A HIERARCHICAL ANOMALY DETECTION MODEL FOR CLOUD COMPUTING SYSTEM BASED ON OPENSTACK

Mbasa Joaquim Molo[1,2,3,5], Abednego Wamuhindo Kambale[5,6], Joke Badejo[1,2], Emmanuel Adetiba[1,2,4], Victoria Oguntosin[1,2], Mughole Daniella Kalimumbalo[1,2,3], Baraka Mushage Olivier[3], Victor Akande[1,2], Nzanzu Patrick Vingi[1,2,3], Claude Takenga[3,5]

[1]Department of Electrical and Information Engineering, Covenant University, Ota, Ogun State, Nigeria

[2]Covenant Applied Informatics and Communication African Center of Excellence, Covenant University, Ota, Ogun State, Nigeria

[3]Génie Electrique et Informatique, Université Libre des Pays des Grands Lacs, BP 360 Goma, Goma,Democratic Republic of the Congo

[4]HRA, Institute for Systems Science, Durban University of Technology, P.O. Box 1334, Durban, South Africa

[5]Infokom GmbH, Entreprise NTIC, Neubrandenburg, Germany

[6]Department Of Electronics, Information And Bioengineering, Politecnico di Milano, Italy

Abstract

The evolution of processing and storage technologies and the success of the Internet have emerged with cloud computing as a new computing model that aims to provide users with reliable, flexible, and QoS-assured computing resources. Cloud computing includes delivering applications and hardware as services over the Internet, and systems software in data centres support those services. To meet the different objectives of large companies, the key criteria of cloud computing are versatility, scalability, manageability, and expandability. However, cloud infrastructure is subject to runtime failure due to its complexity and unimaginable growth of users demand. To ensure that the system performs as expected, it is critical to identify abnormal occurrences that disrupt system performance. Several works perform anomaly detection with logfile datasets that provide a limited number of occurrences preventing machine learning models from learning from the data deeply. In this work, a cloud computing environment based on OpenStack is configured to generate valuable logfiles containing suitable data for anomaly detection that uses the standard logging levels used by Openstack. This paper also proposes a hierarchical anomaly detection model that breaks down the classification problem of anomalies into subproblems to explicitly determine the component source of failure in the cloud computing environment based on OpenStack. In this paper, three different models, including the Support Vector (SVM), Feedforward Neural Network (FFNN) and the Long-Short Term Memory (LSTM), are also implemented and compared in order to leverage the most performant. The best model at each hierarchy level is selected based on its precision, recall and f-measure metrics. The results show that for the first level of the hierarchy, the LSTM is the best model. LSTM and FFNN both performed better for the second level, and for the last level, the FFNN outperformed the LSTM. The overall performance of the hierarchical model is expressed using a set of data dedicated for testing in terms of hierarchical precision, hierarchical recall and hierarchical F-measure. Hence, the hierarchical anomaly detection model can be considered as a critical tool used to initiate the disaster management of cloud computing systems since it provides deep insight into the investigation of cloud-related failure.

Keywords: cloud computing, OpenStack, anomaly detection, Hierarchical classification

1. Introduction

For decades, digitalisation has transformed the perception of humanity regarding how real-world phenomena can be managed. The evolution of technology with the rapid development of electronic devices such as computers and smartphones and the speed at which the Internet of Things (IoT) is evolving today is changing tremendously how today's world functions. Cloud computing solutions are essential for satisfying the fast-increasing technological demand. With cloud computing, resources are delivered as services through the Internet. With many advantages such as high availability, cost-effectiveness, elasticity, educational institutions, government, and businesses are integrating cloud computing as a requirement to find a solution to storage, maintenance and energy consumption related issues (Islam & Miranskyy, 2020).

Cloud solutions are now available from all major cloud service providers (CSP), including IBM Blue, Google Apps, Microsoft Azure, and Cloud Amazon EC2. However, specifications such as functionalities in terms of hardware and software are offered in diverse ways, and Service Level Agreement (SLA) between CSP and users in some cases can be the source of misunderstanding. It may also lead to dissatisfaction of user's requests. For this purpose, cloud opensource management platforms help to offer more flexibility to businesses depending on users' requirements in the delivery of services (Sefraoui et al., 2015).

Cloud management platforms (CMPs) are software and tools that assist in creating and managing cloud systems. They ensure that cloud resources function efficiently and interact with end-users and other applications in an effective manner. Many open-source CMPs have been developed, including OpenStack, OpenNebula, CloudStack, Eucalyptus etc. (Freet et al., 2016a; Marozzo, 2019; Mullerikkal & Sastri, 2015; Pyati et al., 2020). Table 1 briefly summarises the comparison of different CMPs in terms of the number of participants on forums per month and the number of supporters.

Table 1: Comparison of CMPs in terms of the number of participants on forums per month and number of supporters.

C/S	Cloud platform	Number of participants on forums/ month	Supporters	References
1.	OpenStack	800	More than 200	(Pyati et al., 2020) (Mullerikkal & Sastri, 2015)
2.	CloudStack	200	Microsoft, Yahoo, Facebook, Citrix and Google etc.	(Freet et al., 2016b)
3.	Eucalyptus and OpenNebula	100	Dell, Intel, Ubuntu. None for OpenNebula	(Ismaeel & Miri, 2015) (Marozzo, 2019)

Cloud computing environment has an excellent processing capability and runs and manages much work and applications simultaneously. Therefore, a cloud environment is exposed to runtime failure (Jie et al., 2020). Detecting abnormal events that affect the system performance is critical to ensure that the system works as expected and the SLA is not violated.

Because of their complexity, cloud systems can present a wide range of runtime failures, although not all the system's failures are necessarily known by the system administrators (Molo, Badejo, et al., 2021). Therefore, Identifying the system's cause of failure is a complicated task. It is critical to determine whether the system can detect failures, as this is required to initiate recovery. Preferably, the system would detect a failure and explicitly state the source of the issue, allowing system administrators to identify the problem (Cotroneo et al., 2018).

According to the statistics released by IBM, a considerable amount of data is generated every day and can reach up to 2.5 trillion bytes. The source data includes sensors, weather information, social network, log files, etc. (Azizi et al., 2019). In a cloud environment, all the performed operations either implicitly by the different cloud computing components or by the system administrator and the users are chronologically stored in logfiles. Therefore, logfiles are valuable source of information that can help investigate the state of the cloud computing system (Teixeira et al., 2018).

Different approaches to anomaly detection, including Statistical and machine learning, have been developed. First, the statistical approach assumes that normal events as part of the area of a stochastic model have a high probability while anomalies possess a low probability. On the other hand, the machine learning approach targets a high true positive rate while detecting anomalies and keeping a low true negative rate (Aissa & Guerroumi, 2016; N. Ghosh et al., 2019).

This paper uses the machine learning approach to detect anomalies in a cloud computing environment based on OpenStack. The machine learning model uses log file data to perform the detection of anomalies. Although several works have been done in this regard, most of the proposed solutions are based on flat classification, including binary, multi-label, and multi-class classification. This paper proposes a hierarchical anomaly detection model that breaks down the classification problem of anomalies into subproblems to explicitly determine the component source of failure in the cloud computing environment.

The hierarchical classification can be denoted as a structured multi-label classification in which the hierarchical structure can take the form of a tree or a Direct Acyclic Graph (DAG). Hierarchy is considered a tree if each node belongs to a parent node, whereas in a DAG, there are nodes that belong to a least two parent nodes (Daisey & Brown, 2020).

Moreover, several methods for hierarchical classification can be considered; the following are the most common:

(i) Local Classifier per Node (LCN): this method considers a binary classification for each node of the hierarchy apart from the root node (Kiritchenko et al., 2006).
(ii) Local classifier per Parent Node: (LCPN): this approach considers a multi-class classification for every node possessing sub-nodes. The classification, in this case, is made to predict all the sub-classes of the parent node (Silla & Freitas, 2011).

(iii) Local Classifier per Level (LCL): A multi-level classifier is considered for every level of the hierarchy. However, this method requires consistent prediction to avoid any inconsistency in the determination of the right path (Serrano-Pérez & Sucar, 2021).

Figure 1 a, b, and c presents the hierarchical structure of the different approaches:

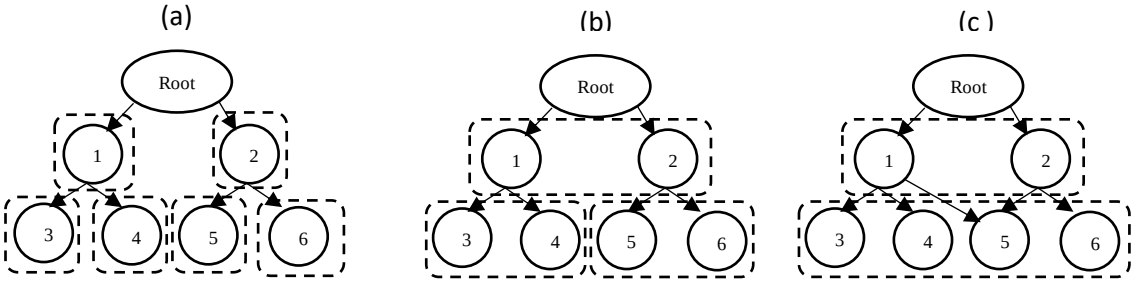

Figure 1: Different classification approaches used for hierarchical classification where (a) is the LCN, (b) is the LCPN and (c) is the LCL

The hierarchical anomaly detection solution is organised as follows:

1. Configuration of an OpenStack based cloud computing environment (FEDGEN)
2. Extraction and Pre-processing of logfile from different OpenStack components of the FEDGEN testbed
3. Training and testing of the hierarchical anomaly detection model

There have been tremendous contributions in anomaly detection for cloud computing systems. The following works give a brief overview of different anomaly detection models developed with logfiles datasets:

(Gulenko et al., 2016) evaluated the performance of several machine learning models to perform anomaly detection. In the course of their experiment, all the models used provided an average of 92% in terms of accuracy and recall. Also, the average F1 score is estimated to be 62%. (Pattarakavin & Chongstitvatana, 2016) used the support vector machine (SVM), the K nearest neighbour (KNN), Decision Tree (DT) and Naïve Bayes to perform anomaly detection for hard disk manufacturing machines. The dataset used to carry out the experiment was collected from log files, and performance metrics provided 100% in terms of accuracy, recall and F-measures. (Sauvanaud et al., 2018) proposed a machine learning model capable of identifying anomalies on a virtual machine. The proposed model provided 90% accuracy as a performance metric. (Lu et al., 2018) proposed an anomaly detection model using a convolutional neural network on a Hadoop Distributed File System (HDFS) logfile. The approach used provided 97.7%, 99.3%, 98.5%, respectively, for precision, recall and F-measures. These results were compared to other models based on Multi-Layer Perception (MLP) and Long-Short Term Memory (LSTM). (Yuan et al., 2019) presented an anomaly detection model that uses a flat classification technique to determine the component source of failure. This paper used natural language processing to investigate anomalies in cloud computing environments based on OpenStack. The performance metrics gave 97% and 95%, respectively, in terms of precision and recall. (Patil et al., 2019) proposed a Short-Term Long Memory - Layerwise Relevance Propagation (LSTM-LRP) deep learning algorithm to detect anomalies for HDFS log files. The primary objective of this work was to build a model that would provide realistic results to build trust and make the model commendable for real-world

applications. The proposed approach provided a precision of 96,6%, recall of 99.3% and an F1-score of 97.7% as performance metrics. (Zhou et al., 2020) proposed LogSayer as a log pattern-driven model for anomaly detection. This approach uses LSTM to take advantage of the historical correlation of log patterns to perform adaptive anomaly detection. The dataset used for the experiment was an OpenStack based dataset. The proposed approach provided 98% of accuracy. (Akanle et al., 2020) used the SVM to provide a binary classifier that detects manually injected fault in an OpenStack logfile. The model provided 100% accuracy, precision and f1-score. (Farzad & Gulliver, 2020) proposed an anomaly detection model based on isolation forest and deep autoencoder. The proposed approach used deep autoencoder for feature extraction in OpenStack logfiles, and isolation forest was used for anomaly detection. The model provided 67.1%, 99.5% and 80.2%, respectively, for precision, recall, and F1-score as performance metrics. (Cotroneo et al., 2021) took advantage of an unsupervised deep learning algorithm that makes use of an autoencoder to optimise data dimensionality and inter-cluster variance to perform anomaly detection on an OpenStack logfile. The proposed approach provided 97% purity. (M. El-Shamy et al., 2021) Proposed a Software Defined Network (SDN) monitoring algorithm to detect performance anomalies and the cause of the bottlenecks in a cloud computing datacentre. In this paper, the model was trained with the SVM, and the model provided 95.54% accuracy. The dataset was collected from simulation due to the lack of a physical cloud testbed.

As a limitation, in the majority of the works reviewed, the different models developed were limited either to binary classifications by determining whether an event is normal or not. Other works were only able to perform flat classification to determine all possible events classes at once. Since cloud environments are heterogeneous and complex systems, cloud computing system administrators may find the task challenging in identifying the component source of failure as one error can affect one or more parts of the infrastructure. Therefore, this paper provides a mean to arrange different runtime failures hierarchically. The hierarchical anomaly detection model is composed of three levels. The first level separates normal and abnormal events. The second level states the level of severity of the event in case the event is abnormal, and the last level precise the component source of system failure.

Apart from the introduction, the rest of this paper is organised into four parts. The second section of this paper will present the FEDGEN testbed's configuration steps required to build a cloud environment that helps to collect log files datasets. The third section will provide an insight into the hierarchical anomaly detection model, the fourth section will present the results, and the last section is the conclusion.

2. FEDGEN testbed Architecture

This section presents the material and methods used to configure the FEDGEN testbed based on OpenStack in order to create a dataset containing log information suitable for hierarchical classification.

The FEDGEN testbed configuration is achieved with the OpenStack cloud operating system. The design is suitable for small, medium, and large-scale cloud deployments. The computational capability can be enhanced by adding a node to the infrastructure. Figure 2 presents the architecture of the FEDGEN testbed with its different components.

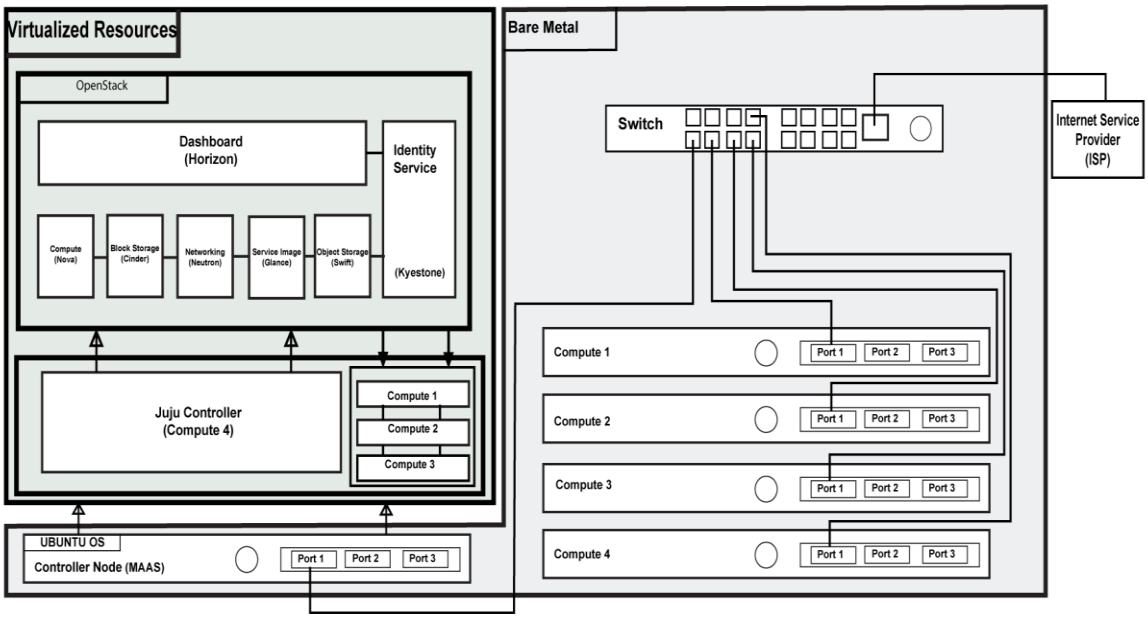

Figure 2: Configuration architecture of the FEDGEN testbed illustrating required hardware components and their physical networking connection.

The first step of the configuration consists of installing the Metal as a Service (MASS) on top of an UBUNTU server operating system. This helps virtualise physical resources. The second step is to deploy Juju on top of the MAAS to manage and orchestrate the different OpenStack cloud applications. The final step is the configuration of the OpenStack cloud application to have a fully functioning cloud environment.

2.1.OpenStack

OpenStack is a cloud management platform that manages and provisions vast pools of processing, storage, and networking resources across a data centre via APIs and standard authentication procedures (Musavi et al., 2016). OpenStack provides well-defined APIs for accessing its many applications. The network service (neutron), storage service, and compute service (Nova) are the three core services that make up its architecture. The neutron component handles the networking. The Nova represents the computational domain and manages network policies, load balancing, DHCP and DNS, among other applications. It is in charge of hosting and maintaining virtual instances and controlling the virtual machine life cycle. Lastly, Swift and Cinder storage are in charge of storage management (Pyati et al., 2020). Figure 3 presents the architecture of OpenStack.

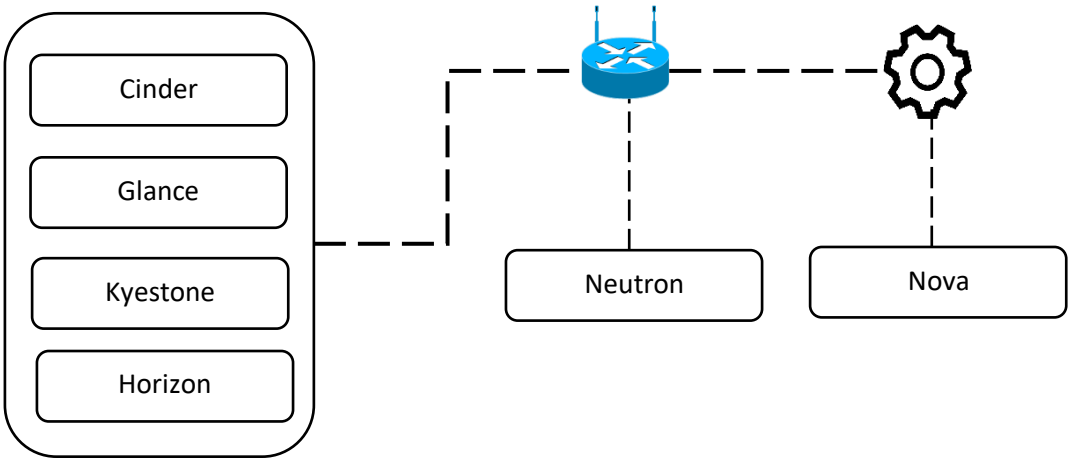

Figure 3: OpenStack Architecture showing its primary components where the neutron is used for networking and Nova for computing

2.2. MAAS

MAAS is an open-source data centre construction program. It converts bare-metal servers into virtualised machines instances, eliminating the need to manage them as separate units. It allows to manage, provision, and delete machines as instances in a public cloud such as Google GCE or Microsoft Azure. All nodes in the FEDGEN cloud system use the MAAS controller as Dynamic Host Configuration Protocol (DHCP) server and a Domain Name System (DNS). Automation, networking and hardware testing are some of the other benefits and characteristics of the MAAS, where automation enables the automated discovery of all network devices (Libri et al., 2021).

The image in Figure 4 presents the fully installed MAAS. The Figure shows the four different computing nodes provisioned with their specifications. The Figure also shows the juju controller deployed on the MASS. All the machines have UBUNTU bionic 18.04 LTS as the operating system.

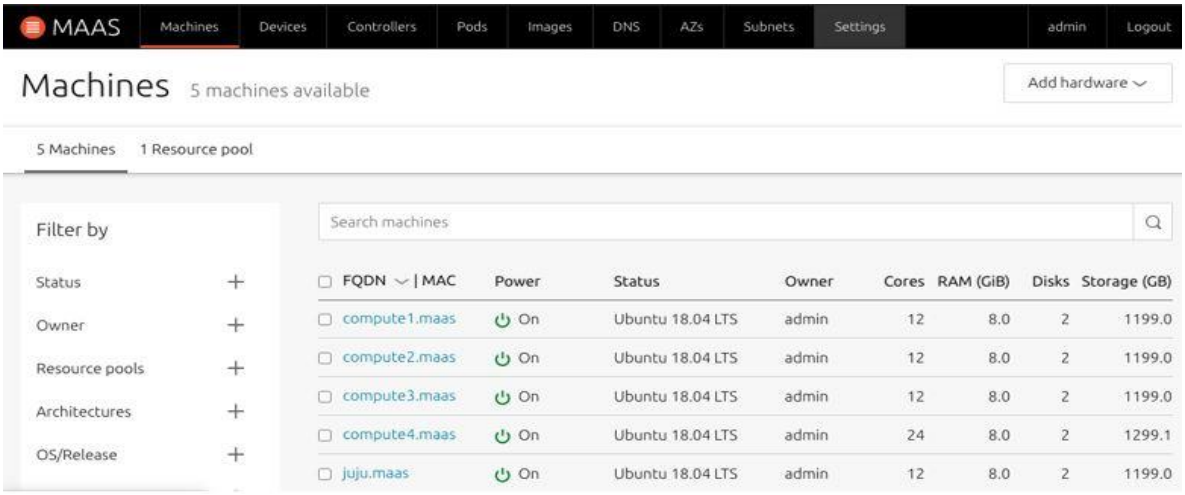

Figure 4: MAAS Dashboard showing the different virtualised machines for juju and computing nodes

2.3. JUJU

Juju is a cloud orchestration tool that assists in the administration of applications, infrastructure in cloud computing environments. It avoids the redundancies that come with having several configuration scripts and charts. It also saves many hours of script management work, lowers expenses, monitors all activities across all substrates applications, and maximises cloud setup (Tesfamicael et al., 2015).

OpenStack is composed of packages, which are referred to as "Charms" in Juju. Charms are small services or packages that incorporate cloud-based maintenance functions, including computing and networking activities. Juju also employs a Charmed Operator Lifecycle Manager, which enables cloud administrators to manage cloud applications without delving deeply into setups. Figure 5 present the applications of the configured juju controller.

17 applications: 17 active

APP	STATUS	VERSION	SCALE	STORE	REV	OS
ceph-mon	Active	14.2.11	3	CharmHub	44	Ubuntu
ceph-osd	Active	14.2.11	3	CharmHub	294	Ubuntu
ceph-radosgw	Active	14.2.11	1	CharmHub	283	Ubuntu
cinder	Active	15.4.1	1	CharmHub	297	Ubuntu
cinder-ceph	Active	15.4.1	0	CharmHub	251	Ubuntu
glance	Active	19.0.4	1	CharmHub	291	Ubuntu
keystone	Active	16.0.1	1	CharmHub	309	Ubuntu
mysql	Active	5.7.20	1	CharmHub	281	Ubuntu
neutron-api	Active	15.3.0	1	CharmHub	282	Ubuntu
neutron-gateway	Active	15.3.0	1	CharmHub	276	Ubuntu
neutron-openvswitch	Active	15.3.0	0	CharmHub	269	Ubuntu
nova-cloud-controller	Active	20.4.1	1	CharmHub	339	Ubuntu
nova-compute	Active	20.4.1	3	CharmHub	309	Ubuntu
ntp	Active	3.2	0*	CharmHub	36	Ubuntu
openstack-dashboard	Active	16.2.0	1	CharmHub	297	Ubuntu
placement	Active	2.0.0	1	CharmHub	19	Ubuntu
rabbitmq-server	Active	3.6.10	1	CharmHub	97	Ubuntu

Figure 5: Juju dashboard showing deployed application including nova, glance, cinder, etc.

Once an instance has been configured in OpenStack, a virtual computer similar to a public cloud such as Google Cloud or Microsoft AZURE is available. The OpenStack dashboard provides an interface that facilitates the configuration of instances. Figure 6 presents different configured instances on the OpenStack dashboard Horizon.

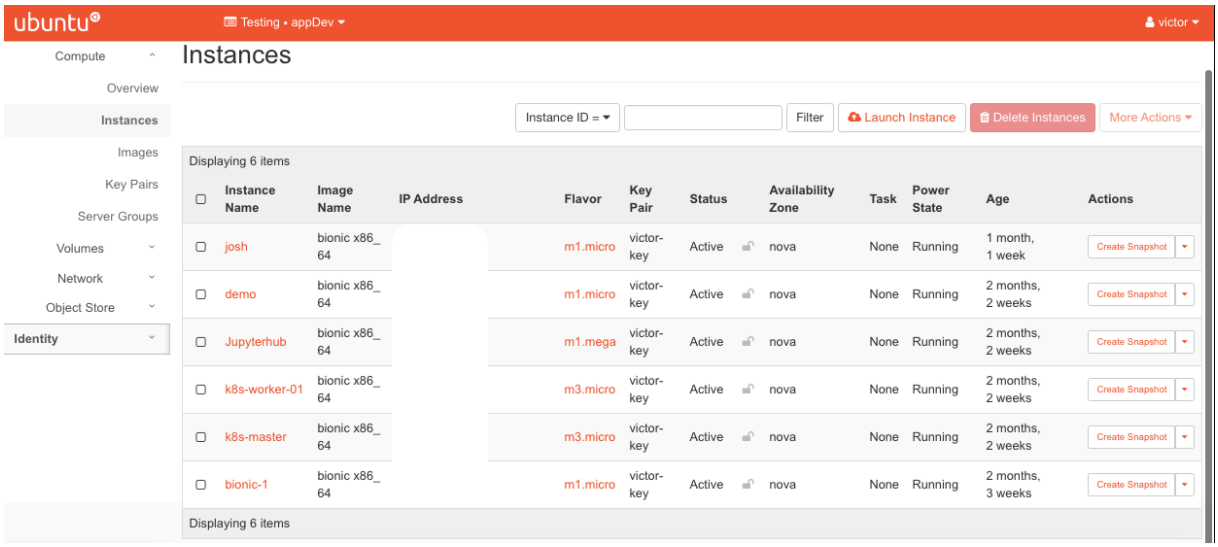

Figure 6:OpenSatck dashboard showing different created instances

2.4. Hardware requirement

The FEDGEN testbed comprises six bare metal servers where one server is used as MAAS, and another is used as Juju while the remaining servers are computing nodes. Table 2 presents the hardware requirement. Further information about the configuration of the FEDGEN testbed is given in (Adetiba et al., 2021).

Table 2: Hardware Requirement of the FEDGEN cloud testbed infrastructure

5 x Bare Metal 0 Dell Poweredge R620	
Processor	Intel® Xeon® E5-2620 12 cores @ 2 Ghz
Memory	8Gb
Storage	1 x 299 Gb (hdd) 1 x 900 Gb (hdd)
Network	4 x NICs
Form-factor	1U Rackmount
Second batch bare Metal	
1 x Bare Metal 1 Dell Poweredge R620	
Processor	Intel® Xeon® E5-2620 24 cores @ 2 Ghz
Memory	8Gb
Storage	1 x 299 Gb (hdd) 1 x 1000 Gb (hdd)
Network	4 x NICs
Form-factor	1U Rackmount
Top of Rack Switch	
Ports	48 x 10G (fiber/copper matching NIC specs)
Features	LACP, VLAN, MLAG/vPC

3. Hierarchical Anomaly detection model

To achieve the hierarchical anomaly detection model, several steps must be performed. These steps include the data acquisition, data pre-processing, training of the anomaly detection model and the performance assessment of the model, as shown in Figure 7.

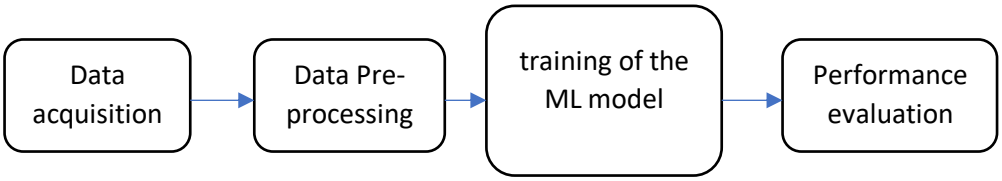

Figure 7:Block diagram of the steps to be performed in order to build the hierarchical namely detection model

3.1.Data acquisition and pre-processing

Log files help to retrace all the activities that are performed in a system chronologically. All the activities are stored in form text obeying to specific log format depending on the technology.

The data acquisition consists of the collection of different log files from the FEDGEN testbed. In this paper, three different log files were collected from the testbed. The different log files include Nova, Cinder, and Glance. To form the dataset, these three files were merged.

Before performing machine learning processing on the collected log files, the first step consists of parsing is the log. The log parsing allows transforming unstructured into structured files. Therefore, the collected log files were parsed with the regular expression (regex) library of python. Figure 8 shows the transformation of the unstructured Glance log file into a Comma Separated Value (CSV) file.

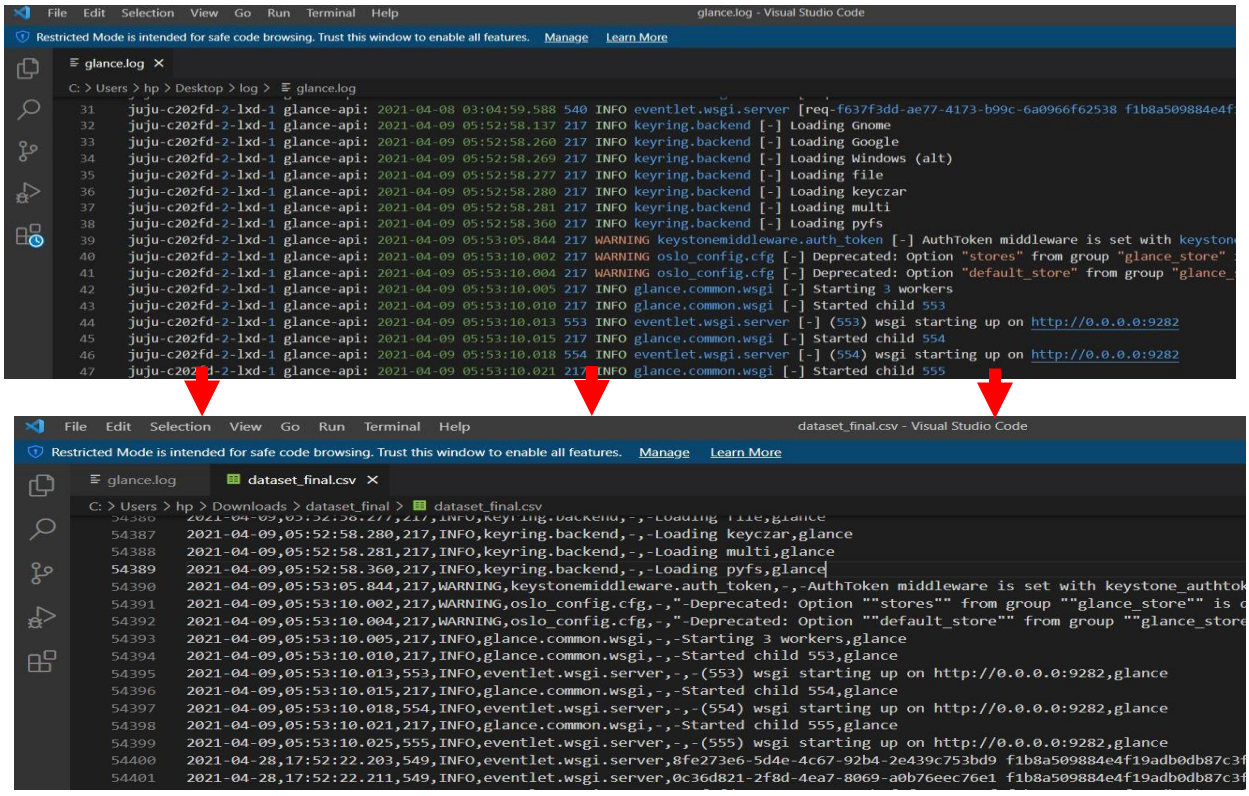

Figure 8: Log parsing using the regular expression library in order to separate different features of the dataset in a CSV file

The second step of pre-processing is data transformation with the Term Frequency-Inverse Document Frequency (TF-IDF) method that helps transform alphanumerical data into numerical data by automatically selecting only the most useful features in the corpus dataset. This method assumes that frequently, the most relevant word does not appear several times

necessarily in a document. It shows how relevant a word is in a given corpus. Figure 9 presents an overview of the collected log transformed into numerical values. The features affected by the transformation are Module, Message and Request.

	Date	Time	PID	...	MessageSum	ModuleSum	RequestSum
0	2021-04-07	14:00:03.697	108577	...	1.893757	1.000000	2.236068
1	2021-04-07	14:00:03.697	108577	...	2.199728	1.000000	2.236068
2	2021-04-07	14:00:03.933	108577	...	1.413910	1.000000	2.236068
3	2021-04-07	14:00:04.034	108577	...	1.412107	1.000000	2.236068
4	2021-04-07	14:00:04.047	108577	...	1.411897	1.000000	2.236068
...	...	...	...	...	...	...	...
953028	2021-06-18	06:25:09.324	2592	...	3.559220	1.414213	2.236055
953029	2021-06-18	06:25:09.360	2592	...	3.668828	1.705461	2.236055
953030	2021-06-18	06:25:09.460	2592	...	5.154845	1.408275	2.236055
953031	2021-06-18	06:25:09.533	2592	...	5.251762	1.408275	2.236055
953032	2021-06-18	06:25:09.819	2592	...	5.154845	1.408275	2.236055

Figure 9:Data transformation with TF-IDF showing the columns message, Module and Request turned into numerical values

3.2. Training of the Hierarchical anomaly detection model

The training of the hierarchical anomaly detection model consists of training and validation process of the dataset. The no free lunch theorem of optimisation supports that all optimisation algorithms perform equally well when performance is averaged over all possible problems. This indicates that there is no one-size-fits-all method for optimisation. Because optimisation, search, and machine learning are inextricably linked, this also indicates that there is no one ideal machine learning algorithm for predictive modelling tasks like classification and regression (Gómez & Rojas, 2016).

Three different algorithms, including LSTM, SVM and FFNN, will be trained and compared based on their performance. The choice of these algorithms is due to the fact that SVM and LSTM are commonly used in the literature. Also, the FFNN is chosen due to the fact that this algorithm is based on the Multi-Layer Perceptron (MLP) that is often used in the literature. The hierarchy, as presented in Figure 10, consists of 3 different levels. The first level classifies normal and abnormal events, the second level classifies different abnormal events, and the third level provides the root source of failure.

The dataset used to build the anomaly detection model, as presented in Table 3, comprises 953033 samples in total for 901 008 samples for normal and 52025 samples for abnormal data. Furthermore, due to the unbalanced nature of this data set, another set of data dedicated for testing is considered. It contains 36029 samples in total for 18203 normal events and 17826 samples as abnormal events (Molo, Victor, et al., 2021). The dataset also includes the following features:

- Date and time: this feature provides information about when an event takes place
- PID: represents the processing Identification of the event.

- The level of severity of the event gives information on whether an event is normal or abnormal. According to the standard logging level used by OpenStack, in the collected dataset, an event is labelled INFO or DEBUG to mean that it is normal and WARNING and ERROR when it is abnormal. However, generally other levels of severity for abnormal events such as AUDIT, CRITICAL and FATAL may be found in an OpenStack dataset to state an abnormal event.
- The Request ID: provide the information about the ID of an event request
- The message: Provide semantic information of a particular activity

Table 3: Summary of the structure of the dataset

Training set				
Level 1	**Normal**	**Abnormal**		**Total**
	901 008	52025		953033
Level 2	**Warning**	**Error**		**Total**
	7472	44553		52025
Level 3	**Storage**	**Image**	**Instance**	**Total**
	51665	10	350	52025
Testing set				
Level 1	**Normal**	**Abnormal**		**Total**
	18203	17826		36029
Level 2	**Warning**	**Error**		**Total**
	2501	15325		17826
Level 3	**Storage**	**Image**	**Instance**	**Total**
	17711	5	110	17826

Figure 10 shows the anomaly detection hierarchy structure. The structure is a DAG due to the nature of the dataset. Furthermore, it considers the LCL method as the classification approach. This is due to the fact that the different logfiles collected from Glance, Nova and Cinder directories were merged into a single dataset. Since the warning and error labels are all part of these logfiles, it is essential to determine the provenance of failure. Building the hierarchical model as DAG helps to spend less time on the training process while allowing the different models to learn all the possible classes associated with a particular level. Also, the imbalanced nature of the dataset may not allow other types of approaches, such as the LCN and LCPN, because these approaches require a considerable amount of samples in each class. Hence, the LCL approach is the most suitable because it will help avoid the underfitting or overfitting phenomena of the different models.

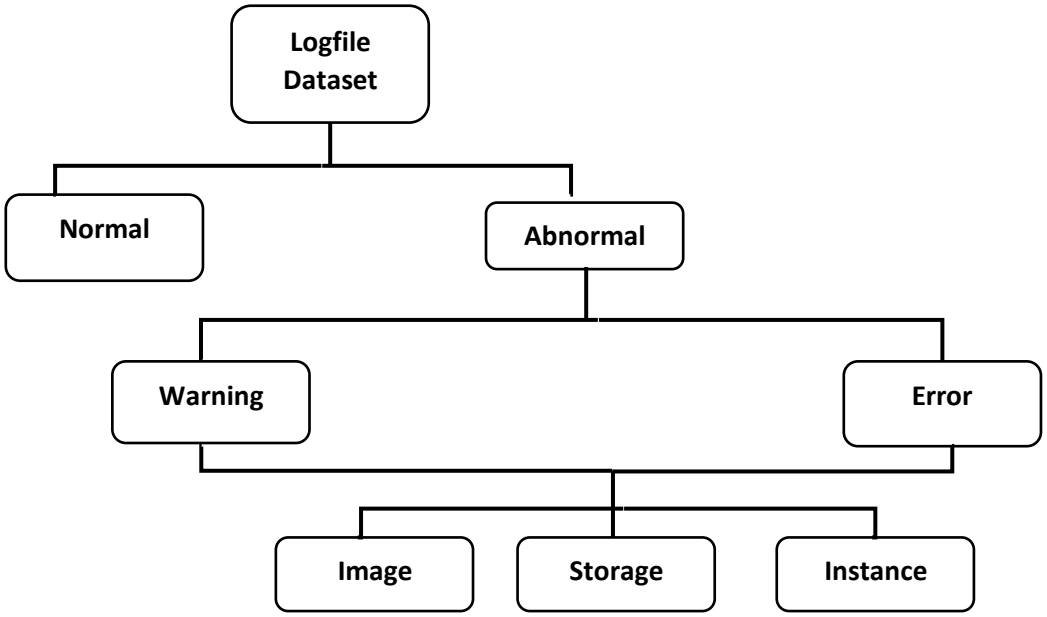

Figure 10: Different levels of the DAG where the first level is composed of two nodes (normal and abnormal), the second level is composed of two nodes (warning and error) and the third level is composed of three nodes (Image, storage and Instance)

To build the hierarchy, three different models are connected in cascade. Figures 11 and 12 present the block diagram of the overall hierarchical anomaly detection model. The two figures show how the detection is performed according to the previous level's output from one level to another. The second local classifier of the hierarchy is activated if the data point is an abnormal event. Otherwise, the first local classifier categorises the occurrence as a normal event. After the second local classifier has been activated, it classifies warning and error events. Whether the occurrence is classified as a warning or error, the third local classifier helps determine the source of the occurrence. Further classification can be performed after determining the component source for failure. A fourth local classifier can help determine which physical machine is affected since one OpenStack component can be installed on several machines.

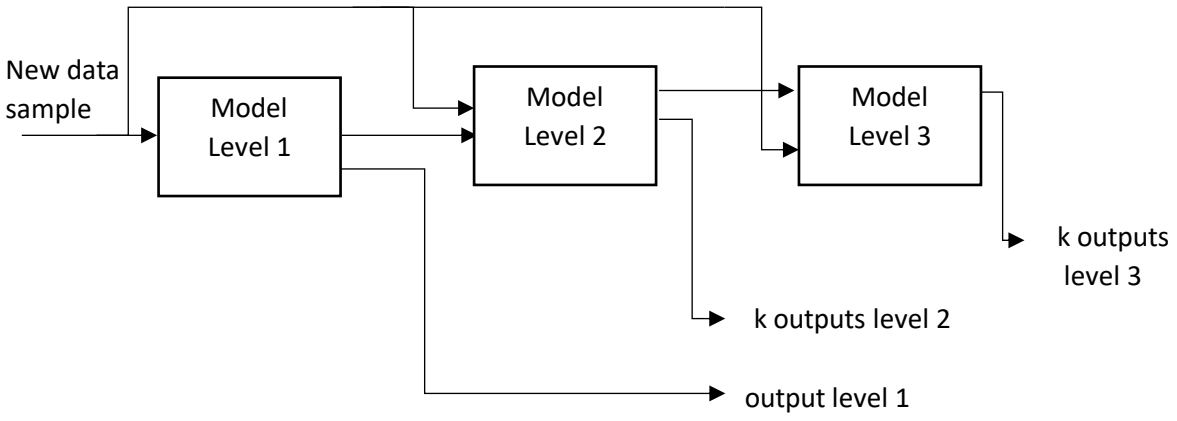

Figure 11: block diagram of the Hierarchical anomaly detection Model

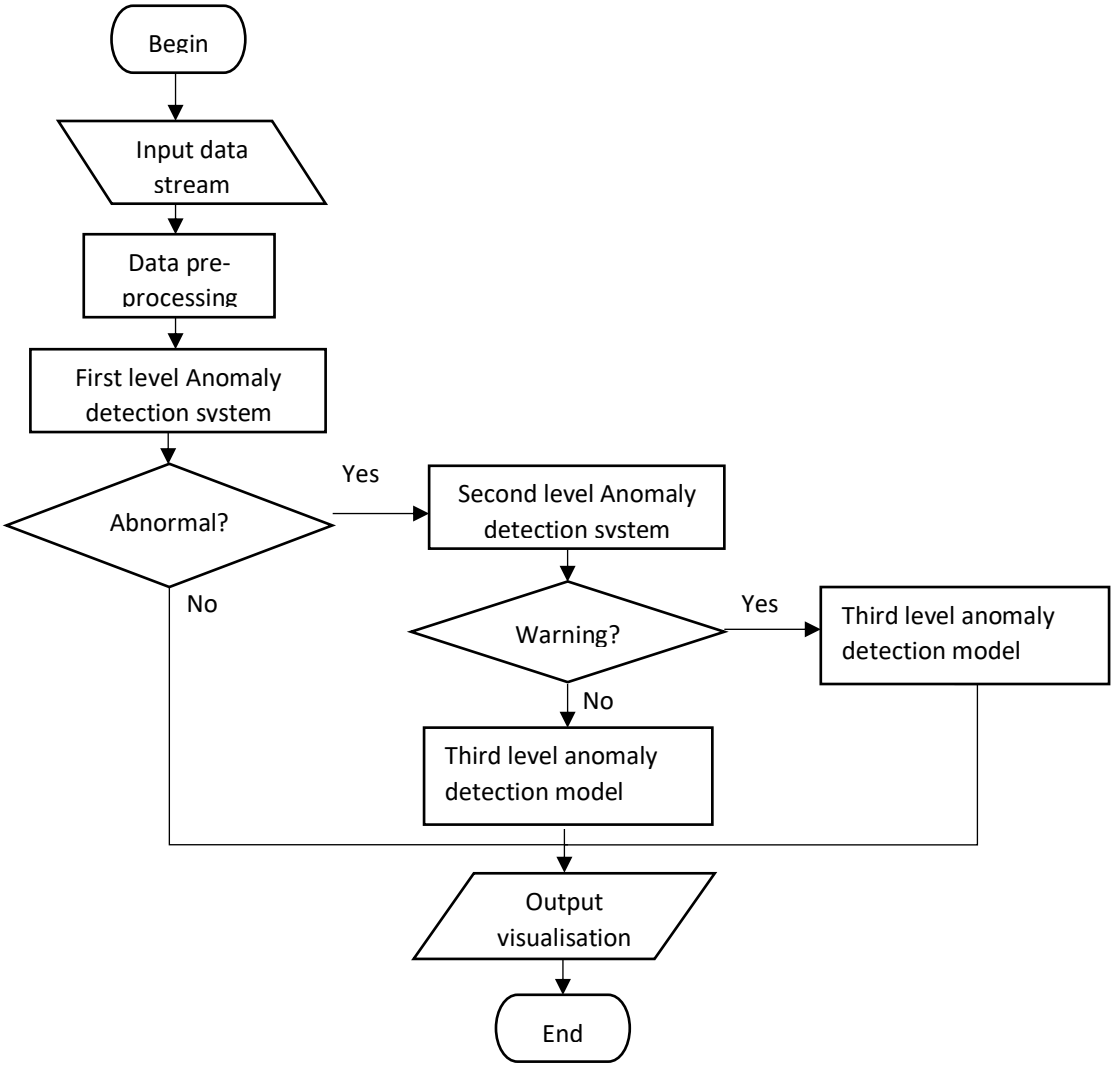

Figure 12: hierarchical anomaly detection sequence from level 1 to level 3.

Figures 13,14 and 15 present the block diagram of the three models. The first model classifies normal and abnormal events. Since the data is unbalanced, the abnormal samples have to be upsampled before further processing. The model is then trained and tested. Once an event has been detected to be abnormal, the event is sent and pre-processed accordingly to the second model. The model is then activated and classifies whether the event is an ERROR or a WARNING. For the level three model, the same sample is sent and pre-processed accordingly to be accommodated by the model. After Knowing whether an event is classified as an error or warning, the level 3 model gives information about the nature of the event (Instance, Storage, Image), which gives more light on the component source of failure. The failure can be an instance failure, Storage Failure, or Image failure. Considering that the data is imbalanced, the samples that are few are upsampled accordingly. Also, the dataset dedicated to testing the models contains new samples unseen by the model. The choice of dedicating a dataset for testing is due to the fact that after the upsampling process, the test set that may emanate from the splitting of the training dataset could contain replicas of the training set. As a result, the performance evaluation may be wrong.

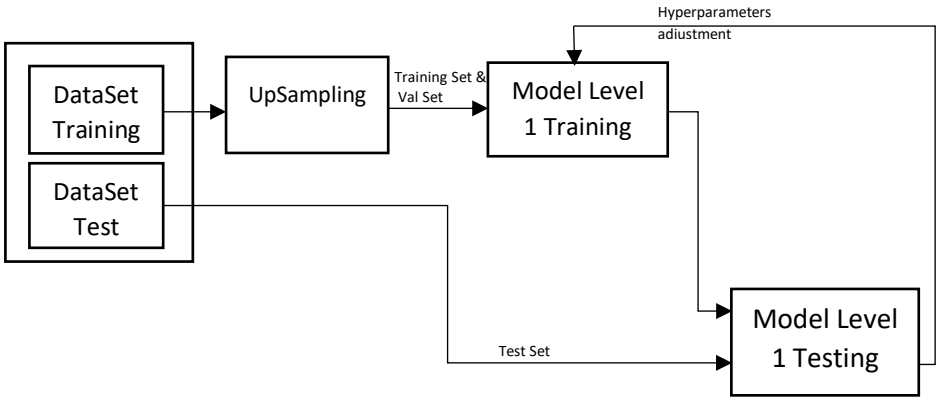

Figure 13: Block diagram of the first model of the hierarchy separating normal and abnormal samples. At this level, the abnormal samples are upsampled.

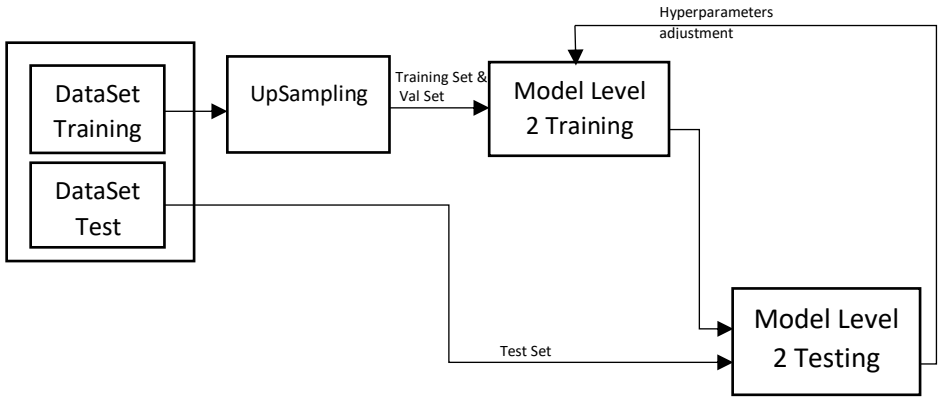

Figure 14: Block diagram of the Second model of the hierarchy separating error and warning. At this level, the warning samples are upsampled

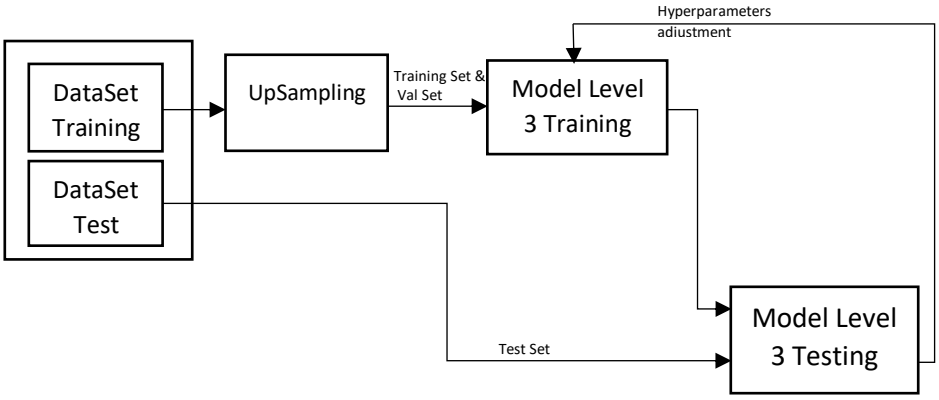

Figure 15: Block diagram of the third model of the Hierarchy determining the nature of the event (Instance, Storage or Image). At this level, Image and Instance samples are upsampled

3.2.1. Models architecture

(i) **FFNN:** Feedforward neural networks, also called multilayer perceptrons, have been shown to be able to approximate any function with any required accuracy when related requirements are fulfilled. Compared to neural networks with one hidden layer, neural networks

with two hidden layers need fewer hidden neurons. However, neural network practitioners are normally not interested in constructing a precise approximator since the data they utilise is frequently noisy, and the number of input-target sets is usually high. From a practical viewpoint, it has been shown that single-hidden-layer neural networks outperform networks with many hidden layers at the same degree of complexity. There is a clear trend toward adopting neural networks with just one hidden layer in engineering applications (Razavi & Tolson, 2011). Figure 16 shows the network architecture used to train different FFNN models at the first and second hierarchy levels. This architecture comprises three inputs, one dense layer of 128 neurons with ReLu as an activation function as a hidden layer and a dense layer output with one neuron using Sigmoid. The dropout between the hidden and output layers is set to 0.5 to avoid overfitting. Meanwhile, as illustrated in Figure 17, the third level FFNN model has the same architecture with four inputs and one dense layer providing three outputs. Further explanation about FFNN is provided in (Razavi & Tolson, 2011).

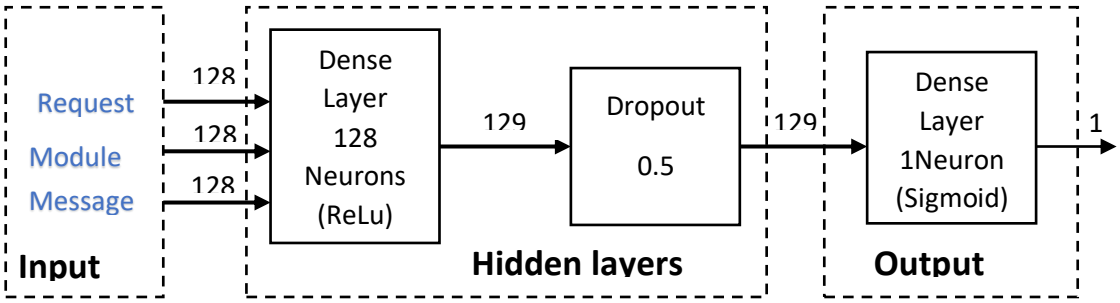

Figure 16: Network architecture of the FFNN model for the hierarchy level 1 and 2

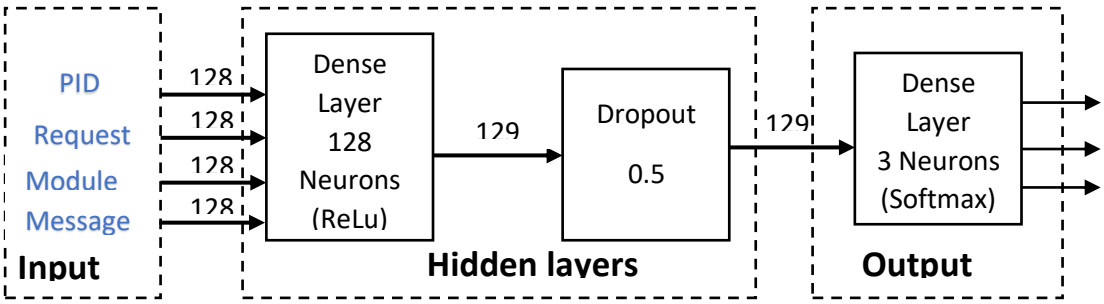

Figure 17: Network architecture of the FFNN model for the hierarchy level 3

(ii) **LSTM**: LSTM is a kind of recurrent neural network (RNN) architecture that was developed specifically for the purpose of modelling temporal sequences. Due to the long-range dependencies inherent in LSTM, it is more accurate than standard RNNs (Salman et al., 2018). As stated in the above paragraph, this paper considers a single layer LSTM because it can be sufficient to train and accurately classify different classes associated with each hierarchy level. Figure 18 shows the network architecture used to train different LSTM models at the first and second hierarchy levels. At the same time, Figure 19 illustrates the LSTM model's architecture of the third hierarchy level. The LSTM model's architecture comprises three layers, including the input layer with three inputs, one LSTM layer with 128 units where the recurrent dropout is 0.5. The output is a dense layer of one neuron for the first and second levels of the hierarchy

and three neurons for the third level of the hierarchy. Further details about LSTM is given in (Hochreiter & Schmidhuber, 1997).

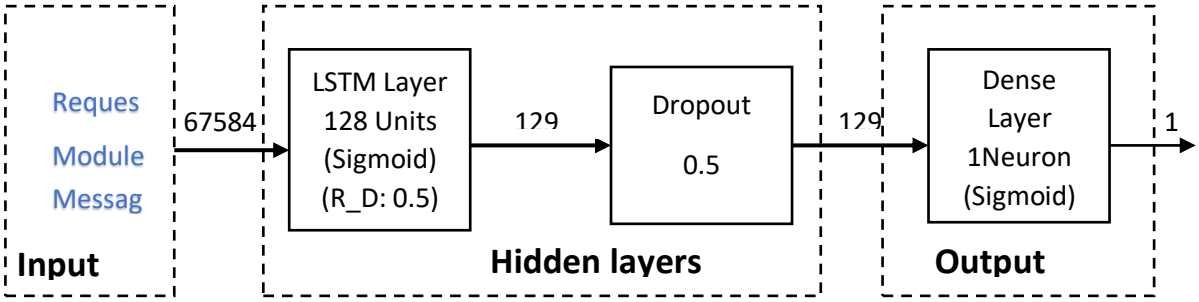

Figure 18: Network architecture of the LSTM model for the hierarchy level 1 and 2

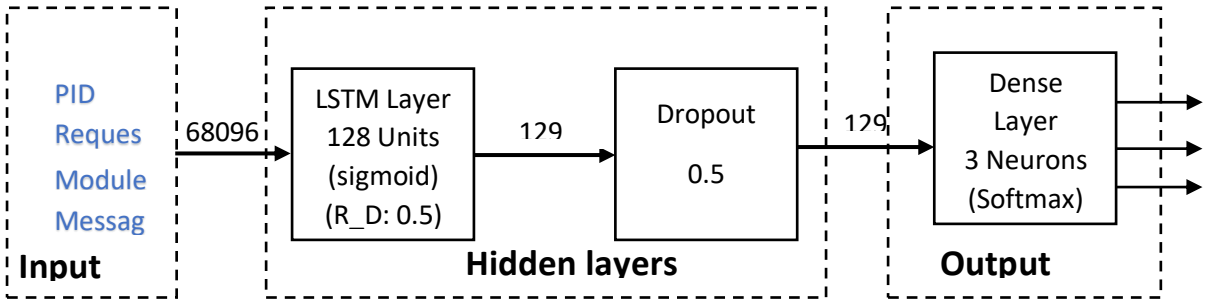

Figure 19: Network architecture of the LSTM model for the hierarchy level 3

(iii) SVM: the SVM algorithm used in this paper considers the tf.contrib.learn.svm module of TensorFlow. This approach allows fast training since the tf.contrib module added in the TensorFlow version 1.x can be run on a GPU. Further details about SVM can be found in (S. Ghosh et al., 2019). The L1 regularisation parameter used to train the different models is 0.1, while the TensorFlow kernel for SVM is linear.

3.3. Performance evaluation of the hierarchical anomaly detection model

Most academics assess hierarchical categorisation systems using traditional flat metrics such as accuracy, precision and recall. These metrics, however, are unsuitable for hierarchical classification since they do not distinguish between various types of misclassification errors. Intuitively, misclassification to an ancestor node of the proper category is preferable than misclassification to a distant node of the incorrect category (Serrano-Pérez & Sucar, 2021).

Many performance evaluation metrics are used in Hierarchical Classification, including accuracy (Babbar et al., 2013; Ramírez-Corona et al., 2016), hierarchical f-measure (Kiritchenko et al., 2006; Naik & Rangwala, 2016; Silla & Freitas, 2011), etc. This paper will consider the hierarchical f-measure (hF), which considers the ratio of accurate predictions to the total number of predictions in the dataset. The hierarchical precision (hP), which represents the proportion of correctly predicted labels to all of the real labels in the dataset and the hierarchical recall (hR) introduced by Kiritchenko et al. (2006) and suggested by Silla and Freitas (2011). These metrics are calculated as follow:

(i) **Hierarchical Precision:**

$$hP = \frac{\sum_i |y_i \cap \hat{y}_i|}{\sum_i |\hat{y}_i|} \quad (1)$$

(ii) **Hierarchical Recall:**

$$hR = \frac{\sum_i |y_i \cap \hat{y}_i|}{\sum_i |y_i|} \quad (2)$$

(iii)**Hierarchical F-measure:**

$$hF = \frac{2 * hP * hR}{hP + hR} \quad (3)$$

where y_i is the extended set of real classes with all ancestors of a particular class in the hierarchy and $\hat{y}_i$ is the extended set of predicted classes with all ancestors of a particular class in the hierarchy.

4. Results and discussion

As mentioned above, the hierarchy consists of 3 different levels. Therefore, three different local classifiers were built and assessed in order to take advantage of the most performant to form the hierarchy. The criterion of choosing the best local classifier is based on individual performance considering the anomaly detection scenario.

Table 4 presents performance evaluation metrics of the different trained local classifiers, including precision, recall and f-measure. Based on the obtained results, all the models provide admirable performance. However, the provided performance evaluation metrics do not provide a detailed appreciation of how each local classifier classifies different classes. Therefore, the confusion matrixes in Figure 20 help select the best model for the first level of the hierarchy based on the TP, FN, FP and TN occurrences.

Table 4: Performance evaluation of the local classifier at the first level of the hierarchy

	Precision	Recall	F-measure
SVM			
Normal	0.87	0.98	0.92
Abnormal	0.98	0.85	0.91
LSTM			
Normal	0.99	1	1
Abnormal	1	0.99	1
FFNN			
Normal	0.99	1	1
Abnormal	1	0.99	1

The confusion matrix of the SVM model in Figure 20 (a) shows that 2609 occurrences were classified normal while they are abnormal. And 362 occurrences were classified abnormal while they were not. In an anomaly detection scenario, it is not preferable to use a model with a large number of false positives occurrences because a system administrator may find the task challenging in detecting failure when it occurs. On the other hand, having many false-negative occurrences may lead to a tremendous number of false alarms, which constitute a drawback in anomaly detection systems. By comparing the false positive and false negative occurrences of the LSTM confusion matrix in Figure 20 (b) and Figure 20 (c), it is clear that the LSTM model

in Figure 20 (b) provides a lower number of false alarms. However, both the LSTM and FNN equally misclassify 113 abnormal occurrences into normal occurrences.

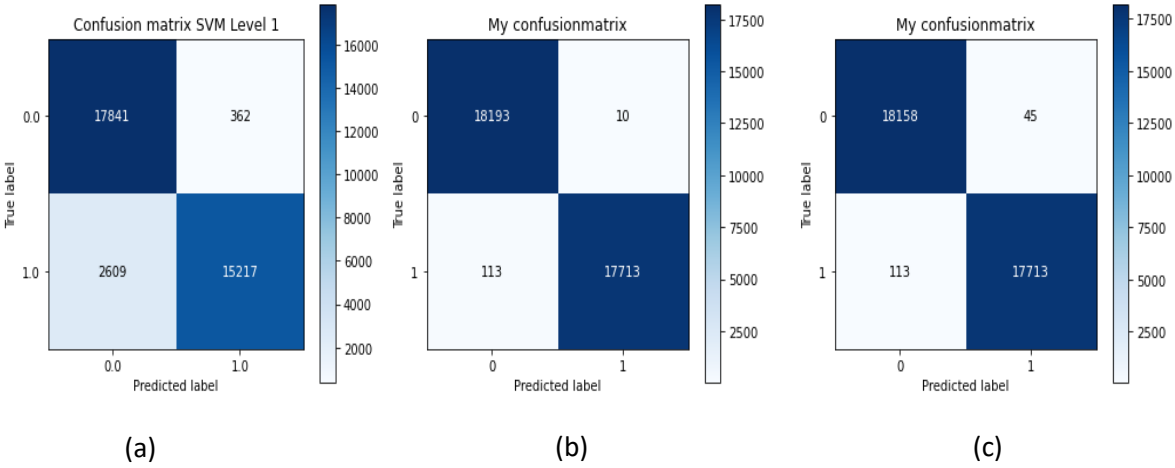

Figure 20: Level 1 confusion matrixes of SVM, LSTM and FNN

At the second level of the hierarchy, all the models also provided admirable output in terms of precision, recall and f-measure, as shown in Table 5. Although, the LSTM and FFNN models in Figures 21 (b) and 21 (c) performed equally for the second hierarchy level. They have the same number of false positive and false negative occurrences. On the other hand, the SVM model in Figure 21 (a) has much more false-positive occurrences than the others. This allows concluding that either the LSTM or the FFNN model can be used as a local classifier for the second hierarchy level.

Table 5: Performance evaluation of the local classifier at the second level of the hierarchy

	Precision	Recall	F-measure
SVM			
Warining	0.99	1	1
Error	1	1	1
LSTM and FFNN			
Warining	1	1	1
Error	1	1	1

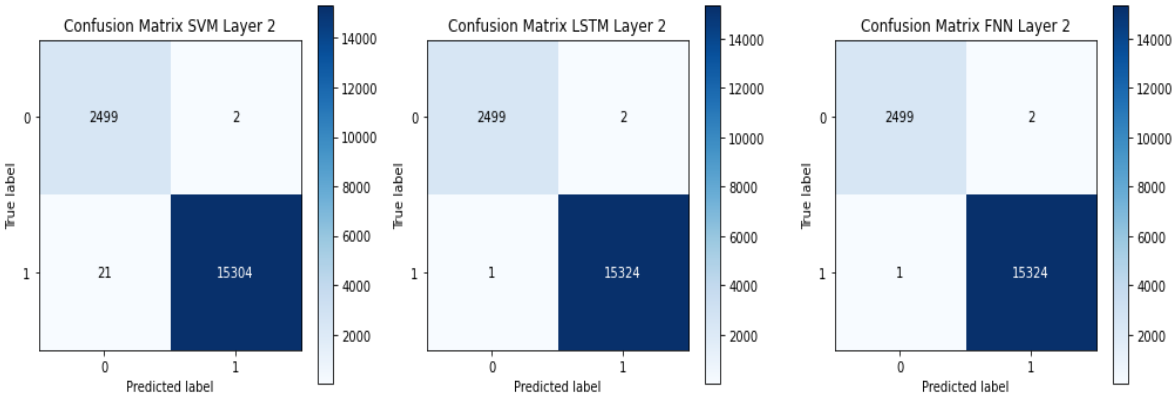

Figure 21:Level 2 confusion matrixes of SVM, LSTM and FNN

Figure 22 presents the confusion matrix of the third level of the hierarchy for LSTM and FFNN. The best model is the FFNN model at this hierarchy level, whose confusion matrix is presented in Figure 22 (b) because it correctly classified occurrences related to all predicted classes. In contrast, the LSTM, whose confusion matrix is presented in Figure 22 (a), could not classify instance occurrences correctly.

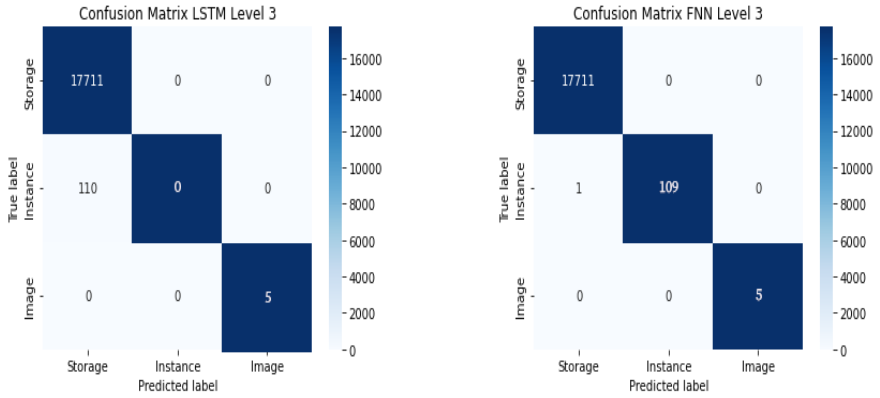

Figure 22: Level 3 confusion matrixes of LSTM, FNN

As mentioned in equations (1), (2), and (3), the $hP, hR, and\ hF$ can be generalised as follows in order to determine the overall performance of the hierarchical anomaly detection model:

$$hP_c = \frac{\sum_{l=1}^{L} TP_{cl}}{\sum_{l=1}^{L} TP_{cl} + \sum_{l=1}^{L} FN_{cl}} \quad (4)$$

$$hR_c = \frac{\sum_{l=1}^{L} TP_{cl}}{\sum_{l=1}^{L} TP_{cl} + \sum_{l=1}^{L} FP_{cl}} \quad (5)$$

$$hF_c = \frac{2 * hP_c * hR_c}{hP_c + hR_c} \quad (6)$$

where c represents a specific class, L is the number of classes starting from the leaf node up all its ancestor except the root. $TP, FN, and\ FP$ represents the True Positive, False Negative and False Positive values of the different predicted classes, respectively.

Table 6 summarises the classification report of the anomaly detection hierarchy starting from the different leaf nodes with Abnormal as ancestor node considering the LSTM, FFNN, and FFNN as models used to build the DAG:

Table 6: Performance evaluation of the DAG for anomaly detection

	hP	**hR**	**hF**
Instance	0.9967	0.9967	0.9967
Storage	0.9978	0.9978	0.9978
Image	0.9967	0.9967	0.9967

Cotroneo et al., (2021) proposed a clustering model that performed flat classification to categorise volume, Instance, Network, SSH, cleanup failures and normal events. Even though this approach provided good performance in terms of purity for up 94%, system administrators may find the task challenging in initiating the audit. Furthermore, considering the complexity of cloud computing environment especially, the OpenStack cloud environment, Figure 5 showed above that several cloud components are involved during the configuration process. In addition, these components can be installed on several machines at the same time. Therefore, a flat classification can be limited in providing admirable assistance in the detection of system failure. This paper's hierarchical anomaly detection approach proves that a hierarchical classification model can provide a better performance, as presented in Table 6 while increasing the granularity of a problem to end up with a small problem to facilitate the audit. On the other hand, the training time required for a hierarchical model is considerably higher than the training time required to build a flat classification model. Moreover, the tradeoff in this scenario is that a system administrator can choose one of the different approaches by taking into account the training time over the complexity of the flat classification approach and vice-versa.

5. Conclusion

This paper presented a hierarchical anomaly detection model of the FEDGEN tested, a cloud computing system based on OpenStack that enables creating OpenStack datasets contained in log files. The training and validation of the dataset are required for the construction of the hierarchical anomaly detection model. Several steps were performed to build the hierarchical anomaly detection model. These processes involve data collection, data pre-processing, the training of an anomaly detection model, and the model's performance evaluation. The hierarchy is divided into three levels: the first model classifies normal and abnormal events, the second

classifies ERROR and WARNING events, and the third level provides information about the failed component. The failure can be an instance failure, Storage failure, or Image failure. Based on their performance, three distinct models were trained and compared: LSTM, SVM, and FFNN. For the first level of the DAG, the results show that LSTM is the best model. LSTM and FFNN both performed better for the second level, and for the last level, the FFNN outperformed the LSTM. The performance evaluation of the DAG was expressed using a set of data dedicated for testing in terms of hierarchical precision, hierarchical recall and hierarchical F-measure.

According to the results of this study, further research, including the expansion of the DAG to detect abnormal events on different physical machines as the OpenStack component in charge of Instances, for example, can be installed on several machines. Also, further research, including implementing the hierarchical model on a federated cloud, should be explored.

References

Adetiba, E., Akanle, M., Akande, V., Badejo, J., Nzanzu, V. P., Molo, M. J., Oguntosin, V., Oluwadamilola, O., & Adebiyi, E. (2021). FEDGEN Testbed: A Federated Genomics Private Cloud Infrastructure for Precision Medicine and Artificial Intelligence Research. *International Conference on Informatics and Intelligent Applications (ICIIA 2021)*, 1–17. https://doi.org/Accepted

Aissa, N. B., & Guerroumi, M. (2016). Semi-supervised Statistical Approach for Network Anomaly Detection. *Procedia Computer Science*, *83*, 1090–1095. https://doi.org/10.1016/j.procs.2016.04.228

Akanle, M., Adetiba, E., Akande, V., Akinrinmade, A., Ajala, S., Moninuola, F., Badejo, J., & Adebiyi, E. (2020). Experimentations with OpenStack System Logs and Support Vector Machine for an Anomaly Detection Model in a Private Cloud Infrastructure. *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (IcABCD)*, 1–7. https://doi.org/10.1109/icABCD49160.2020.9183878

Azizi, Y., Azizi, M., & Elboukhari, M. (2019). Log files Analysis Using MapReduce to Improve Security. *Procedia Computer Science*, *148*, 37–44. https://doi.org/10.1016/j.procs.2019.01.006

Babbar, R., Partalas, I., Gaussier, E., & Amini, M.-R. (2013). Maximum-Margin Framework for Training Data Synchronization in Large-Scale Hierarchical Classification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 8226 LNCS* (Issue PART 1, pp. 336–343). https://doi.org/10.1007/978-3-642-42054-2_42

Cotroneo, D., De Simone, L., Di Martino, A., Liguori, P., & Natella, R. (2018). Enhancing the Analysis of Error Propagation and Failure Modes in Cloud Systems. *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 140–141. https://doi.org/10.1109/ISSREW.2018.00-13

Cotroneo, D., De Simone, L., Liguori, P., & Natella, R. (2021). Enhancing the analysis of software failures in cloud computing systems with deep learning. *Journal of Systems and Software*, *181*, 111043. https://doi.org/10.1016/j.jss.2021.111043

Daisey, K., & Brown, S. D. (2020). Effects of the hierarchy in hierarchical, multi-label classification. *Chemometrics and Intelligent Laboratory Systems*, *207*, 104177. https://doi.org/10.1016/j.chemolab.2020.104177

Farzad, A., & Gulliver, T. A. (2020). Unsupervised log message anomaly detection. *ICT Express*, *6*(3), 229–237. https://doi.org/10.1016/j.icte.2020.06.003

Freet, D., Agrawal, R., Walker, J. J., & Badr, Y. (2016a). Open source cloud management platforms and hypervisor technologies: A review and comparison. *SoutheastCon 2016*, *2016-July*, 1–8. https://doi.org/10.1109/SECON.2016.7506698

Freet, D., Agrawal, R., Walker, J. J., & Badr, Y. (2016b). Open source cloud management platforms and hypervisor technologies: A review and comparison. *SoutheastCon 2016*, *2016-July*, 1–8. https://doi.org/10.1109/SECON.2016.7506698

Ghosh, N., Maity, K., Paul, R., & Maity, S. (2019). Outlier Detection in Sensor Data Using Machine Learning Techniques for IoT Framework and Wireless Sensor Networks: A Brief Study. *2019 International Conference on Applied Machine Learning (ICAML)*, 187–190. https://doi.org/10.1109/ICAML48257.2019.00043

Ghosh, S., Dasgupta, A., & Swetapadma, A. (2019). A Study on Support Vector Machine based Linear and Non-Linear Pattern Classification. *2019 International Conference on Intelligent Sustainable Systems (ICISS)*, 24–28. https://doi.org/10.1109/ISS1.2019.8908018

Gómez, D., & Rojas, A. (2016). An Empirical Overview of the No Free Lunch Theorem and Its Effect on Real-World Machine Learning Classification. *Neural Computation*, *28*(1), 216–228. https://doi.org/10.1162/NECO_a_00793

Gulenko, A., Wallschlager, M., Schmidt, F., Kao, O., & Liu, F. (2016). Evaluating machine learning algorithms for anomaly detection in clouds. *2016 IEEE International Conference on Big Data (Big Data)*, 2716–2721. https://doi.org/10.1109/BigData.2016.7840917

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Islam, M. S., & Miranskyy, A. (2020). Anomaly Detection in Cloud Components. *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, *2020-Octob*, 1–3. https://doi.org/10.1109/CLOUD49709.2020.00008

Ismaeel, S., & Miri, A. (2015). A universal unit for measuring clouds. *2015 IEEE Canada International Humanitarian Technology Conference, IHTC 2015*. https://doi.org/10.1109/IHTC.2015.7238044

Jie, L., Xiangxiang, L., & Haoxiang, L. (2020). Anomaly detection method of power dispatching data based on cloud computing platform. *2020 International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, 23–26. https://doi.org/10.1109/ICBASE51474.2020.00012

Kiritchenko, S., Matwin, S., Nock, R., & Famili, A. F. (2006). Learning and Evaluation in the Presence of Class Hierarchies: Application to Text Categorisation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 4013 LNAI* (pp. 395–406). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11766247_34

Libri, A., Bartolini, A., & Benini, L. (2021). DiG: enabling out-of-band scalable high-resolution monitoring for data-center analytics, automation and control (extended). *Cluster Computing*. https://doi.org/10.1007/S10586-020-03219-7

Lu, S., Wei, X., Li, Y., & Wang, L. (2018). Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 151–158. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037

M. El-Shamy, A., A. El-Fishawy, N., Attiya, G., & A. A. Mohamed, M. (2021). Anomaly Detection and Bottleneck Identification of The Distributed Application in Cloud Data Center using Software–Defined Networking. *Egyptian Informatics Journal*. https://doi.org/10.1016/j.eij.2021.01.001

Marozzo, F. (2019). Infrastructures for High-Performance Computing: Cloud Infrastructures. In *Encyclopedia of Bioinformatics and Computational Biology* (Vols. 1–3, pp. 240–246). Elsevier. https://doi.org/10.1016/B978-0-12-809633-8.20374-9

Molo, M. J., Badejo, J. A., Adetiba, E., Nzanzu, V. P., Noma-Osaghae, E., Oguntosin, V., Baraka, M. O., Takenga, C., Suraju, S., & Adebiyi, E. F. (2021). A Review of Evolutionary Trends in Cloud Computing and Applications to the Healthcare Ecosystem. *Applied Computational Intelligence and Soft Computing*, *2021*, 1–16. https://doi.org/10.1155/2021/1843671

Molo, M. J., Victor, A., Nzanzu, V. P., Adetiba, E., & Badejo, J. A. (2021). *OpenStack log files*. https://doi.org/https://doi.org/10.6084/m9.figshare.17025353.v1

Mullerikkal, J. P., & Sastri, Y. (2015). A Comparative Study of OpenStack and CloudStack. *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*, 81–84. https://doi.org/10.1109/ICACC.2015.110

Musavi, P., Adams, B., & Khomh, F. (2016). Experience Report: An Empirical Study of API Failures in OpenStack Cloud Environments. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 424–434. https://doi.org/10.1109/ISSRE.2016.42

Naik, A., & Rangwala, H. (2016). Inconsistent Node Flattening for Improving Top-Down Hierarchical Classification. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 379–388. https://doi.org/10.1109/DSAA.2016.47

Patil, A., Wadekar, A., Gupta, T., Vijan, R., & Kazi, F. (2019). Explainable LSTM Model for Anomaly Detection in HDFS Log File using Layerwise Relevance Propagation. *2019 IEEE Bombay Section Signature Conference (IBSSC)*, *2019Januar*, 1–6. https://doi.org/10.1109/IBSSC47189.2019.8973044

Pattarakavin, T., & Chongstitvatana, P. (2016). Detection of Machines Anomaly from Log Files in Hard Disk Manufacturing Process. *2016 International Conference on Multimedia Systems and Signal Processing (ICMSSP)*, 60–63. https://doi.org/10.1109/ICMSSP.2016.022

Pyati, M., Narayan, D. G., & Kengond, S. (2020). Energy-efficient and Dynamic Consolidation of Virtual Machines in OpenStack-based Private Cloud. *Procedia Computer Science*, *171*, 2343–2352. https://doi.org/10.1016/j.procs.2020.04.254

Ramírez-Corona, M., Sucar, L. E., & Morales, E. F. (2016). Hierarchical multilabel classification based on path evaluation. *International Journal of Approximate Reasoning*, *68*, 179–193. https://doi.org/10.1016/j.ijar.2015.07.008

Razavi, S., & Tolson, B. A. (2011). A New Formulation for Feedforward Neural Networks. *IEEE Transactions on Neural Networks*, *22*(10), 1588–1598. https://doi.org/10.1109/TNN.2011.2163169

Salman, A. G., Heryadi, Y., Abdurahman, E., & Suparta, W. (2018). Single Layer & Multi-layer Long Short-Term Memory (LSTM) Model with Intermediate Variables for Weather Forecasting. *Procedia Computer Science*, *135*, 89–98. https://doi.org/10.1016/j.procs.2018.08.153

Sauvanaud, C., Kaâniche, M., Kanoun, K., Lazri, K., & Da Silva Silvestre, G. (2018). Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, *139*, 84–106. https://doi.org/10.1016/j.jss.2018.01.039

Sefraoui, O., Aissaoui, M., & Eleuldj, M. (2015). Management platform for Cloud Computing. *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, 1–5. https://doi.org/10.1109/CloudTech.2015.7336968

Serrano-Pérez, J., & Sucar, L. E. (2021). Artificial datasets for hierarchical classification. *Expert Systems with Applications*, *182*, 115218. https://doi.org/10.1016/J.ESWA.2021.115218

Silla, C. N., & Freitas, A. A. (2011). A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, *22*(1–2), 31–72. https://doi.org/10.1007/s10618-010-0175-9

Teixeira, C., de Vasconcelos, J. B., & Pestana, G. (2018). A knowledge management system for analysis of organisational log files. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, *2018-June*, 1–4. https://doi.org/10.23919/CISTI.2018.8399229

Tesfamicael, A. D., Liu, V., & Caelli, W. (2015). Design and implementation of unified communications as a service based on the open stack cloud environment. *Proceedings - 2015 IEEE International Conference on Computational Intelligence and Communication Technology, CICT 2015*, 117–122. https://doi.org/10.1109/CICT.2015.133

Yuan, Y., Anu, H., Shi, W., Liang, B., & Qin, B. (2019). Learning-Based Anomaly Cause Tracing with Synthetic Analysis of Logs from Multiple Cloud Service Components. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, *1*, 66–71. https://doi.org/10.1109/COMPSAC.2019.00019

Zhou, P., Wang, Y., Li, Z., Wang, X., Tyson, G., & Xie, G. (2020). LogSayer: Log Pattern-driven Cloud Component Anomaly Diagnosis with Machine Learning. *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 1–10. https://doi.org/10.1109/IWQoS49365.2020.9212954