Computer Networks and Communications
https://ojs.wiserpub.com/index.php/CNC

Research Article

Dynamic Real Time Framework for Abnormal Detection of IMS Core in Kubernetes Cloud

Rasel Chowdhury[1*], Saad Inshi[1], Hakima Ould-Slimane[2], Chamseddine Talhi[1], Azzam Mourad[3,4]

[1] Department of Software Engineering and Information Technology, Ecole de Technologie Superieure, Montreal, Canada
[2] Department of Mathematics and Computer Science, University of Quebec at Trois-Rivieres, Trois-Rivieres, Canada
[3] Khalifa University 6G Research Center, Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates
[4] Artificial Intelligence and Cyber Systems Research Center, Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon
E-mail: rasel.chowdhury.1[at]ens.etsmtl.ca

Received: 11 February 2025; **Revised:** 28 April 2025; **Accepted:** 29 April 2025

Abstract: In the ever-evolving telecommunications sector, advancing from 5G towards 6G, maintaining the security of core infrastructures has become supreme. This study addresses the critical need for proactive and real-time anomaly detection within cloud-native environments. Leveraging cloud-native implementations within Kubernetes clusters, our framework utilizes advanced machine learning techniques to analyze data from applications and clusters. Specifically, this paper introduces a novel integration of k-means clustering and Long Short-Term Memory (LSTM) models for real-time anomaly detection in Kubernetes-based cloud-native environments, offering a unified framework capable of addressing both global and local anomalies across multiple layers of the IP Multimedia Subsystem (IMS) core. Existing research on anomaly detection in Kubernetes environments often focuses on specific application layers or isolated metrics, lacking a comprehensive solution that addresses the multidimensional and dynamic nature of IMS core anomalies across application, pod, and node levels in real time. By employing k-means clustering and LSTM models, our approach achieves approximately 90% accuracy in anomaly detection. Extensive experiments with various model versions demonstrate the effectiveness of the framework, ensuring robust security and reliability for next-generation telecom networks.

Keywords: anomaly detection, security, cloud native, Virtualized IP Multimedia Subsystem (VIMS), machine learning, LSTM

Abbreviation

CapEx	Capital expenditure
CNN	Convolutional neural network
CP	Control Point
CSCF	Call Session Control Function
DB	Database
GAN	Generative Adversarial Networks
GD	Global Anomaly Detector

IMS	IP Multimedia Subsystem
IoE	Internet of Everything
IoT	Internet of Things
IoV	Internet of Vehicles
LD	Local Anomaly Detector
LSTM	Long-Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
NFV	Network Function Virtualization
OpEx	Operational Expenses
SLA	Service Level Agreement
UMTS	Universal Mobile Telecommunications System
VNF	Virtual Network Functions
vEPC	Virtualized evolved packet core
vIMS	Virtualized IMS
Vo5G	Voice of 5G
VoIP	Voice over IP

1. Introduction

Ensuring the security of telecom networks is critical, especially since the industry connects everything through the internet, radio networks, and various communication infrastructures. According to the Ericsson Mobility Report [1], a staggering one billion mobile subscribers use multiple smart devices, leading to an average data consumption of around 19 GB per user per month in 2023. This enormous data usage generates exabytes of data that the telecom industries must manage efficiently. Given this massive data generation, telecom networks have become the backbone of modern communication, providing essential services such as text messaging, VoIP, Internet connectivity, and emergency services. These networks also interconnect systems like IoT, IoE, and IoV. To meet user demands and ensure satisfaction, the telecom industry has rapidly evolved from simple radio communication to VoIP and now to Vo5G, which is becoming the standard for communication. This swift transition is driven by the need for improved mobile broadband, ultra-reliable communication, and low latency [2].

The IMS plays a pivotal role in this evolution. Initially designed to evolve UMTS networks and provide multimedia services using the Internet Protocol to mobile users, IMS has become a core component of 3G and next-generation fixed telecom networks [3]. Traditionally, the IMS core operates on proprietary, over-provisioned hardware, resulting in high CapEx. To reduce both CapEx and OpEx, telecom industries are shifting to vendor-neutral, software-based solutions by virtualizing the IMS core, running it as a cloud-native application using microservices or leveraging VNF and NFV technologies. IMS, standardized by 3GPP, enables various multimedia services such as MMS image sharing, VoIP, instant messaging, and more [4].

Microservices have gained popularity in cloud-native solutions due to their independent, decoupled application modules that communicate using language-agnostic APIs [5]. For instance, Lu et al. [6] proposed a virtualization-based cloud platform for the IMS core network, addressing challenges like low utilization, poor scalability, and high costs. They demonstrated that the IMS mechanism allows dynamic resource allocation, VM live migration, load balancing, and disaster protection. However, as telecom industries adopt these technologies, they encounter limitations in orchestration platforms like Kubernetes, Red Hat OpenShift, Docker Swarm, and Amazon EKS [7]. These challenges include performance models, standardization of metrics, and limited policies for orchestration. As Hawali et al. [8] proposed a NFV framework that bundles multiple functions of a vEPC in a single physical device, it is evident that optimizing resource management remains a significant focus in the industry.

Moreover, the security of IMS cores, which control cellular networks and support VoIP and Internet services, is a significant concern. Attackers can exploit vulnerabilities in the IMS core through various vectors, including denial of service, unauthorized access, billing errors, and other types of cyberattack [9]. Furthermore, IMS cores are susceptible to failures due to hardware and software issues, as well as cyber attacks, leading to abnormal behaviors such as jitters, latency, call quality problems, memory leaks, and CPU usage spikes [10], [11]. Therefore, addressing these security issues is paramount. Techniques such as the use of microservices for authentication and authorization [12], and NFV-based vIMS architecture for handling UE registration and call control [13], highlight the ongoing efforts to enhance IMS security and efficiency.

Despite the advancements in anomaly detection techniques for cloud-native environments, existing research predominantly focuses on isolated application layers or specific metrics, leaving a critical gap in addressing the multidimensional and dynamic nature of anomalies within Kubernetes clusters. To bridge this gap, this study introduces a comprehensive framework that combines k-means clustering for global anomaly detection with LSTM models for local anomaly detection. This novel integration enables real-time identification and analysis of anomalies across the IMS core's application, pod, and node layers, providing a unified solution that significantly enhances the reliability and security of next-generation telecom networks. Through extensive experiments with different machine learning algorithms, we identified the most effective methods for real-time anomaly detection. For example, Pereira et al. [14], [15] proposed LSTM-based models to classify SIP dialogs for anomaly detection, demonstrating the efficacy of advanced machine learning techniques in this domain. The contributions of this paper can be summarized as follows:

• A dynamic, real-time framework for detecting abnormal behaviors in IMS cores within Kubernetes cloud environments.

• An innovative architecture for real-time anomaly detection in a cloud-native setting.

• Intelligent dynamic detection of global and local anomalies for the cloud-native vIMS using k-means clustering and LSTM.

• Extensive experiments with different algorithm variants to determine the best fit for each type of anomaly.

The remainder of the paper is organized as follows. Section 2 presents the review of the background literature related to IMS security vulnerabilities and the cloud native IMS fault detection mechanism. Section 3 describes the architecture, problem definition, and schemes. Section 4 presents the experiments and the performance evaluation. Finally, Section 5 concludes the paper and provides future directions.

2. Background and literature review

IMS is a global, access-independent, standard IP connectivity and service control architecture that enables various types of multimedia services to end-users using common Internet-based protocols. IMS is standardized by 3GPP [3], [4] and is used to provide different services such as MMS image sharing, VoIP multimedia telephony, instant messaging (SMS) and presence service, peer-to-peer video sharing, Push-to-talk, real-time text for users.

Virtualizing the IP Multimedia Subsystem is the moving of the proprietary over-provisioned hardware-based IMS cores into modular software components in the cloud. The main goal of the IMS core virtualization is to reduce CapEx and OpEx as well as to use the features like portability, computations, etc. that the clouds are providing, and the standards specified by 3GPP. Some of the recent research on vIMS is as follows. Lebdeh et al. [16] discussed different requirements for cloudifying IMS services, as well as they have evaluated different available cloudification techniques of IMS architecture. Lu et al. [6] proposed a virtualization-based cloud platform for the IMS core network that met the three main challenges of the current IMS infrastructure (low utilization, poor scalability, and high cost). Their proposed architecture for the virtualized IMS core allows dynamic resource allocation, VM live migration, load balancing, and disaster protection. They have shown that the IMS mechanism allows dynamic allocation based on the workload and is able to recover from disaster in seconds using live mitigation of VMs.

The authors in [17] provided an introduction to an IMS based virtualized telecommunications-specific cloud manager called Telecom Cloud Manager as an extension of the 4G/LTE system defined by the 3GPP. They have used OpenStack (as NFV MANO) for their simulation platform, and their experimentation was done using two IMS core and showed that the

CPU, memory, and network throughput for one vIMS core as well as when the platform running two instances of vIMS core.

In [12], the authors used the microservices approach to disassemble the traditional IMS core as functional VNF services. Their microservice approach is mainly focused on authentication and authorization procedures of the IMS service. This approach allows the design of VNFs based on atomicity, statelessness, and loose coupling. The approach is divided into three steps: (1) functional decomposition by identifying the functionalities of the IMS so that the NF designs can be made into atomic services, (2) separation of state achieved through the granularity levels of a network service separating the service logic of functional entities and the handled data or state, and (3) finally functional independence by removing the predefined sequences between the IMS functions.

In [13], the authors proposed an NFV-based vIMS architecture that decomposes the S-CSCF into two functions (Registration function and Control Function). One handles the UE registration combined with I-CSCF [vI-CSCF(R)] and the other processes the call control [vS-CSCF(C)]. This decomposition allows for reducing load on the S-CSCF (as it does not have to deal with the registration process). They also implemented the architecture using NFV based vIMS (project Clearwater) to evaluate the feasibility and success rate of SIP with different available vIMS (Clearwater SProutx1, Clearwater Sproutx2).

Hawali et al. [8] propose a NFV framework in the virtual environment as well as a criterion to bundle multiple functions of a vEPC in a single physical device or a group of adjacent devices. The Virtual Resources Manager, the VNF Manager, and the Orchestrator are grouped at the hypervisor level as their NFV framework. VNFs for the vEPC are grouped together based on their interactions and workload (four groups). Using the grouping method allows 70% of reduced network control traffic.

The authors [18] presented a NFV fault management system to monitor NFVI using Playnet-MANO. Their fault management technique manages NFVI faults for hardware and virtual resources, and VNFs. The system continuously monitors the VNF for resource utilization by the monitoring server and raises an alarm based on the administrator's configuration. The alarm is classified as warning, critical, and disaster. For alarm correlation, they have used interlayer correlation between resources and VNFs, VNFs and the network links, as well as the resources metrics of the VNFs and the physical system. Based on the alarm, the action controller of the NFVO takes action from the database based on the action policies set by the administrator.

The authors in [19] designed a fault tolerance for IMS control plane operations. Their design provides modular redundancy to perform real-time failure recovery for high service availability in vIMS. The failure detection is done through Finite State Machine (FSM) where different FSM states keep track of different stages of SIP procedure as well as failure recovery procedure. Raza et al. [20], [21] refactored the IMS NFs modules by pipelining the processing of the data packets and fetching its control instructions, as well as reconfiguring the modules to recover from a failure.

Like software components, vIMS are prone to failure due to the irregular behavior of the system. Fault tolerance is one of the main criteria which allows a system to work normally in the event of a failure of some of its components. Anomalies are rare items, events, or observations which raise suspicions by differing significantly from most of the data. Anomalies can be caused by jitters, latency, call quality, memory leaks, CPU usage, and abnormal behaviors of counters, loops in the CSCF and attacks from external entities [9], [10]. There are many challenges for anomaly detection in real time as the data have a high dimensionality and volume coming from different sources at multiple velocity which have a variety of values and veracity [22]. There are also different types of anomalies [23] namely:

- Global anomaly or point anomaly is a single instance of data where it is too far off from the rest of the data instances;
- Local anomaly or contextual is based on specific context or features;
- Micro cluster or collection is a set of data instances collectively that is away from the rest of the data points.

According to the survey provided by [22], there are different ways to detect anomalies using supervised learning, semi-supervised learning, and unsupervised learning. Some of the popular techniques are the k-nearest neighbor, local outlier factor, isolation forests, support vector machines, neural networks, auto-encoders, variational auto-encoders, LSTM, etc.

The authors proposed in [24] proposed a docker container anomaly monitoring system that can monitor the multidimensional resource metric, automatically adjust the monitoring period, and analyze the cause of the anomalies using an optimized isolation forest algorithm that sets weights for different resource metrics and can locate the anomalous resource metric considering the type of container application workload.

[25] proposed an unsupervised anomaly detection approach based on the reconstruction error of the deep autoencoder model for time series data using density-based spatial clustering of applications with noise (DBSCAN) for batch processing. Lin et al. [26] implemented a variable auto-encoder model to summarize the local information of a short window into a low-dimensional embedding. The LSTM model, which acts on the low-dimensional embeddings produced by the VAE model, manages the sequential patterns over longer term, in which the hierarchical structure detects anomalies occurring over both short and long periods.

[27] discusses the problem of automatic anomaly detection for Prometheus metrics with forecasting. It explains the concepts of logs, metrics, and traces, and their importance in monitoring and understanding system behavior. The document also introduces the idea of using forecasting methods to detect anomalies in metrics, rather than relying on hard-coded values for alerting. It discusses different types of forecasting methods and their relevance to the problem at hand. The document also presents an architectural setup for implementing the proposed solution, including a centralized structure with a manager node and worker nodes. Overall, the document aims to analyze current solutions, identify areas for improvement, and propose a new solution for observability in a Kubernetes cluster.

[14] provides a comprehensive and systematic review of the use of Transformers in time series modeling. Transformers have been successful in natural language processing and computer vision tasks, and their ability to capture long-range dependencies and interactions makes them attractive for time series analysis. The paper examines the development of time series Transformers from two perspectives: network structure and applications. It summarizes the adaptations and modifications made to Transformers to address the challenges in time series analysis and categorizes time series Transformers based on common tasks such as forecasting, anomaly detection, and classification. The authors also perform empirical analysis to study how Transformers perform in time series, including robust analysis, model size analysis, and seasonal-trend decomposition analysis.

[15] discusses the implementation and testing of a cloud-native infrastructure. The infrastructure includes features definition, data collection, pipeline deployment, and Chaos Engineering experiments. The article also presents the results and discussion of offline and online learning experiments, including the selection of anomaly detection models and the deployment of a voting system for real-time anomaly detection.

[28] illustrates various methodologies and findings from a series of papers related to cybersecurity and anomaly detection in Kubernetes environments. The authers acknowledges the limitations of existing approaches, including the absence of automated actions upon alert occurrence, the inability to trace back to the application action that caused the alerts, and the need to train a dedicated model for each application being monitored. Furthermore, the article emphasizes the importance of network monitoring in analyzing infrastructure baselines and identifying abnormal behaviors. Overall, it provides a comprehensive summary of the methodologies and findings from various papers, highlighting their strategies and achievements in the field of Kubernetes dataset analysis and anomaly detection.

[29] proposed a classification algorithm that is capable of detecting abnormality in the SIP header for invalid syntax, invalid values, SQL query, etc. [30], [31] proposed four LSTM-based classification models to classify the SIP dialog for anomaly detection. They have compared their work using two other methodologies based on k-means and a semisupervised threshold-based classifier. [32] introduced a LSTM model for predicting the VoIP traffic and compare with the system's traffic generated. They are able to spot anomalies using a real-running VoIP system. [33] showed the comparison of two different deep learning LSTM and CNN, including their proficiency in detecting abnormality in SIP messages.

In summary, the techniques and schemes discussed above illustrate various ways the IMS can be virtualized using VNFs and NFVs for a cloud-native approach and fault detection. However, none of the existing works fully address the need for automated abnormality detection in the IMS application, the container, and the virtual machine where it is running. To address this gap, we have developed a scheme capable of detecting abnormalities across these layers and have experimented with multiple algorithm variants to determine the most suitable for each type of anomaly.

While these existing methods provide significant advancements in anomaly detection, they often focus on specific aspects of the system, such as individual applications or protocols, and do not offer a holistic solution encompassing the entire Kubernetes environment. In contrast, our research presents a comprehensive framework for anomaly detection, leveraging both k-means clustering and LSTM models to analyze data from applications, pods, and nodes within Kubernetes clusters. This approach provides a unified and robust solution for real-time anomaly detection across all layers of the vIMS environment.

The Table 1 shows the summary and differences between our proposed scheme and the other related works, also points to several gaps identified in existing methodologies.

Table 1. Comparison of related work along with proposed method

Reference	Methodology	Application	Pod			Node		
			CPU	Memory	Network	CPU	Memory	Network
[30], [31]	LSTM	✓	✗	✗	✗	✗	✗	✗
[29]	Classification	✓	✗	✗	✗	✗	✗	✗
[32]	LSTM	✓	✗	✗	✗	✗	✗	✗
[33]	LSTM, CNN	✓	✗	✗	✗	✗	✗	✗
[18]	Threshold	✓	✓	✓	✗	✗	✗	✗
[19]	Finite State Machine	✓	✓	✓	✗	✗	✗	✗
[20], [21]	Classification	✓	✓	✓	✗	✗	✗	✗
Our approach	K-means, LSTM	✓	✓	✓	✓	✓	✓	✓

3. Methodology

3.1 *Architecture*

Our proposed scheme targets the cloud-native monitoring and orchestration for the detection of abnormal behavior of IMS application for anomalies caused by the applications as well the CPU and Memory. Figure 1 shows the high-level overview of the architecture for detecting global anomalies and local anomalies hosted in a Kubernetes cloud-native cluster. The work flow of our approach is shown in Figure 2. Our approach saves resources and time by performing a single global detection of all metrics instead of performing individual anomaly detection on all metrics to pin point where the anomaly has occurred.

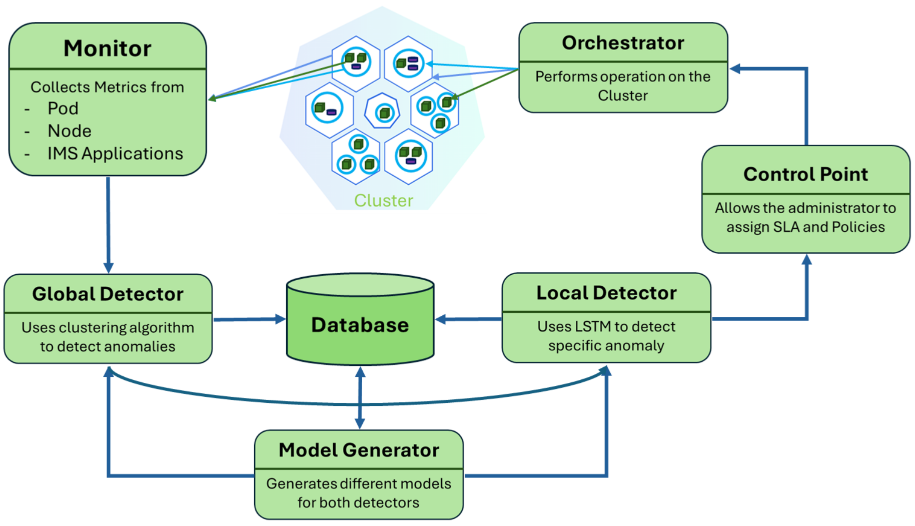

Monitor
Collects Metrics from
- Pod
- Node
- IMS Applications
Cluster
Orchestrator
Performs operation on the Cluster
Control Point
Allows the administrator to assign SLA and Policies
Global Detector
Uses clustering algorithm to detect anomalies
Database
Local Detector
Uses LSTM to detect specific anomaly
Model Generator
Generates different models for both detectors

Figure 1. Proposed architecture

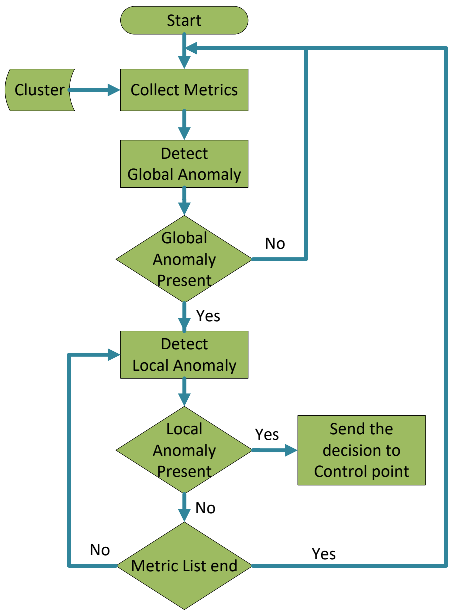


Figure 2. Work flow

The architecture collects data from different sources, namely applications, pods, and nodes. Then, we perform a behavior analysis on the data received from different sources to decide whether the application is performing optimally or if there is an anomaly at that moment. The Monitor collects data from the application running on the pod which is hosted in a node of a Kubernetes cluster. Afterwards, it aggregates the collected data and transfers them to the GD. GD performs an ML operation and decides whether the collected data is normal or not. If there is abnormal behavior, it forwards the data to the LD. Otherwise, it stores the decision in the DB. LD performs multiple LSTM operations on the data and tries to find in which part of the application or the hardware the anomaly has occurred. Once an anomaly is detected, the LD sends the name of the metric where the anomaly occurred to the CP. Based on the SLA provided by the administrator, it sends the type of operation required by the Orchestrator to perform on the application, the pod, or the node. Finally, the Orchestrator performs the desired operations based on the CP directives. All modules run on a Kubernetes cluster. The detection and operation are performed immediately without any latency. In the sequel, we present the details of each module.

3.1.1 *Monitor*

The monitor is the entry point for data collection from the different sources. It collects the data, aggregates them into a sequence format using metrics as features, and then sends the data to the GD for processing. The metrics collected from the sources are divided into three categories:

- Application data are collected from the data generated from the IMS application itself, running in the containerized cluster,
- Pod data are the resource utilization of the Kubernetes pod where the container is situated,
- Node data are the server resource utilization where the pod is being hosted.

The monitor is implemented using the Python Prometheus API. The Monitor performs queries in the Kubernetes cluster, finds the list of all the deployments (applications) running in the cluster, selects one application at a time, and performs queries to retrieve the data from the IMS application, pod hardware resource utilization, and node resource utilization. Once all data collection is done from all sources, it is aggregated along with a time stamp and sent to the GD to perform the ML operation. The details regarding the data collected as features and the sources from which it is collected are shown in Table 2.

Table 2. Types of metrics collected

Data	Application	Pod	Node
SIP request code	✓	✗	✗
Number of sip request	✓	✗	✗
SIP response	✓	✗	✗
SIP latency	✓	✗	✗
CSCF dropped ratio	✓	✗	✗
CSCF seizure ratio	✓	✗	✗
CSCF success ratio	✓	✗	✗
CSCF session ratio	✓	✗	✗
CSCF session setup time ratio	✓	✗	✗
Pod name	✗	✓	✓
Node name	✗	✓	✓
Assigned CPU	✗	✓	✓
Assigned memory	✗	✓	✓
CPU utilization	✗	✓	✓
Memory utilization	✗	✓	✓
Assigned network speed	✗	✓	✓
Network utilization	✗	✓	✓

3.1.2 *Global detector*

GD performs global anomaly detection on data received from the Monitor. Once a data is received, it uses a model generated by the Model Generator for all features/metrics to decide whether an anomaly has occurred or not in that aggregated data. When the GD receives the data, it normalizes the data and categorizes them according to their features. To detect the global anomaly, we are using K-means clustering to decide whether the data are caused by an anomaly or whether the application and hardware are in a normal state. If the data are classified as benign, it sends the data along with the decision to the time series database for storage. GD is implemented using Python along with the necessary libraries required for the clustering algorithm.

3.1.3 *Local detector*

LD is responsible for finding the appropriate anomaly point in the system. Once anomalous data are received from the GD, it performs multiple time series LSTM operations on the data to find the exact source of the anomaly. Once the source is identified from the data. It sends the name of the source to the CP. The LD is implemented using Keras. When the LD receives the data and performs feature extension, it normalizes the data and predicts the data for all features.

3.1.4 *Control point*

Control Point is an interface for the application administrator to assign policies, SLA to the cluster with respect to the type of operations required for each type of anomalies. This module is the only way for the administrator to access the system for intervention and updating of policies. Control Point allows the administrator to set different parameters from the service-level features, like restarting the pod or application.

3.1.5 *Database*

Database is required to store the data generated by the system and the operation decision in the time series database, which later will be used by the Model Generator to generate models for anomaly detection. For implementation of the database, we have used InfluxDB for its popularity to store time series data.

3.1.6 *Orchestrator*

Orchestrator enforces the orchestration operation on the Kubernetes cluster. It has the necessary functionalities to execute the appropriate commands to perform the operation for restarting the application instances, pods, and as well as the nodes. This module updates the Kubernetes cluster with the decision provided by the CP based on the best possible orchestration solutions. Implementation of the orchestrator is done using the Kubernetes Python API. At the moment, we kept only one operation that is restarting the pod, node, or running instance of the application.

3.1.7 *Model generator*

Model generator generates multiple Machine Learning models using unsupervised learning for global anomalies as well as local anomalies. The model generator periodically collects the data from the database to generate models for the local anomaly detector and the global anomaly detector. Once the dataset is collected from the database, it expands the time feature into hours, days of the week, and days of the year as a number and then passes the dataset to the ML training algorithms for model generation. The details regarding the formulations and algorithms are explained later in this section.

3.2 *Implementation*

This section explains the procedure for solving the global anomaly and local anomaly problem, as well as the threshold for calculating the abnormality from the data received from the Monitor.

3.2.1 *Threshold calculation*

To detect the abnormality in the IMS application, we need a threshold value which is calculated using the regular traffic. If the current value is higher than the threshold of the predicted value, then an abnormality has occurred at that moment. In our research, we have used three types of thresholds as follows:

• Mean Squared Error is the average squared value of the difference between the received and predicted values. The equation is given by:

$$\text{MSE} = \frac{1}{n}\sum(y-y')^2 \tag{1}$$

• Mean Absolute Error is a distance of errors between received value paired with the predicted value. The equation is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\|y-y'\| \tag{2}$$

• Mahalanobis' distance is a measure of the distance between a predicted value and a standard distribution of the received value. The equation is given by:

$$\text{Mahalanobis} = \sqrt{(y-y')^2 S^{-1}(y-y')} \tag{3}$$

where, S is a positive-definite covariance matrix and μ is the mean and S is derived using $S = \frac{1}{m}\sum_{i=1}^{m}(y^i-\mu)^T(y^i-\mu)$

3.2.2 *Global anomaly*

Global anomaly detection can be seen as a classification problem, where we have a set of metrics as features $(x_1, x_2, \cdots, x_n)$ and each set has a dimensional vector of d, which is in the Euclidean space and is NP-hard even for two groups [34]. The problem of classifying them into groups will take around $\text{O}(n^{dk+1})$ [35]. So the problem is a computationally NP-hard problem. Therefore, to classify into two main groups, k-means clustering is used to partition n observations into sets of k $S = S_1, S_2, \cdots, S_k$. The objective function is defined as follows:

$$\arg\min \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_k\| = \arg\min \sum_{i=i}^{k} |S_i| \operatorname{Var} S_i \tag{4}$$

where, μ_i is the mean of points in S_i

Algorithm 1 Global detector

```
Require: Metrics, model, MAE
  Find the cluster metrics belongs to using model Equation (4)
  Find the euclidean distance from the centriod to the datapoint in $x$
  if $x >$ MAE then
    send metrics to Algorithm 2
  else
    Wait for next metrics
  end if
```

The algorithm for detecting the global anomaly is shown in Algorithm 1, which requires the aggregated metrics received from the Monitor, as well as the MSA calculated after training the model. When the algorithm receives the metrics, it uses the trained k-means clustering algorithm to find the cluster where the metrics belong and then calculates the Euclidean distance from the centroid of that cluster. Once the Euclidean distance calculation is performed, it checks whether the distance is greater than that of the MAE threshold, then sends the metrics to the Algorithm 2 to find the local anomaly, otherwise it does nothing. For implementing the Global Anomaly detector, we have used Scikit of Python for their implementation of the k-means algorithm. Using this algorithm, we can detect whether there is an anomaly or not.

3.2.3 *Local anomaly*

Local anomaly detection involves the prediction of time series data. In our case, LSTM models have proven to be a highly suitable deep learning algorithm for this purpose. These models effectively capture sequential patterns and dependencies in time series data, making them ideal for identifying local anomalies in our framework. The LSTM model for our use case is shown in Equation (5). In this scenario, we want to predict the normal value on the basis of the features. The process of detecting the Local Anomaly is explained in Algorithm 2.

$$
\begin{aligned}
i^t &= \sigma(W^i x^t + \upsilon^i h^{(t-1)}) && \text{Input gate} \\
f^t &= \sigma(W^f x^t + \upsilon^f h^{(t-1)}) && \text{Forget gate} \\
o^t &= \sigma(W^o x^t + \upsilon^o h^{(t-1)}) && \text{Exposure gate} \\
\tilde{c}^t &= \mathrm{tahn}(W^c x^t + \upsilon^c h^{(t-1)}) && \text{New memory cell} \\
c^t &= f^t \circ \tilde{c}^{(t-1)} + i^t \circ \tilde{c}^t && \text{Final memory cell} \\
h^t &= o^t \circ \mathrm{adam}(c^t)
\end{aligned}
\tag{5}
$$

where, σ is the non-linearity function, x is the input, h is the hidden layer, W and υ are the weights, tahn and adam is the activation function.

Algorithm 2 Local detector

```
Require: Metrics, models, MAE, MSE, mahalanobis
  for all $x$ in Metrics do
    Predict the value of $M$ using the Metrics using the Model defined using Equation (5)
    if $x >$ M(MAE | MSE | Mahalanobis) then
      transfer the Metric name and Metric value to CP
    else
      do nothing
    end if
  end for
```

The algorithm requires all the LSTM models, collected metrics, and thresholds. When the data is received from the monitor, it uses all the models to predict each metrics. Then it uses the received value, the predicted value, and the threshold to decide whether an anomaly has occurred or not. If there is an anomaly, it sends those metrics to the control point for further processing. The implementation of the LSTM algorithm is accomplished using Keras of Python for their rich Deep Learning algorithms.

4. Experiments and evaluation

In this section, we will show the experiments carried out using our framework for anomaly detection, as well as the comparison with different algorithms.

4.1 *Testbed*

To test our proposed framework for automated monitoring and orchestration, we have used native IMS microservices in the cloud running in a Kubernetes cluster, which contains one master and three nodes. Details of the configuration are shown in Table 3.

Table 3. System specification

System name	CPU	RAM	Operating system	Kubernetes version
Master	8 Cores	8 GB	Ubuntu Desktop 18.04.4	1.18.2
Slave 1	4 Cores	4 GB	Ubuntu Core 18.04.1	1.17.0
Slave 2	4 Cores	4 GB	Ubuntu Core 18.04.1	1.17.0

4.2 *Dataset*

The dataset we have used for the experiments was provided by cloud-native implementation of IMS for a period of 4 months. Figure 3 is the normalized data of the metrics that were collected over the period and illustrated in one figure. The dataset also contains normal data and abnormal data. The green line in the figure illustrates the anomalies that occurred in the system. Dataset contains time series data from different sources of one IMS system instance running in the Kubernetes cluster. The metrics we have collected from the system are already shown in Table 2.

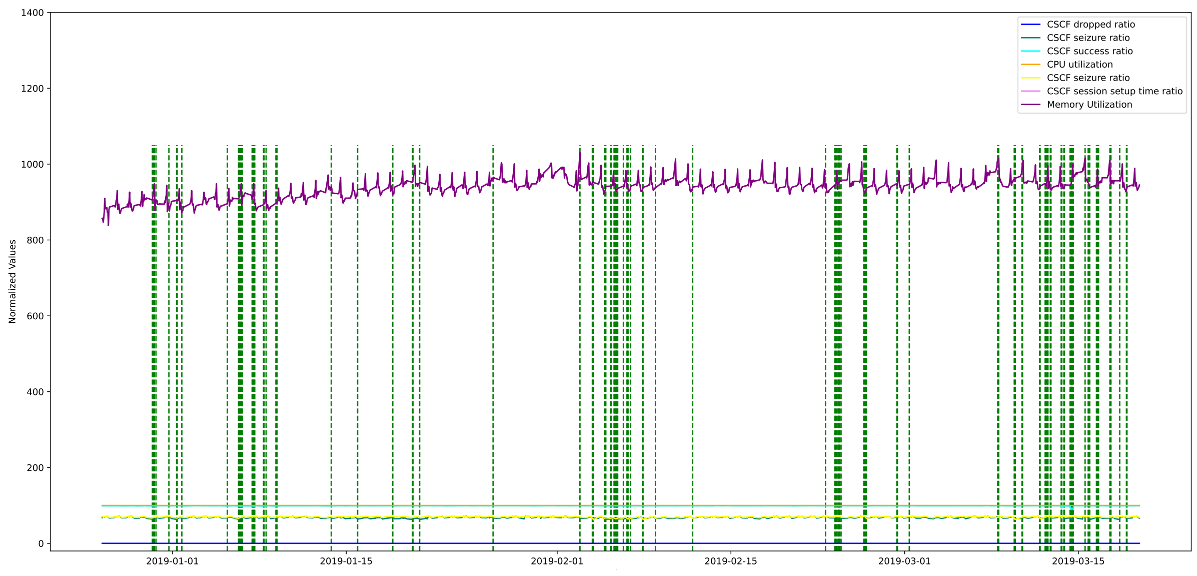


Figure 3. Dataset

4.3 *Evaluation*

In order to evaluate our algorithms, we have used the confusion matrix. So, we analyzed precision, recall, and accuracy. The details are provided in the following:

• Precision is the specificity of the proportion for abnormal behaviour that are correct detected. The equation is given by

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} \tag{6}$$

• Recall also known as sensitivity is the total number of abnormal behavior detected divided by the total number of abnormalities. The equation is given by:

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \tag{7}$$

• Accuracy is the F-Score which is the harmonic mean of precision and recall.The equation is given by:

$$\text{Accuracy} = 2 \times \frac{\text{Precision} \times \text{recall}}{\text{Precision} + \text{recall}} \tag{8}$$

4.4 *Global anomaly experiments*

Global anomaly detector detects abnormal behavior in the overall system using k-means clustering. For testing the algorithms, we have used the master to train the models and the slave to predict data traffic for anomaly detection. For this approach, we have used all the features/metrics of Table 2 to find out if the data at that time are anomalous. To train the model for k-means, we have trained the model with all the traffics for a period of one month. We have also performed other experiments using different algorithm parameters and feature extractions. The results are explained in Tables 4 and 5.

Tables 4 and 5 explore that expanding time into hours, days, months, and days of the year yields better results than using time. We have achieved better results when we used hours and days of the week to predict the anomalies as shown in experiments number 4 and 5. For finding the anomalies, we have used MSE and MAE, both have almost similar accuracy of 89% and 90%, respectively. Regarding the precision of the algorithm, the threshold using MAE has higher accuracy than MSE. From these experiments, we can conclude that the model with hour and day feature expansion and using MAE as the threshold have better results in terms of accuracy and resource utilization.

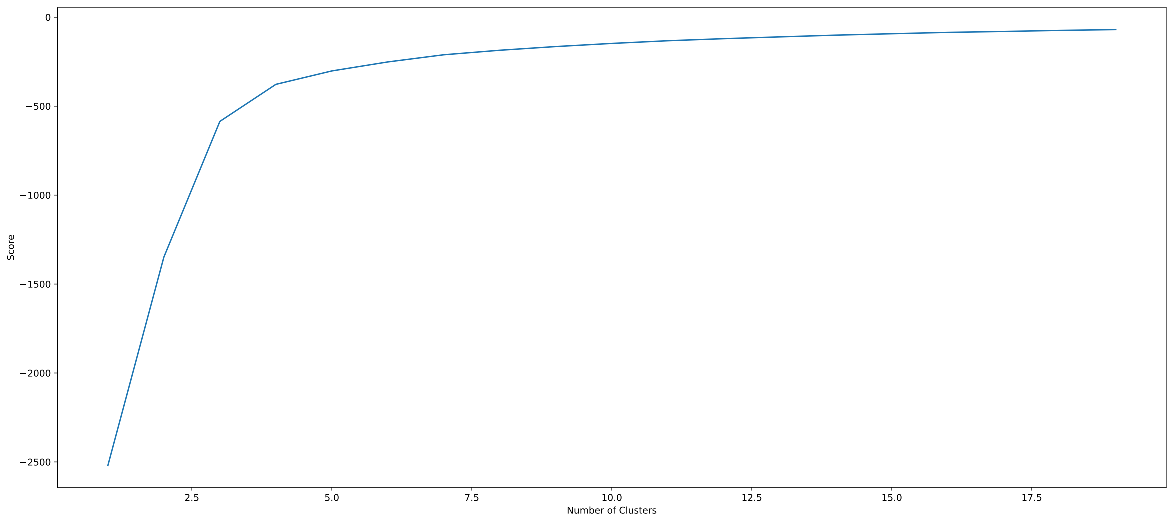
Table 4. Global anomaly model evaluation

Experiment number	Feature expansion	Threshold	Precision	Recall	Accuracy
1	None	MAE	0.4	0.6667	0.5
2	Hour, Day, Month, Day of the year	MSE	0.75157	0.62087	0.68
3	Hour, Day, Month, Day of the year	MAE	0.825457	0.87609	0.85
4	Hour, Day	MSE	0.852458	0.93100	0.89
5	Hour, Day	MAE	0.895252	0.90479	0.90

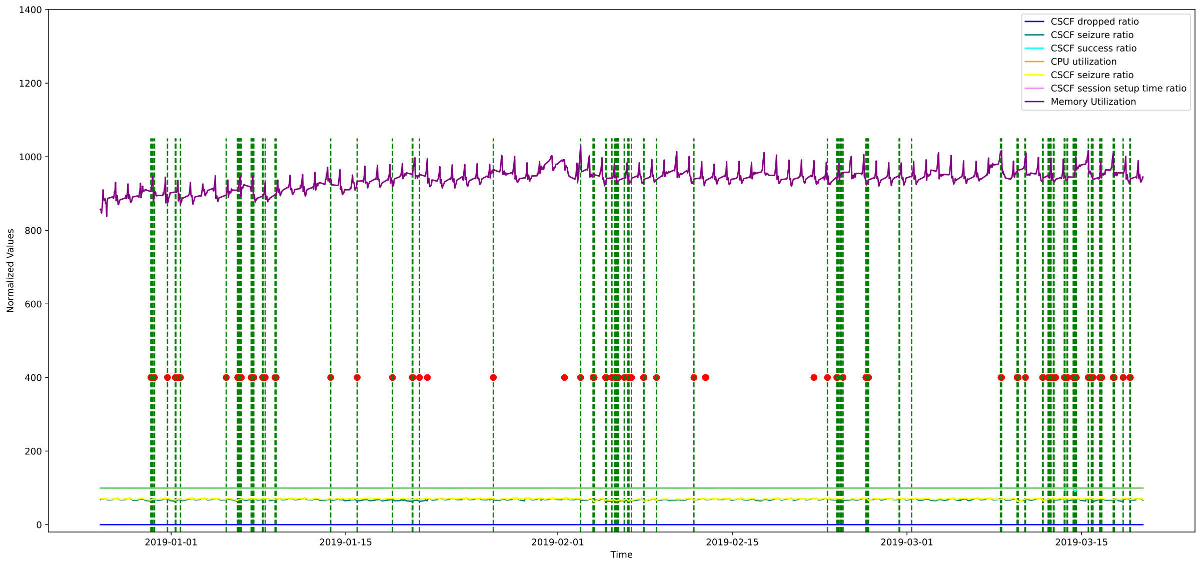
Table 5. Global anomaly resource evaluation

Experiment	Train CPU	Train memory	Train time	Predict CPU	Predict memory	Predict time
1	76	231,548	2.59254484	60	1,850	0.6578542
2	80	2,951,215	3.0457896	80	3,589	1.0245878
3	82	31,784,965	3.0078439	80	3,577	1.451258
4	82	251,148	2.920124587	70	2,540	0.785124248
5	80	250,148	2.710151434	70	1,952	0.613979578

Figure 4a shows the elbow curve of the training using the k-means algorithm using the feature expansion of the setup of experiment number 5 shown in Table 4. For the feature expansion, we have expanded the time into hour and day of the week. Using this feature expansion, we achieved the highest accuracy. Figure 4b shows the visualization of the algorithm to detect global anomalies for all metrics. In the figure, the vertical green dotted lines represent the actual anomalies that have occurred and the red dots represent the anomalies detected by our anomaly detection algorithm. The global anomaly detection algorithm can detect most anomalies in the system.



Elbow curve for the clustering algorithm



Prediction

Figure 4. Experimental result for global anomaly detection

To demonstrate the performance of various global anomaly detection algorithms, we conducted a comparative study using multiple approaches. Table 6 presents a performance comparison of these algorithms. Our experiments evaluated K-Means clustering, Isolation Forest, DBSCAN, Autoencoders, and SVM, with the table detailing their performance in terms of precision, recall, F1-score, training time, and prediction time.

Table 6. Global anomaly algorithm comparison

Algorithm	Precision	Recall	F1 score	Training time	Prediction time
K-Means clustering	0.8953	0.9048	0.9000	2.7102	0.6140
Isolation forrest	0.9528	0.8226	0.8829	1.8490	0.5250
DBSCAN	0.9466	0.8448	0.8928	5.2746	1.7577
Autoencoders	0.5221	0.5927	0.5552	873.4710	0.0042
SVM	0.7829	0.8536	0.8167	712.1811	0.8273

4.5 *Local anomaly experimentation*

The local anomaly detects the abnormal behavior of each of the components, including the pod, node, and application modules individually. For testing the algorithms, we have used the master to train the models and the slave to predict data traffic for anomaly detection. The local anomaly detector uses LSTM algorithms on all metrics to find exactly where the abnormality occurred. To train the LSTM models for all metrics, we have utilized traffic that does not contain any abnormality for a period of one month. We have also performed experiments using different other algorithm parameter and feature extractions. The analysis of these experiments is explained in Table 7 for the evaluation of the models. Table 8 shows the utilization of the model resources in the Kubernetes cluster.

Table 7. Local anomaly model evaluation

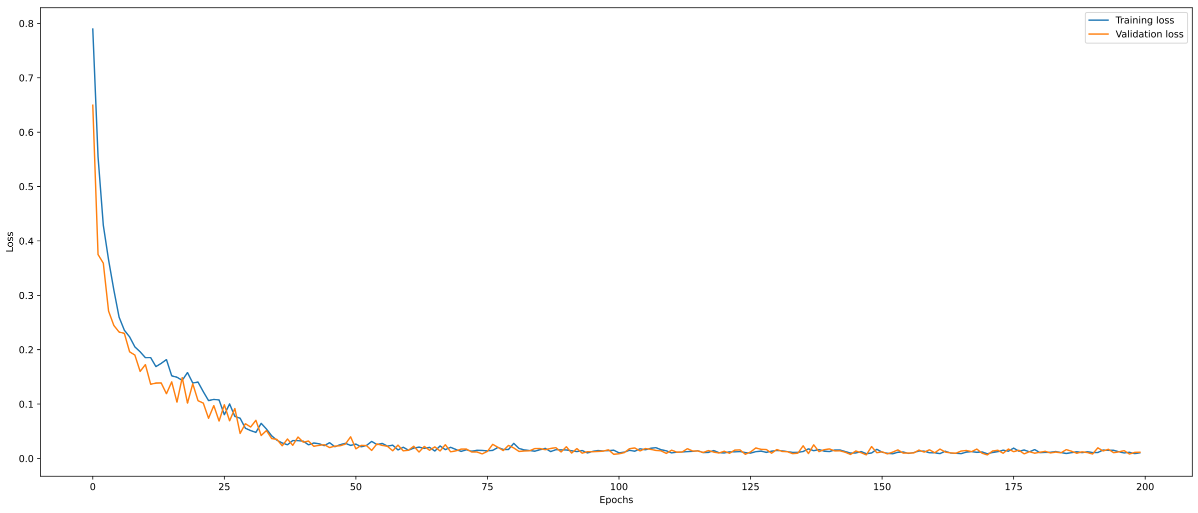
Metric	Model (layers, hidden layers)	Learning window hours	Feature expansion	Epochs	Threshold	Precision	Recall	Accuracy
CSCF dropped ratio	LSTM (3,128)	30	None	25	MAE	0.4458	0.56920	0.5
CSCF dropped ratio	LSTM (4,128)	60	None	100	MAE	0.57	0.69175	0.625
CSCF dropped ratio	VAE (4,512) LSTM (4,256)	48	None	8 20	MAE	0.58	0.67757	0.625
CSCF dropped ratio	LSTM (3,256)	400	Hour, Day, Month, Year	200	MAE	0.5555	0.45458	0.5
CSCF dropped ratio	LSTM (2,128)	450	Hour, Day	200	Mahalanobis' distance	0.8542	0.95099	0.9
CSCF seizure ratio	LSTM (3,128)	450	Hour, Day	200	MAE	0.8789	0.89688	0.8878
CSCF success ratio	LSTM (2,256)	450	Hour, Day	200	Mahalanobis' distance	0.8841	0.90065	0.8923
CSCF session ratio	LSTM (2,256)	450	Hour, Day	200	Mahalanobis' distance	0.8945	0.89871	0.8966
CSCF session setup time ratio	LSTM (2,128)	450	Hour, Day	200	MAE	0.8925	0.89772	0.8951
Pod CPU	LSTM (3,512)	450	Hour, Day	200	MAE	0.8858	0.94107	0.9126
Pod memory	LSTM (2,64)	450	Hour, Day	200	Mahalanobis' distance	0.8764	0.91692	0.8962
Node CPU	LSTM (2,64)	450	Hour, Day	200	MAE	0.8977	0.86587	0.8815
Node memory	LSTM (2,64)	450	Hour, Day	200	Mahalanobis' distance	0.8998	0.85324	0.8759

Table 8. Local anomaly resource evaluation

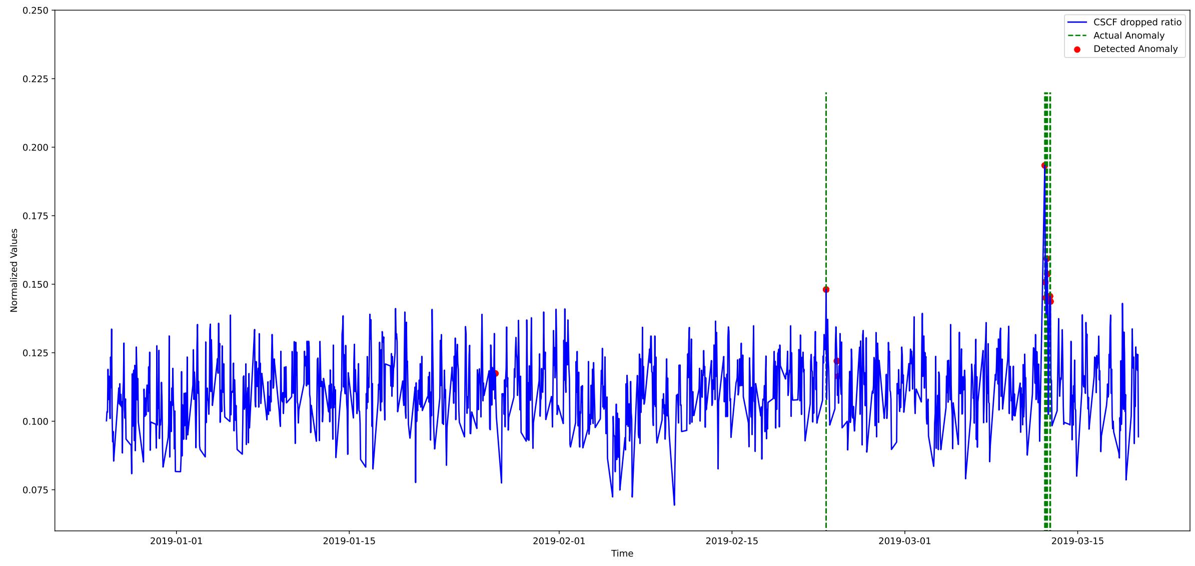
Metric	Train CPU	Predict CPU	Train memory	Predict memory	Train time	Predict time
CSCF dropped ratio	50	30	1,165,504	3,457,916	572	2
CSCF seizure ratio	56	36	1,511,606	11,897,355	419	3
CSCF success ratio	60	40	1,658,451	12,589,785	458	3
CSCF session ratio	60	40	1,726,878	13,574,741	423	3
Pod CPU	55	45	1,824,578	14,578,978	450	3
Pod memory	50	38	1,024,567	5,874,216	380	2
Node CPU	65	35	1,064,781	6,048,787	410	4
Node memory	70	40	1,124,678	5,547,821	430	2

Table 7 illustrates that feature expansion yields better accuracy than without feature extraction. Moreover, using specific feature expansions of time has the highest accuracy than using all the feature expansions. Experiments 1 to 5 are done for the CSCF dropped ration metric. They explore how feature expansion, model design, and threshold type have an impact on the performance of anomaly detection. From these experiments, we have shown that we are able to reach the accuracy of above 89% by changing the parameters of the model, feature expansion, and threshold. Table 8 contains the utilization of training resources and the prediction of anomaly detection. For the prediction of the anomalies, it takes around 2 ms and the training takes around 400 to 500 s. The utilization of resources for training and prediction is around 50% of the CPU and the memory is around 1,500,000 KB.

Figure 5a shows the learning curve of one of the LSTM models for the CSCF drop ratio using the expansion of features for time, which is the hour and day of the week. Figure 5b shows the performance of the LSTM algorithm in detecting anomalies in the CSCF dropped ratio.



Model train and validation loss



Prediction

Figure 5. Experimental result for local anomaly detection

In order to demonstrate the performance of various local anomaly detection algorithms, we conducted a comparative study using multiple algorithms.Table 9 presents a comparative analysis of these algorithms. Our experimental results for LSTM, Autoencoder, VAE LSTM, and GAN highlight their effectiveness in detecting anomalies across different metrics. The comparison includes a detailed breakdown of precision, recall, F1 score, training time, and prediction time for each algorithm.

Table 9. Local anomaly algorithm evaluation

Metric	Algorithm	Precision	Recall	F1 score	Training time	Prediction time
CSCF dropped ratio	LSTM	0.8542	0.9510	0.9000	572	0.0020
	Autoencoders	0.7168	0.7146	0.7157	637	0.0038
	VAE LSTM	0.5800	0.6776	0.6250	593	0.0033
	GAN	0.824	0.9849	0.8973	418	0.0037
CSCF seizure ratio	LSTM	0.8789	0.8969	0.8878	419	0.0030
	Autoencoders	0.8484	0.8216	0.8348	524	0.0024
	VAE LSTM	0.8362	0.8661	0.8509	556	0.0026
	GAN	0.8548	0.9202	0.8863	659	0.0033
CSCF success ratio	LSTM	0.8945	0.8987	0.8966	458	0.0030
	Autoencoders	0.7975	0.7498	0.7729	498	0.0026
	VAE LSTM	0.8891	0.8949	0.8920	638	0.0037
	GAN	0.8587	0.8814	0.8699	487	0.0024
CSCF session ratio	LSTM	0.8925	0.8977	0.8951	423	0.0030
	Autoencoders	0.8858	0.8317	0.8579	515	0.0037
	VAE LSTM	0.8596	0.8879	0.8735	495	0.0031
	GAN	0.8309	0.8534	0.8420	432	0.0029
Pod CPU	LSTM	0.8858	0.9411	0.9126	450	0.0030
	Autoencoders	0.8708	0.9026	0.8864	564	0.0035
	VAE LSTM	0.8304	0.8535	0.8418	572	0.0023
	GAN	0.8437	0.9771	0.9055	558	0.0035
Pod memory	LSTM	0.8764	0.9169	0.8962	380	0.0020
	Autoencoders	0.8057	0.9950	0.8904	626	0.0024
	VAE LSTM	0.8794	0.9025	0.8908	616	0.0025
	GAN	0.8571	0.8171	0.8366	699	0.0027
Node CPU	LSTM	0.8977	0.8659	0.8815	410	0.0040
	Autoencoders	0.8769	0.8713	0.8741	552	0.0049
	VAE LSTM	0.8485	0.8384	0.8434	684	0.0035
	GAN	0.8210	0.8095	0.8152	466	0.0049
Node memory	LSTM	0.8998	0.8532	0.8759	430	0.0020
	Autoencoders	0.8653	0.8798	0.8725	635	0.0025
	VAE LSTM	0.8085	0.8267	0.8175	545	0.0021
	GAN	0.8496	0.9094	0.8785	549	0.0024

5. Future direction and conclusions

In this paper, we have introduced a comprehensive framework for detecting anomalies in a native IMS application running within a Kubernetes cloud environment. Our framework performs proactive anomaly detection, employing both k-means clustering and LSTM models to identify global and local anomalies. Through extensive experimentation, we demonstrated that our proposed scheme achieves strong results while optimizing resource usage for anomaly detection. The ability of the framework to detect anomalies with approximately 90% precision ensures robust security and reliability for next-generation telecom networks. By leveraging advanced machine learning techniques, our solution anticipates system behavior and identifies abnormalities in the IMS core, enhancing the security of critical infrastructures as the telecommunications sector evolves from 5G to 6G.

In terms of limitations, we acknowledge that our current model uses static thresholds and is trained on a private dataset, which may limit generalizability. To address these concerns, we outline the following actionable next steps for future work:

- **Dataset validation:** Evaluate the framework on publicly available or synthetic datasets in different domains (e.g., Yahoo Webscope, NAB) to assess generalizability.
- **Model adaptability:** Implement online learning and periodic model retraining to adapt to evolving system behaviors and concept drift.
- **Explainability enhancements:** Incorporate explainability methods such as SHAP (SHapley Additive Explanations) and attention-based visualization to increase model transparency and facilitate debugging in real-world deployments.
- **Temporal resolution:** Apply sliding window techniques to improve the granularity of time-series anomaly detection and capture transient issues more effectively.
- **Federated learning:** Deploy federated learning strategies to allow collaborative anomaly detection across decentralized cloud-native infrastructures while preserving data privacy.

• **Automated response:** Develop a more proactive orchestrator capable of autonomously responding to detected anomalies (e.g., restarting specific pods, nodes, or applications based on severity and SLA policy).

For practical application and implementation barriers, our framework is designed to integrate with existing telecom infrastructures that leverage cloud-native technologies such as Kubernetes and monitoring tools like Prometheus. This makes deployment feasible without requiring substantial architectural changes. In practice, a telecom operator can use the system to continuously monitor the telemetries of the IMS service and automatically mitigate detected issues through more advanced orchestrated responses based on the SLA.

However, several practical barriers must be considered. First, data heterogeneity across different deployments may affect model generalizability. Second, operational constraints such as CPU, memory, and model latency must be carefully managed to avoid interfering with core services. Finally, SLA complexities could restrict certain automated actions (e.g., pod restarts) unless validated against business policies. These factors must be addressed through further testing, adaptive configurations, and policy-aware orchestration mechanisms in production environments.

These enhancements will further solidify the robustness, adaptability, and practical relevance of our anomaly detection framework in modern telecom infrastructures.

Conflict of interest

The authors declare that they have no competing interests.

References

[1] "Ericsson mobility report 2022," [Online]. Available: https://www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf. [Accessed Dec. 20, 2022]

[2] "Virtualization and containerization of the mobile network," [Online]. Available: https://www.metaswitch.com/knowledge-center/white-papers/virtualization-and-containerization-of-the-mobile-network. [Accessed Dec. 20, 2022]

[3] "IP multimedia subsystem," [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2176. [Accessed Dec. 20, 2022]

[4] K. K. Tam, H. L. Goh, "Sip: session initiation protocol," in Proc. 2002 IEEE International Conference on Industrial Technology, 2002. IEEE ICIT '02., Dec. 11-14, 2002, Bankok, Thailand, 2002.

[5] L. Mike and S. Steve, "Microservices adoption in 2020," [Online]. Available: https://www.oreilly.com/radar/microservices-adoption-in-2020/. [Accessed Dec. 20, 2022]

[6] F. Lu, H. Pan, X. Lei, X. Liao, and H. Jin, "A virtualization-based cloud infrastructure for ims core network," in Proc. 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), Xi'an, China, Dec. 2-5, 2013, pp. 25-32.

[7] E. Casalicchio, *Container Orchestration: A Survey*. Cham, Switzerland: Springer International Publishing; 2019, pp. 221-235.

[8] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Communications Surveys & Tutorials*, vol. 28, no. 6, pp. 18-26, 2014.

[9] D. Geneiatakis, T. Dagiuklas, G. Kambourakis, C. Lambrinoudakis, S. Gritzalis, K. S. Ehlert, and D. Sisalem, "Survey of security vulnerabilities in session initiation protocol," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 3, pp. 68-81, 2006.

[10] D. Sisalem, J. Kuthan, and S. Ehlert, "Denial of service attacks targeting a sip voip infrastructure: attack scenarios and prevention mechanisms," *IEEE Network*, vol. 20, no. 5, pp. 26-31, 2006.

[11] S. Ehlert, D. Geneiatakis, and T. Magedanz, "Survey of network security systems to counter sip-based denial-of-service attacks," *Computers & Security*, vol. 29, no. 2, pp. 225-243, 2010.

[12] A. Boubendir, E. Bertin, and N. Simoni, "A vnf-as-a-service design through micro-services disassembling the ims," in Proc. 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France, Mar. 7-9, 2017, pp. 203-210.

[13] W.-K. Chiang and J.-W. Wen, "Design and experiment of nfv-based virtualized ip multimedia subsystem," in Proc. 2018 3rd International Conference on Computer and Communication Systems (ICCCS), Nagoya, Japan, Apr. 27-30, 2018, pp. 397-401.

[14] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," *arXiv*, 2022, arXiv:2202.07125.

[15] J. G. Almaraz-Rivera, "An anomaly-based detection system for monitoring kubernetes infrastructures," *IEEE Latin America Transactions*, vol. 21, no. 3, pp. 457-465, 2023.

[16] M. Abu-Lebdeh, J. Sahoo, R. Glitho, and C. W. Tchouati, "Cloudifying the 3gpp ip multimedia subsystem for 4g and beyond: A survey," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 91-97, 2016.

[17] Y. Seraoui, B. Raouyane, M. Belmekki, and M. Bellafkih, "Opening into a virtualized ims based telecom architectural framework: Telecom cloud manager," in Proc. 2017 International Conference on Wireless Networks and Mobile Communications (WINCOM), Rabat, Morocco, Nov. 1-4, 2017, pp. 1-6.

[18] H. B. Lee, S. I. Kim, and H. S. Kim, "A fault management system for nfv," in Proc. 2018 International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, Jan. 10-12, 2018, pp. 640-645.

[19] M. T. Raza, H.-y. Tseng, C. Li, and S. Lu, "Modular redundancy for cloud based ims robustness," in Proc. 15th ACM International Symposium on Mobility Management and Wireless Access, Miami, FL, USA, Nov. 21-25, 2017, pp. 75-82.

[20] M. T. Raza and S. Lu, "Enabling low latency and high reliability for ims-nfv," in Proc. 2017 13th International Conference on Network and Service Management (CNSM), Tokyo, Japan, Nov. 26-30, 2017, pp. 1-9.

[21] M. T. Raza, S. Lu, M. Gerla, and X. Li, "Refactoring network functions modules to reduce latencies and improve fault tolerance in nfv," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2201-2214, 2018.

[22] S. Thudumu, P. Branch, J. Jin, and J. J. Singh, "A comprehensive survey of anomaly detection techniques for high dimensional big data," *Journal of Big Data*, vol. 7, no. 1, pp. 1-30, 2020.

[23] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS ONE*, vol. 11, no. 4, p. e0152173, 2016.

[24] Z. Zou, Y. Xie, K. Huang, G. Xu, D. Feng, and D. Long, "A docker container anomaly monitoring system based on optimized isolation forest," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 123-135, 2019.

[25] T. Amarbayasgalan, V. H. Pham, N. Theera-Umpon, and K. H. Ryu, "Unsupervised anomaly detection approach for time-series in multi-domains using deep reconstruction error," *Symmetry*, vol. 12, no. 8, p. 1251, 2020.

[26] S. Lin, R. Clark, R. Birke, S. Schönborn, N. Trigoni, and S. Roberts, "Anomaly detection for time series using vae-lstm hybrid model," in Proc. ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, May 4-8, 2020, pp. 4322-4326.

[27] O. Mart, C. Negru, F. Pop, and A. Castiglione, "Observability in kubernetes cluster: Automatic anomalies detection using prometheus," in Proc. 2020 IEEE 22nd International Conference on High Performance Computing and Communications, Sydney, Australia, Dec. 14-16, 2020, pp. 565-570.

[28] A. Aly, M. Fayez, M. Al-Qutt, and A. M. Hamad, "Multi-class threat detection using neural network and machine learning approaches in kubernetes environments," in Proc. 2024 6th International Conference on Computing and Informatics (ICCI), Cairo, Egypt, Jan. 7-9, 2024, pp. 103-108.

[29] M. Azrour, J. Mabrouki, Y. Farhaoui, and A. Guezzaz, "Experimental evaluation of proposed algorithm for identifying abnormal messages in sip network," in Proc. Intelligent Systems in Big Data, Semantic Web and Machine Learning, Marrakech, Morocco, Feb. 15-17, 2021, pp. 1-10.

[30] D. Pereira, R. Oliveira, and H. S. Kim, "Abnormal signaling sip dialogs detection based on deep learning," in Proc. 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland, Apr. 25-28, 2021, pp. 1-6.

[31] Z. Khaleel, A. Lakizadeh, "Classification of abnormal signaling sip dialogs through deep learning," *IEEE Access*, vol. 9, pp. 165557-165567, 2021. https://doi.org/10.1109/ACCESS.2021.3134567

[32] F. Cecchinato, L. Vangelista, G. Biondo, and M. Franchin, "Anomaly detection using lstm neural networks: an application to voip traffic," in Proc. 2021 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE), Shanghai, China, Oct. 12-14, 2021, pp. 1-7.

[33] D. Pereira and R. Oliveira, "Detection of abnormal sip signaling patterns: A deep learning comparison," *Computers*, vol. 11, no. 2, p. 27, Feb. 2022. https://doi.org/10.3390/computers11020027

[34] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451-461, 2003. https://doi.org/10.1016/S0031-3203(02)00060-2

[35] K. P. Sinaga and M.-S. Yang, “Unsupervised k-means clustering algorithm,” *IEEE Access*, vol. 8, pp. 80716-80727, 2020. https://doi.org/10.1109/ACCESS.2020.2988796