{W}$, and w^i denotes the vector in the i-th row in $\mathbf{W}$.

By adapting the L21-norm regularization term, the target problem becomes

$$\min \|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_2^2 + \lambda \|\mathbf{W}\|_{2,1}. \tag{8}$$

We call this model a Multi-Class L21-norm Least Squares model (MCL21LS in short). MCL21LS is effective in selecting the most relevant attributes. Algorithm 2 sketches how to solve MCL21LS and it is compatible with online learning.

Algorithm 2 Solving MCL21LS Using SGD.

Input: Newly collected cloud performance data $\mathbf{X}$
Output: w
Initialize weight w, learning rate lr, and parameter λ
Compute the objective function $\mathbb{F}$
Set $d = \infty$
while $d > 1e^{-6}$ **do**
  Compute the derivative der using Equation (4)
  Update the value of the objective function $\mathbb{F}_{new} = \mathbb{F}_{previous} - lr * der$
  Update $d = \mathbb{F}_{previous} - \mathbb{F}_{new}$

From Equation (7), we obtain the derivative of the L21-norm $\mathbf{W}$ as follows.

$$\frac{\partial \|\mathbf{W}\|_{2,1}}{\partial \mathbf{W}} = \left[\frac{\partial \left(\sum_{i=1}^{d} \|w_i\|_2\right)}{\partial w_j}\right]_{d\times 1} \tag{9}$$

$$= \left[\frac{\partial \left(\sum_{i=1}^{d} \left(w_i w_i^T\right)^{\frac{1}{2}}\right)}{\partial w_j}\right]_{d\times 1} \tag{10}$$

$$= \left[\frac{w_j}{\|w_j\|_2}\right]_{d\times 1} \tag{11}$$

$$= \begin{bmatrix} \frac{1}{\|w_1\|_2} & & & \\ & \frac{1}{\|w_2\|_2} & & \\ & & \ddots & \\ & & & \frac{1}{\|w_d\|_2} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \tag{12}$$

$$= \begin{bmatrix} \frac{1}{\|w_1\|_2} & & & \\ & \frac{1}{\|w_2\|_2} & & \\ & & \ddots & \\ & & & \frac{1}{\|w_d\|_2} \end{bmatrix} \mathbf{W} \tag{13}$$

$$= \Sigma \mathbf{W} \tag{14}$$

where Σ is a diagonal matrix. Thus, the derivative of Equation (8) w.r.t. W can be expressed as follows.

$$\frac{\partial \mathbb{F}}{\partial \mathbf{W}} = -2\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\mathbf{W}) + \lambda \Sigma \mathbf{W}. \tag{15}$$

Applying Algorithm 2, we can solve Equation(8) efficiently.

IV. Performance Evaluation

We have implemented a prototype of the self-evolving anomaly detection framework and models and have conducted experiments using performance data collected from a real-world cloud computing system. In this section, we present the experimental results and evaluate the performance of our anomaly detectors.

A. Experiment Settings

We collect performance data from a cloud computing environment that consists of 362 servers connected by gigabit Ethernet in a local data center. The cloud servers are equipped with two to four Intel Xeon or AMD Opteron cores and 2.5 to 8 GB of RAM. We have installed hypervisors (Xen 4.16.0) on the cloud servers. The operating system on a virtual machine is Linux. Each cloud server hosts up to eight VMs. A VM is assigned up to two VCPUs, among which the number of active ones depends on applications. The amount of memory allocated to a VM is set to 2 GB. We run the RUBiS distributed online service benchmark, MapReduce and machine learning jobs from the Hadoop MapReduce benchmark suite as cloud applications on VMs. The applications are submitted to the cloud computing system through a web based interface.

We have developed a fault injection program, which is able to randomly inject four major types with 17 sub-types of faults to cloud servers. They mimic faults from CPU, memory, disk, and network. One fault was injected at a time and the time between faults was randomly distributed. We exploit third-party monitoring tools, such as sysstat [2] to collect runtime performance data in the hypervisor and VMs, and a Linux profiler perf [1] to profile performance counters from the hypervisor on each cloud server. In total, 518 metrics are profiled, i.e., 182 from the hypervisor, 182 from VMs and 154 from performance counters every minute. They cover the statistics of every component of a cloud server, including the CPU usage, process creation, task switching activity, memory and swap space utilization, page faults, interrupts, network activity, I/O and data transfer, power management and so on. We collected the performance data from the cloud testbed for two months. In total, about 520 GB performance data were collected and recorded from the cloud computing testbed during the period. The rolling time window is set as 1,440 records for a day.

An important design advantage of our self-evolving anomaly detection method is that it requires a small amount of labeled data in the collected cloud performance data and reduces the load of cloud operators in checking the cloud execution logs and data. We initialize the weights that make the predictions biased towards the normal records which are dominant. Instead of randomly initializing the weights using a

Gaussian distribution, we use $w = rand() + \alpha X^{-1} Xe$, where α is a parameter that regulates the anomaly rate and $rand()$ is a function that generates weights following a standard Gaussian distribution.

B. Experimental Results

We evaluate the anomaly detectors in terms of sensitivity and specificity. Other performance metrics, such as accuracy and F1-score, are also measured. Our anomaly detectors use the learning methods that can evolve the models. We set the number of epochs to 300 in the experiments unless otherwise specified.

Note that when detecting the type of anomalies, the number of records in each type is small as the majority of records are for normal states, which is common in production cloud systems. The imbalance problem in training may introduce a significant bias towards normal records, and the training may ignore anomalies. As a solution, we adopt an up-sampling method, i.e., SMOTE [7], to generate more anomaly records and thus balance the data distribution.

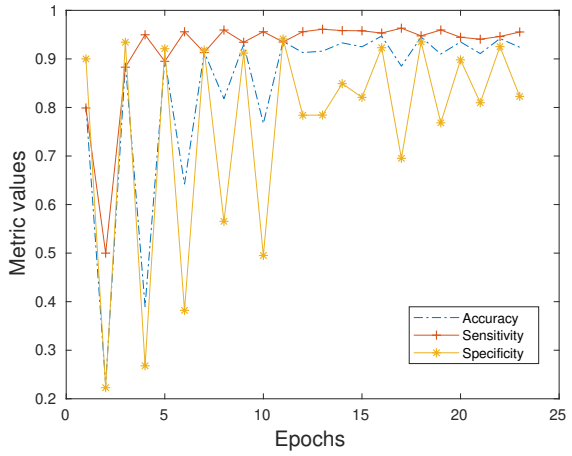

Fig. 1: Performance of a self-evolving anomaly detector using the L1LS model.

1) Performance of the L1LS Anomaly Detector: Figure 1 shows the performance of a self-evolving anomaly detector using the L1LS model. From the figure, we can see that the specificity fluctuates in early epochs and then converges to 82%-92%. The accuracy follows a similar trend, indicating that the detector is effective and converges quickly with about 10 epochs. In contrast, the sensitivity always achieves a high level (around 95%). This may be due to the weight initialization method and the fact that the normal records outnumber the failure records in the dataset.

2) Performance of the MCL21LS Anomaly Detector: An anomaly detector using the MCL21LS model is able to identify the type of anomalies in addition to anomalies themselves. Faults in the dataset consist of four major types from CPU, memory, disk, and network. Figure 2 presents the detection performance of an anomaly detector using the MCL21LS model for each fault type.

TABLE I: Performance comparison with other detection methods.

	Sensitivity	Specificity
Ensemble [11]	72.5%	-
Hybrid [9]	92.1%	83.8%
Ours(L1LS)	**94.90%**	85.99%
Ours(MCL21LS)	83.47%	**95.72%**

We plot the sensitivity, specificity and additional accuracy in the figures. Overall, the anomaly detector performs better on all anomaly types as the number of epochs increases. After about 10 epochs, the detection performance becomes stable. The average sensitivity over the last five epochs across all anomaly types reaches 83.47%, and the average specificity achieves 95.72%. Remarkably, our self-evolving anomaly detector achieves a fairly high specificity for all anomaly types, which indicates that almost all the detected anomalies are true faults.

We also plot the receiver operating characteristic (ROC) curve for each epoch as the ROC curve is independent of the data distribution and can visualize the performance of the anomaly detector. Figure 3 includes the ROC curves for the first 10 epochs. The area under the curve (AUC) is shown in Figure 4. We can see in Figures 3 and 4 that the performance of our anomaly detector becomes better as the number of epochs increases and then gets stable. The AUC stabilizes around 0.94. The high AUC value indicates that the anomaly detector would not be influenced by the imbalance in the data distribution.

From the above results, we find that 1) The anomaly detector using the MCL21LS model progressively achieves a better performance as more data are available, and becomes stable in a short period of time; 2) Our detector is able to detect different types of anomalies accurately; 3) Across all the metrics that are evaluated, our anomaly detector performs well. As demonstrated in Figures 1 and 2, our detector could be effective and practical for real-world applications.

3) Performance Comparison with Other Approaches: We also compare our self-evolving anomaly detector with other existing approaches using learning algorithms, such as an ensemble of Bayesian sub-models and decision tree classifiers [11] and a hybrid 1 and 2-class SVM [9]. We list the performance comparison results with several other approaches in Table I. As our approach employ online learning, we measure the average performance over the last five epochs in comparison.

In Table I, we can see our anomaly detector with the L1LS model achieves the best sensitivity (94.90%), but not the best specificity. The detector using the L21LS model achieves the best specificity (95.72%), but not the best sensitivity. This is because the L1LS model is designed for type-agnostic anomaly detection, while the L21LS model is tailored for type-aware anomaly detection. Beside the better performance, a major advantage is that our self-evolving detectors are capable of improving/updating the detection model continuously at

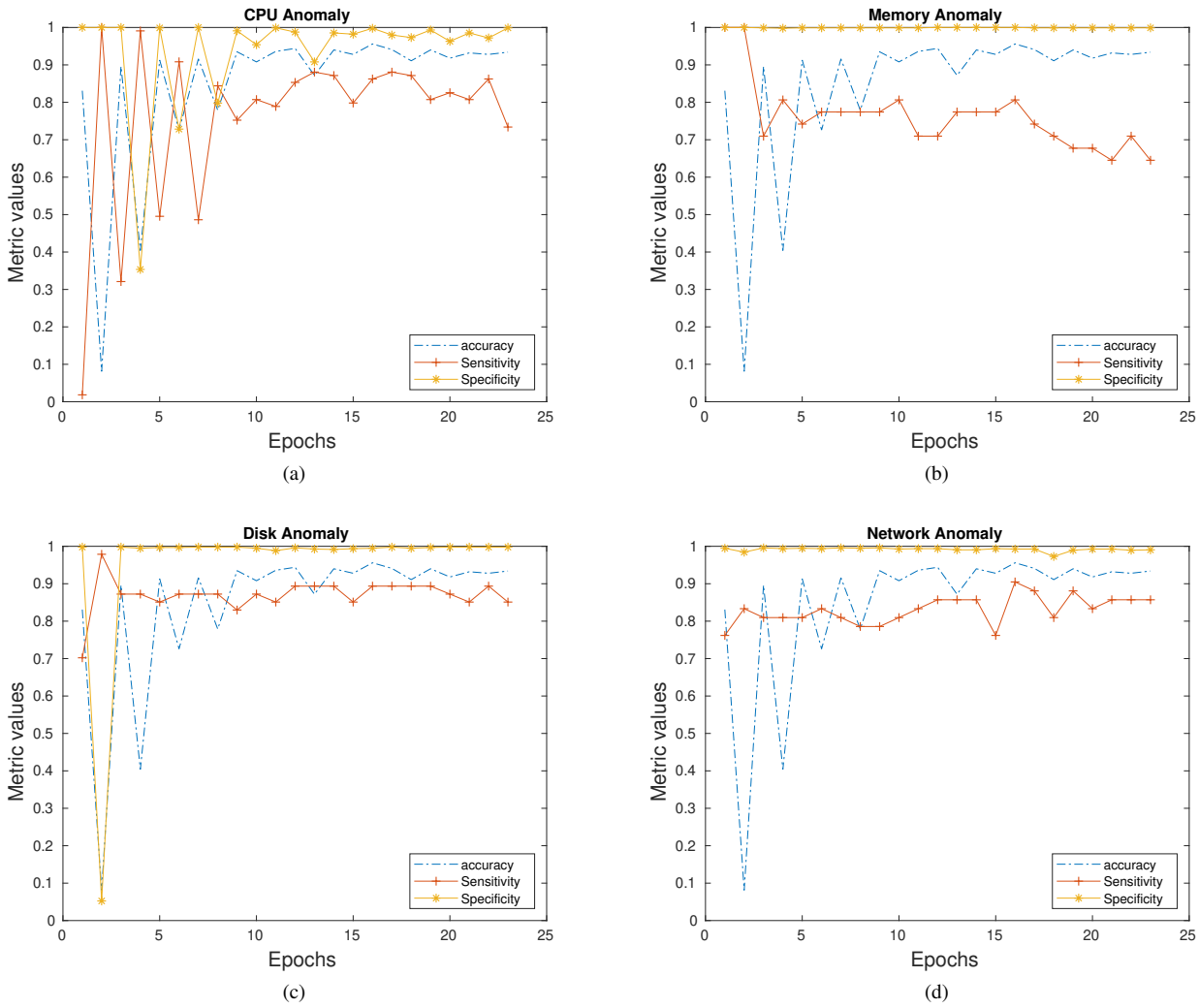

Fig. 2: Performance of an anomaly detector using the MCL21LS model. From left to right and top to bottom are the detection results on CPU, memory, disk, and network anomalies.

runtime, which is not provided by other approaches. Furthermore, our approach can adaptively select the most relevant attributes for future detections. These advantages make the self-evolving anomaly detector suitable for real-world cloud computing systems.

4) Efficiency of Anomaly Detection: Different from existing works that require labels of all training data, in our self-evolving framework, the cloud operators only check the detected records, which dramatically reduces the operators' workload in real-world deployments. As faults are rare in a system, most of the collected cloud performance records make little contribution to updating the anomaly detector. The overhead of our self-evolving anomaly detector is comparable to that of the conventional counterpart.

From Figure 5, we have the following interesting observations. 1) Our self-evolving anomaly detector achieves a similar sensitivity as the all-evolving counterpart (which retrains a new model for future detection), indicating that our detector is sensitive to the normal records. 2) Although their sensitivity is comparable, the specificity of our self-evolving anomaly detector is a little lower, as the incremental online learning leads to smaller changes/updates compared with the all-evolving counterpart. This is acceptable when we take the higher overhead due to retraining into consideration. As shown in Figure 5, we only need to label 21.06% (average over the last 10 epochs) of the original data to achieve satisfactory results. These results demonstrate the efficacy of our self-evolving anomaly detectors and justify our design of using only part of the records to update the anomaly detectors at runtime.

C. Ablation Study

To evaluate the effectiveness of our proposed approach, we run the anomaly detectors with different settings on the collected cloud performance dataset. The measurements are compiled and listed in Table II.

TABLE II: Effects of settings on the performance of self-evolving anomaly detectors.

	Self-evolving anomaly detectors (epoch size:300)															
	L1LS								MCL21LS							
SMOTE		✓			✓		✓	✓		✓			✓		✓	✓
Biased Init			✓		✓	✓		✓			✓		✓	✓		✓
Self-evolving				✓		✓	✓	✓				✓		✓	✓	✓
Sensitivity	93.70	95.38	93.98	95.05	95.47	94.93	95.71	96.06	84.81	87.49	84.81	84.01	87.49	83.47	88.10	88.94
Specificity	94.79	94.05	94.86	76.38	94.07	84.49	72.99	76.11	96.16	96.89	96.16	96.07	96.89	95.72	96.01	96.40

By leveraging SMOTE to up-sample fault data, we can see the sensitivity is improved both for L1LS and MCL21LS. They also achieve a higher sensitivity compared with the other approaches.

After applying the biased initialization module, the performance shows little difference, indicating that the biased initialization module does not influence the detection accuracy. This complies with our expectation, since this module is mainly to reduce the labeling workload.

As a key feature of our proposed anomaly detectors, we analyze the impact of the self-evolving (online learning and model updating) component on the detection performance. For the L1LS-based anomaly detector, we can see that there is a decrease in specificity after we add the self-evolving component. In contrast, the MCL21LS-based anomaly detector maintains a high specificity. An interesting observation is that the sensitivity of L1LS is often higher than its specificity, while an opposite trend is observed for MCL21LS. A possible reason is that the L1LS model is designed for type-agnostic anomaly detection, while the MCL21LS model is for type-aware anomaly detection, which can detect both anomalies and their types.

D. Attribute Selection

We also study the performance of attribute selection. For this purpose, we investigate L1LS and MCL21LS models. As shown in Equations (3), (5), and (8), the objective functions consist of two modules. The first one is a loss function that aims to fit the distribution of the input data, and the second one is the regularization term that aims to avoid over-fitting and improve the model's generalization. As a trick, we use the L1/L21-norm based regularization instead of the traditional squared L2-norm based regularization. This provides sparsity [6], [32] for the pursuant weights and makes our models able to select the most relevant attributes. To evaluate the effect from each attribute, we rank the attributes by the L2-norm distance of the corresponding rows in the weight matrix. Experimentally, we tune the parameter λ in the set of $[0.01, 0.1, 1, 10]$. The top ten attributes are selected.

The detailed results are provided in Tables III and IV. We observe that the selected attributes are relevant and critical, indicating that our approach is effective. Further research will be performed to analyze the selected features. For example, we will analyze the relationship between the detected anomalies and the selected attributes, aiming to predict and prevent system failures and improve cloud reliability. We also observe that MCL21LS selects almost the same attributes as L1LS, when $\lambda \in \{0.01, 0.1, 1\}$. This indicates that the selected features would not change much for different detection tasks and could be consistent. As a future study, We will correlate the results from selection with the value/range of λ. Experimental results show that when $\lambda = 10$, the weights of the attributes are rather close. However, when $\lambda \in \{0.01, 0.1, 1\}$, the weights are different. Therefore, we select λ as 0.01 in the experiments.

V. Conclusion

In this paper, we present a new online self-evolving anomaly detection framework for cloud computing systems. It features a self-evolving capability of updating and improving the anomaly detector at runtime without retraining or a prior failure history. From experiments, We show that with only 21.06% of labeled records, our self-evolving detector can achieve a comparable performance with the existing approaches. To improve the detection accuracy, we design an L1LS model for type-agnostic anomaly detection and MCL1LS and MCL21LS models for type-aware anomaly detection. In the experimental evaluation, our anomaly detectors achieve a consistent improvement in both sensitivity and specificity with the self-evolving functionality. Moreover, in contrast to the existing approaches, our detectors have the capability of selecting the most relevant attributes for detection refinement. By carefully tuning the hyper-parameters of the detectors, we will further improve the detection performance.

By combining the self-evolving capability and the designed L1LS and MCL21LS models, our anomaly detectors naturally integrate the online learning, labeling effort reduction, and feature selection with anomaly detection. We also note that due to the application of SGD, our design does not follow the linear space. As future research, we plan to explore non-linear kernel projection methods to analyze and further enhance our anomaly detectors. The frequency of data collection is one minute in our experiments. We note this is not fast enough for certain anomalies, such as those caused by attacks. Our design works for higher data collection frequencies. We will evaluate the efficiency of our detectors when processing data coming at a higher speed.

Acknowledgement

This work has been supported in part by the U.S. National Science Foundation grants CNS-2113805, CNS-1852134, OAC-2017564, ECCS-2010332, CNS-2037982, CNS-1563750, DUE-2225229, and CNS-1828105.

TABLE III: Attribute selection by an anomaly detector using the L1LS model.

rank	$\lambda = 0.01$		$\lambda = 0.1$		$\lambda = 1$		$\lambda = 10$	
	weights	Attributes	weights	Attributes	weights	Attributes	weights	Attributes
#1	1.1563	runq-sz	1.1378	runq-sz	0.6223	runq-sz	0.0337	oseg/s
#2	0.8523	plist-sz	0.7171	ldavg-1	0.3141	plist-sz	0.0322	%vmeff
#3	0.7992	ldavg-1	0.7068	plist-sz	0.2975	cswch/s	0.0318	%soft 4
#4	0.6899	tcpsck	0.5946	tcpsck	0.2794	tcpsck	0.0267	idel/s
#5	0.5683	cswch/s	0.564	cswch/s	0.2744	ldavg-1	0.0261	txpck/s eth0
#6	0.5178	bufpg/s	0.4248	totsck	0.2306	udp6sck	0.026	orq/s
#7	0.4177	svctm dev8-0	0.3811	bufpg/s	0.1696	%util dev253-1	0.0238	pgsteal/s
#8	0.3991	totsck	0.3382	%soft 1	0.1357	totsck	0.0222	%usr 4
#9	0.3556	%soft 1	0.3321	svctm dev8-0	0.1194	pgscank/s	0.0221	tps
#10	0.3528	kbcached	0.2987	ldavg-15	0.1112	%usr 4	0.0215	%util dev253-0

TABLE IV: Attribute selection by an anomaly detector using the MCL21LS model.

rank	$\lambda = 0.01$		$\lambda = 0.1$		$\lambda = 1$		$\lambda = 10$	
	weights	Attributes	weights	Attributes	weights	Attributes	weights	Attributes
#1	2.0591	runq-sz	1.907	runq-sz	1.5179	runq-sz	0.1649	%sys all
#2	1.3902	plist-sz	1.5266	plist-sz	1.3218	ldavg-1	0.1649	rd_sec/s dev253-1
#3	1.2291	ldavg-1	1.3514	ldavg-1	1.2645	plist-sz	0.1648	rxpck/s eth0
#4	1.1666	tcpsck	1.2505	tcpsck	0.9829	tcpsck	0.1648	%sys 5
#5	0.9077	cswch/s	0.8944	cswch/s	0.94	cswch/s	0.1648	%sys 2
#6	0.8969	bufpg/s	0.8323	bufpg/s	0.88	bufpg/s	0.1647	totsck
#7	0.6911	rxdrop/s eth0	0.7129	totsck	0.6897	kbbuffers	0.1647	majflt/s
#8	0.5885	await dev8-0	0.6536	rxdrop/s eth0	0.6383	totsck	0.1647	intr/s
#9	0.5835	kbbuffers	0.6048	await dev8-0	0.5802	ldavg-15	0.1647	%iowait all
#10	0.5635	kbcached	0.5586	proc/s	0.5789	await dev8-0	0.1646	tps dev253-1

REFERENCES

[1] *perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/.*

[2] *sysstat utilities: a collection of performance monitoring tools for Linux. http://sebastien.godard.pagesperso-orange.fr/.*

[3] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, 2013.

[4] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," *arXiv preprint arXiv:1811.05269*, 2018.

[5] A. Borghesi, M. Molan *et al.*, "Anomaly detection and anticipation in high performance computing systems," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[6] G. Cai, R. Zhang, F. Nie, and X. Li, "Feature selection via incorporating stiefel manifold in relaxed k-means," in *Proceedings of the 25th IEEE International Conference on Image Processing (ICIP)*, 2018.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[8] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics, 2001, vol. 1, no. 10.

[9] S. Fu, J. Liu, and H. Pannu, "A hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines," in *Proceedings of the International Conference on Advanced Data Mining and Applications*, 2012, pp. 726–738.

[10] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2013.

[11] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems," *Journal of Communications*, vol. 7, no. 1, pp. 52–61, 2012.

[12] T. Hagemann and K. Katsarou, "A systematic review on anomaly detection for cloud computing environments," in *AICCC*, 2021.

[13] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning," *arXiv preprint arXiv:1704.07706*, 2017.

[14] J. Huang and X. Yan, "Related and independent variable fault detection based on kpca and svdd," *Journal of Process Control*, vol. 39, pp. 88–99, 2016.

[15] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*. John Wiley & Sons, 2004, vol. 46.

[16] I. Jolliffe, *Principal component analysis*. Springer, 2011.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[18] P. Mianjy and R. Arora, "Stochastic pca with l_2 and l_1 regularization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[19] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE access*, vol. 6, pp. 47 980–48 009, 2018.

[20] A. B. Nassif, M. A. Talib *et al.*, "Machine learning for anomaly detection: A systematic review," *IEEE Access*, vol. 9, pp. 78 658–78 700, 2021.

[21] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the 21st International Conference on Machine learning (ICML)*, 2004.

[22] F. Palmieri, U. Fiore, and A. Castiglione, "A distributed approach to network anomaly detection based on independent component analysis," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1113–1129, 2014.

[23] G. Pang, C. Shen *et al.*, "Deep learning for anomaly detection: A review," *ACM Computing Surveys*, vol. 54, pp. 1–38, 2021.

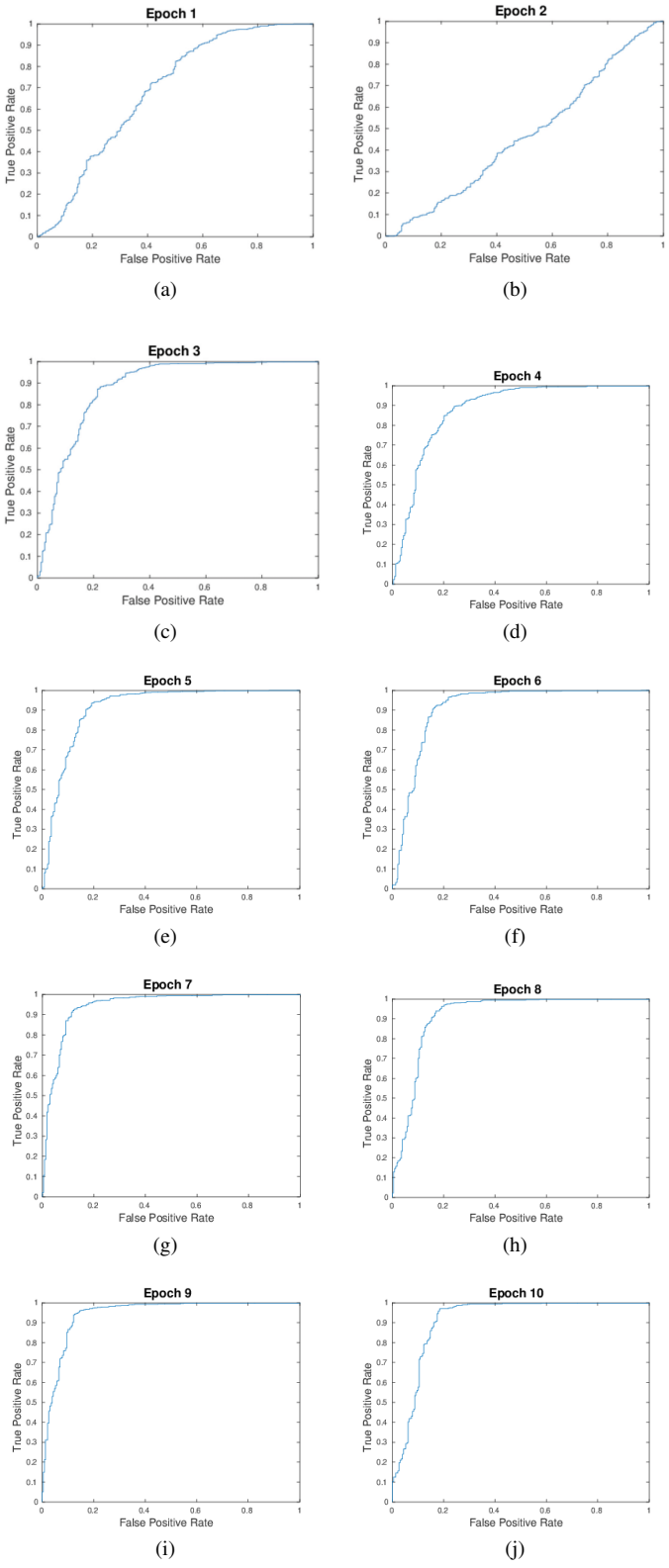

Fig. 3: The receiver operating characteristic (ROC) curves for the first 15 epochs.

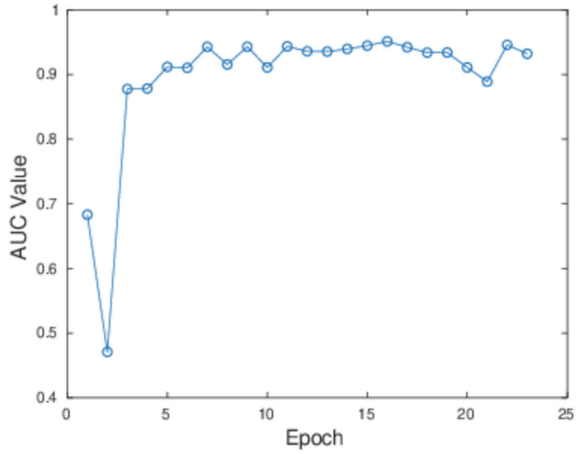

Fig. 4: The area under the curve (AUC) for epochs.

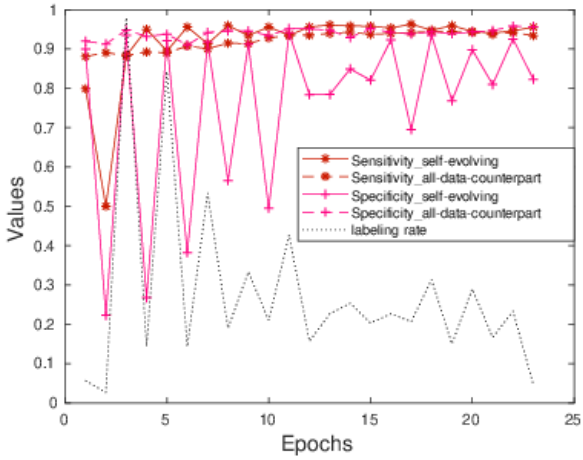

Fig. 5: Performance of self-evolving anomaly detectors compared with other detection methods.

[24] H. S. Pannu, J. Liu, and S. Fu, "Aad: Adaptive anomaly detection system for cloud computing infrastructures," in *Proceedings of the 31st IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2012.

[25] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI workshop on empirical methods in artificial intelligence*, 2001.

[26] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International Conference on Machine Learning*, 2018, pp. 4390–4399.

[27] D. Smith, Q. Guan, and S. Fu, "An anomaly detection framework for autonomic management of compute cloud systems," in *Proceedings of the 34th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2010.

[28] N. Subramanian and A. Jeyaraj, "Recent security challenges in cloud computing," *Computers & Electrical Engineering*, vol. 71, pp. 28–42, 2018.

[29] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[30] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011.

[31] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu, "Cloud computing: a perspective study," *New generation computing*, vol. 28, no. 2, pp. 137–146, 2010.

[32] J. Wen, X. Fang, J. Cui, L. Fei, K. Yan, Y. Chen, and Y. Xu, "Robust sparse linear discriminant analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 2, pp. 390–403, 2019.