LOG-BASED ANOMALY DETECTION

ST. TERESA'S COLLEGE(AUTONOMOUS)

AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfillment of the requirements for the award of the degree of

BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

By

FATHIMA FIDHA PS - SB21BCA014

&

LIBNAH MARIA ANIL - SB21BCA021

III DC BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

Under the guidance of

Ms. SREELAKSHMY I J

DEPARTMENT OF BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

MARCH 2024

LOG-BASED ANOMALY DETECTION

ST. TERESA'S COLLEGE(AUTONOMOUS)

AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfillment of the requirements for the award of the degree of

BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

By

FATHIMA FIDHA PS - SB21BCA014

&

LIBNAH MARIA ANIL - SB21BCA021

III DC BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

Under the guidance of

Ms. SREELAKSHMY I J

DEPARTMENT OF BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

MARCH 2024

DECLARATION

We, the undersigned, hereby declare that the project report, **"Log-Based Anomaly Detection**", submitted for partial fulfillment of the requirements for the award of a degree of BCA (Cloud Technology and Information Security Management) at St. Teresa's College (Autonomous), Ernakulam (Affiliated to Mahatma Gandhi University), Kerala, is a bonafide work done by us under the supervision of Ms. Sreelakshmy I J. This submission represents our ideas in our own words where ideas or words of others have not been included. We have adequately and accurately cited and referenced the sources. We also declare that we have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not previously formed the basis for the award of any degree, diploma, or similar title of any other University.

Ernakulam

March 2024

FATHIMA FIDHA PS - SB21BCA014

LIBNAH MARIA ANIL - SB21BCA021

ST. TERESA'S COLLEGE (AUTONOMOUS), ERNAKULAM

BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

DEPARTMENT OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the report entitled "**Log-Based Anomaly Detection**", submitted by **Fathima Fidha PS** and **Libnah Maria Anil** to the Mahatma Gandhi University in partial fulfillment of the requirements for the award of the Degree of BCA (Cloud Technology and Information Security Management) is a bonafide record of the work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

ST. TERESA'S COLLEGE DEPT. OF COMPUTERS ERNAKULAM

20/03/2024

26/March/24

Ms. SREELAKSHMY I J	Ms. ARCHANA MENON P	
Internal Supervisor	**Head of the department**	**External Supervisor**

ACKNOWLEDGEMENT

First and foremost, we thank God Almighty for his blessings. We take this opportunity to express our gratitude to all those who helped us in completing this project successfully. I wish to express our sincere gratitude to the Manager **Rev. Dr. Sr. Vinitha CSST** and the Principal **Dr. Alphonsa Vijaya Joseph** for providing all the facilities.

We express our sincere gratitude towards the Head of the department **Ms. Archana Menon P** for her support. We deeply express sincere thanks to our project guide **Ms. Sreelakshmy IJ** for her proper guidance and support throughout the project work.

We are indebted to our beloved teachers whose cooperation and suggestions throughout the project helped us a lot. We thank all our friends and classmates for their support.

We convey our hearty thanks to our parents for their moral support, suggestions, and encouragement.

ABSTRACT

Log-based anomaly detection is a technique used to identify unusual patterns or behaviors in system or application logs. In the context of computer systems, applications, or networks, logs are records of events, actions, or transactions that occur. Analyzing these logs can help detect abnormal activities that may indicate security threats, system malfunctions, or other issues. The traditional log anomaly detection method is no longer useful anymore due to the increasing complexity of the attacks. Recently, many machine learning and deep learning methods have been proposed to automatically detect anomalous log messages. But one or the other existing approach has various challenges related to data quality, lack of datasets, false alerting, etc. Many hybrid algorithms have been developed to find the abnormalities in datasets and are known to be more precise and efficient. This paper proposes a hybrid model of deep learning to distinguish between anomalous and non-anomalous events. Log files are created and then used for further analysis. A portal that helps users to find anomalies in their log files in a more user-friendly and easy way. A systematic literature review is also carried out in order to provide an overview of various deep learning models, data pre-processing mechanisms, anomaly detection techniques, and evaluations.

This study aims to help users understand relevant issues of log analysis and focuses on open issues for future work. and reconstructs log sequences through unsupervised training for anomaly detection. Finally, the proposed method was implemented. The results indicate that the proposed method can achieve a higher accuracy rate than the existing deep learning approaches. This proves the effectiveness and superiority of hybrid deep learning models.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DL - Deep Learning

CNN - Convolutional Neural Network

LSTM - Long-Short Term Memory Network

RNN - Recurrent Neural Network

DFD - Data Flow Diagram

CSV - Comma-Separated Values

KNN - K-Nearest Neighbors

DDPG - Deep Deterministic Policy Gradient

SMOTE - Synthetic Minority Over-sampling Technique

JSON - JavaScript Object Notation

CHAPTER 1
INTRODUCTION

1.1 GENERAL BACKGROUND

The evolution of log analysis has been driven by advancements in technology like distributed computing , cloud computing, and big data. Traditional methods of log analysis are proved to be inadequate as systems became more complex and generate large volumes of log data.This has led to the development of automated log management and analysis tools.These automated tools use various techniques such as machine learning, pattern recognition, statistical analysis,and anomaly detection to extract details from log data. Today, log analysis plays a critical role in maintaining the reliability, security, and performance of IT infrastructures across various industries. It enables organizations to proactively monitor systems, detect and mitigate security threats, troubleshoot issues, optimize resource utilization, and comply with regulatory requirements.

1.2 LOG FILES

A log is a file or record containing information about activities in a computer system. Log files provide a rich source of information when it comes to monitoring computer systems.A log file is a textual data file that stores events, processes, messages, and other data from applications, operating systems, or devices. They provide information based on the actions performed by users as well as monitor IT environments. Thereby, the majority of log events are usually generated as consequences of normal system operations, such as starting and stopping processes, restarting virtual machines, users accessing resources, etc.Logs are important for troubleshooting computer systems and keeping track of user activity. They can help inform decisions regarding system security and maintenance, provide evidence of malicious activity, and alert administrators to potential issues before they become serious problems. Logs also provide an audit trail to track system activity in case of a technical issue.

Each of the leading operating systems is uniquely configured to generate and categorize event logs in response to specific types of events. Log management systems centralize all log files, to

gather, sort and analyze log data, and make it easy to understand, trace, and address key issues related to application performance. Common log formats include CSV, JSON, Key Value Pair, and Common Event Format - each has benefits and specifications.

1.2.1 Types of Logs

Different types of logs exist. The following are the most significant logs generated by various software systems and network components:

- Authentication Logs - These types of logs track user login and logout activities in a computer system.
- System Logs - These logs record the events in an operating system. This includes system changes, critical system errors and events, unexpected shutdowns, and other important options.
- Security logs - These logs contain a record of activities that could harm the security of the system. It includes the removal of important files, failed login attempts, and incorrect logouts.
- Change logs - These logs keep track of all the changes or modifications that are made to an application or a file of the system. They are important for software developers. It includes information like the date and time of the change, the name of the person who made the change, etc.
- Server logs - It is a text document that contains a record of activities performed by a specific server over a specific period of time. These logs are important for monitoring how the server is being used, spotting any security issues, fixing any problems, etc.
- Event logs - These logs are files that capture and record important events that occur on the system. These events or activities include system startup and shutdown, user login or logouts, software installations, etc.
- Application logs - These logs capture events or errors that occur within applications, such as crashes or warnings.

1.2.2 Monitoring of Log Files

Log files store valuable information which can be used to recreate past events, find security flaws, or troubleshoot.

- Better Performance : By monitoring log files, you can decrease downtime, minimize the risk of data loss, and access valuable information like the need for updates or areas where performance may be improved. For example, timestamps in logs show us the time between events.
- Faster Troubleshooting: Log information can help determine what went wrong and troubleshoot when encountering an error. For example, if a service goes down, you can check the log files to see why it crashed.
- Security Checks and Pentesting : Security experts use log files as a trustworthy source of audit information since they provide a complete history of system activity such as access attempts, command line input, changes to sensitive information, and more.
- Understand User Behavior: Companies can use log monitoring to understand how users interact with their applications by going over the log files and looking into user behavior. This helps developers understand the user's needs better and optimize the application to fit them.
- Data Analysis : The data from the log files is often transferred to a secure server that serves as a centralized logging point. Log management software enables us to easily collect, parse, and analyze log files.

1.3 ANOMALY DETECTION

Anomalies are referred to as instances in the dataset that exhibit unexpected patterns that are completely different from the rest of the data. Anomaly detection identifies data that does not conform to expected patterns. It is a way to gain insight into resource behavior, allowing you to potentially catch issues before they escalate into more severe events. Log files could be automatically monitored to monitor abnormal activities and behavior in a system or a network.

Traditionally, developers usually check the logs manually with keyword search and rule matching. The volume of logs has increased as the scale and complexity of modern systems increase. To reduce manual effort, many anomaly detection methods based on automated log analysis are proposed.

1.4 DEEP LEARNING

Deep Learning is a subset of machine learning, which involves algorithms inspired by the arrangement and functioning of the brain. The deep learning algorithms run through various layers of neural network algorithms and learn from their reactions. It uses deep neural networks, which have multiple layers of interconnected nodes. These networks can learn complex representations of data by discovering hierarchical patterns and features in the data. Deep Learning algorithms can automatically learn and improve from data without the need for manual feature engineering.

Today deep learning has become one of the most popular and visible areas of machine learning. It has a variety of applications, such as computer vision, natural language processing, and reinforcement learning.

Deep learning can be used for supervised, unsupervised as well as reinforcement machine learning.

- Supervised Machine Learning: Supervised machine learning is the machine learning technique in which the neural network learns to make predictions or classify data based on the labeled datasets. Deep learning algorithms like Convolutional neural networks and recurrent neural networks are used for many supervised tasks. Supervised learning can be divided into two types of problems:

    - Classification: This type of problem includes categorizing data instances into predefined classes based on their features. The aim is to build a model from the labeled training data that can accurately classify new, unseen instances. The main goal of classification is to identify the category of the given dataset and is mainly used to predict the output for categorical data. Some of the classification algorithms are KNN, Decision Trees, Logistic Regression, etc.

      There are two types of classifications: Binary Classifier and Multi-class Classifier.

      Binary Classifier: If the classification has only two outcomes, then it is called Binary Classifier.

      Eg:- YES or NO, MALE or FEMALE, etc.

Multi-class Classifier: If the classification has more than two outcomes, then it is known as Multi-class Classifier.
Eg:- Dollars or Weight.

- Regression: This is a type of supervised machine-learning technique that focuses on predicting continuous numerical values based on input variables. The goal is to build a model that can estimate or approximate a target variable based on the relationships and patterns observed in the training data. Regression in supervised machine learning is crucial for tasks such as price prediction, trend analysis, and forecasting in various domains. Some of the regression algorithms are linear regression, polynomial regression, neural network regression, etc.

- Unsupervised Machine Learning: Unsupervised machine learning is the type of machine learning technique in which the neural network learns to discover the patterns or to cluster the dataset based on unlabeled datasets. Here there are no target variables. Deep learning algorithms like autoencoders and generative models are used for unsupervised tasks like clustering, dimensionality reduction, and anomaly detection. Clustering is a machine-learning technique, which groups the unlabeled dataset such as size , colour , shape , behavior etc

- Reinforcement Machine Learning: Reinforcement Machine Learning is the machine learning technique in which an agent learns to make decisions in an environment to maximize a reward signal. The agent interacts with the environment by taking action and observing the resulting rewards. Deep learning can be used to learn policies, or a set of actions, that maximizes the cumulative reward over time. Deep reinforcement learning algorithms like Deep Q networks and Deep Deterministic Policy Gradient (DDPG) are used to reinforce tasks like robotics and game playing etc.

1.4.1 Models

There are two types of deep learning models that are mainly used in the experiment : CNN and LSTM models.

A CNN (convolutional neural network) is a type of deep learning algorithm that is useful for visual analysis and language processing. Convolutional neural networks have an edge over other artificial neural networks due to their ability to process textual, visual, and audio data. The CNN architecture consists of three main layers: convolutional layers, pooling layers, and a fully connected (FC) layer. CNNs use a series of layers, each of which detects different features of an input image. Depending on the complexity of its intended purpose, a CNN can contain dozens, hundreds, or even thousands of layers, each building on the outputs of previous layers to recognize detailed patterns. Each additional layer that processes the input data increases the model's ability to recognize objects and patterns in the data.

Basic Architecture

There are three types of layers that make up CNN. When these layers are stacked, a CNN architecture will be formed.

1. Convolutional Layer

   The convolutional layer is the key component of convolutional neural networks.This is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

   The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image. Once the convolution operation is applied in the input, This layer in CNN passes the result to the next layer. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

2. Pooling Layer

   A pooling layer reduces the dimensionality of the input. Like a convolutional operation, pooling operations use a filter to sweep the whole input image, but it doesn't use weights. The filter instead uses an aggregation function to populate the output array based on the receptive field's values. Pooling layers are important in a CNN model as they reduce its complexity and increase the efficiency. It also reduces the risk of overfitting. There are two key types of pooling:

   Average pooling: The filter calculates the receptive field's average value when it scans the input.

   Max pooling: The filter sends the pixel with the maximum value to populate the output array. This approach is more common than average pooling.

3. Fully Connected Layer

   It is the final layer of a CNN model. The Fully Connected layer performs classification tasks with the help of the features extracted from the previous layers and filters. Instead of ReLu functions, the FC layer typically uses a softmax function that classifies inputs more appropriately and produces a probability score between 0 and 1.This is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

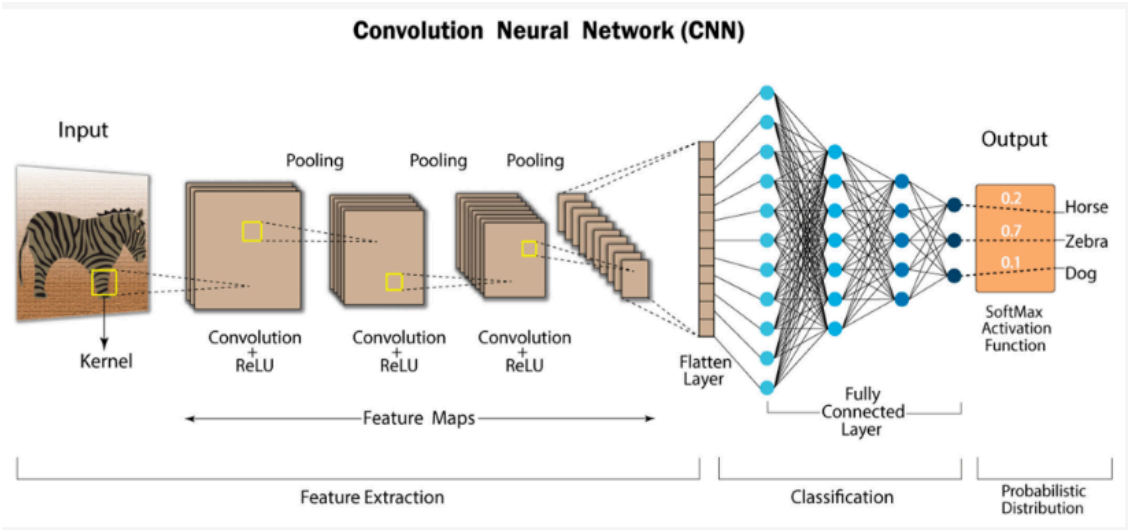


Fig 1.1: Architecture of CNN

A long short-term memory network, also referred to as LSTM, is a variation of an RNN model. It was proposed in 1997 by Sepp Hochreiter and Jurgen Schmidhuber. LSTM has feedback connections. It can process single data points as well as entire sequences of data . It adds extra structures, or gates, to an RNN to improve memory capabilities. An RNN can only memorize short-term information while LSTM can handle long time-series data. They use a memory cell and gates to control the flow of information, allowing them to selectively retain or discard information as needed and thus overcoming the limitations of traditional RNNs.

Basic Architecture

The central role of an LSTM model is held by a memory cell known as a 'cell state' that maintains its state over time. The cell state is the horizontal line that runs through the top of the below diagram. It can be visualized as a conveyor belt through which information just flows, unchanged.

The structure of an LSTM network consists of a series of LSTM cells, each of which has a set of gates. The gates allow the LSTM to maintain long-term dependencies in the input data as they selectively forget or retain information from the previous time steps. The output of each LSTM cell is passed to the next cell in the network, allowing the model to process and analyze sequential data over multiple time steps.

There are three types of gates in an LSTM: the input gate, the forget gate, and the output gate. These gates are all implemented using sigmoid functions, which produce an output between 0 and 1. These gates are trained using a backpropagation algorithm through the network.

- **Input gate:** It controls the flow of information into the memory cell. It is trained to open when the input is important and close when it is not. The output gate controls the flow of information out of the LSTM and into the output.
- **Forget gate:** The forget gate controls the flow of information out of the memory cell. It is trained to open when the information is no longer important and close when it is.
- **Output gate:** The output gate is responsible for deciding which information to use for the output of the LSTM.

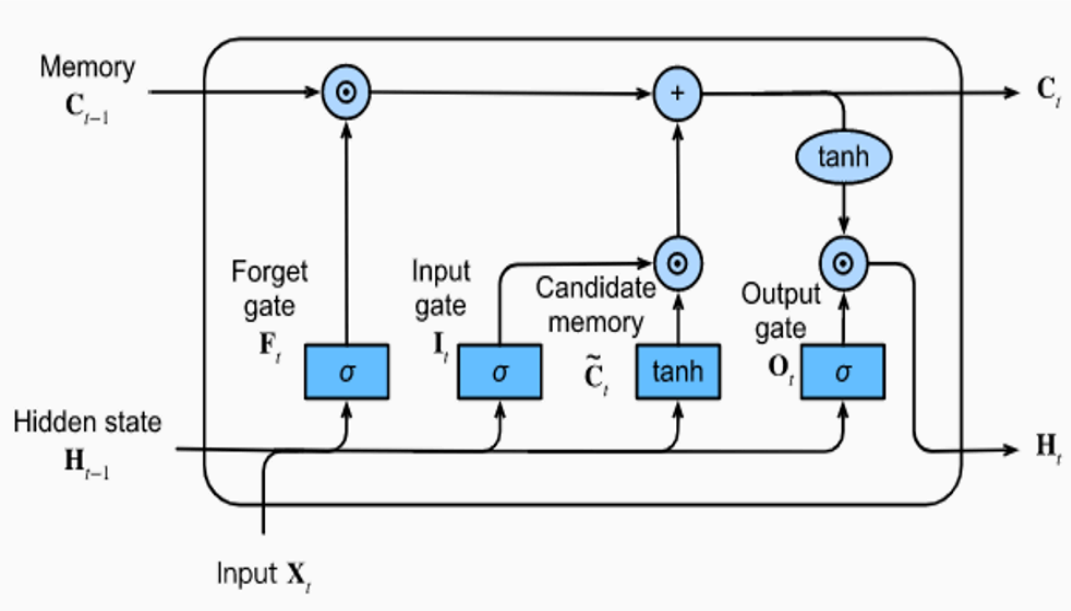


Fig 1.2 : Architecture of LSTM

1.5 HYBRID DEEP LEARNING

Hybrid deep learning refers to a combination of different types of deep learning architectures or models within a single system. This could involve combining convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other types of neural networks to address specific challenges or tasks. It can also integratE deep learning models with traditional machine learning algorithms to create more powerful and versatile systems.

Various advantages of using hybrid deep learning are:

- **Greater Accuracy:** Due to the combination of various models,it can produce more accurate results than traditional models.
- **Versatility:** Hybrid models are ideal for diverse applications and can handle a variety of data types and structures.
- **Efficiency:** These models can deal with complex scenarios more efficiently and thus reduce the computational load.
- **Improved Learning:** Hybrid Machine Learning benefits from the strengths of both machine learning and advanced AI, ensuring enhanced learning and prediction capabilities.

1.6 PERFORMANCE MANAGEMENT

To evaluate the performance of deep learning algorithms many different scoring techniques can be used.

1.6.1 F1-Score

The F-score (F1 score or F-measure) is a metric used to evaluate the performance of a Machine Learning model. It combines the precision and recall scores of a model.Accuracy in making positive predictions is measured by a recall, while identifying all positive occurrences in the data is quantified by precision. The F-score ranges from 0 to 1, with higher values indicating better performance.

$$F - score = \frac{2*(precision*recall)}{precision + recall}$$

$$Precision = \frac{True_{positive}}{True\ positive + False_{negative}}$$

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}$$

Evaluation of anomaly detection methods commonly depends on four different cases, true or false negatives and true or false positives.

1.6.2 Accuracy

Accuracy is a metric that measures how often a machine learning model correctly performs across all classes. Accuracy can be calculated by dividing the number of correct predictions by the total number of predictions. It covers all possible cases and rewards methods for finding true negatives. When every prediction made by the model becomes correct, a perfect accuracy of 1.0 is achieved.

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

1.7 OBJECTIVES

The main objective of this project is to distinguish between anomalies and non-anomalies in the log files of a system. There is a need to implement log anomaly detection techniques in order to safeguard our system from unusual events. A hybrid deep learning model, the CNN-LSTM model is used here. This is implemented to identify anomalies in a log file with more precision and accuracy. A system where users can upload and download the log files is developed.

CHAPTER 2
LITERATURE SURVEY

The paper analyzes the working of deep learning models and looks at how well these models can find anomalies in computer systems by looking at log data. An in-depth analysis of five state-of-the-art deep learning-based models is being conducted. Four public log datasets, namely HDFS, BGL, Thunderbird, and Spirit are used to evaluate the studied models for anomaly detection in logs. Several aspects of model evaluation that include training data selection strategies, data grouping, class distribution, data noise, and early detection capability are investigated. The result points out that all these aspects have a large impact on the evaluation results and the performance of the models is not as accurate as expected.[1]

This paper shows a survey of 62 scientific approaches by which deep learning is used for the detection of anomalous events or processes in the log data of a computer or a system. Here, various model architectures are used. These include models for sequential input data such as recurrent or convolutional neural networks, language-based models such as transformers, as well as unsupervised models such as Autoencoders or generative adversarial networks. There are different features used for training and detection. The processing of these features is done by encoding them as numeric vectors. Anomalies are then detected either directly through classification or by deriving some kind of anomaly score from the network that allows to discern normal from anomalous system behavior. Still, there are some challenges that are unable to be resolved by existing approaches, like detection techniques that go beyond sequential anomalies, low explainability of trained models and classification results, lack of representative evaluation data sets containing diverse attack artifacts, and a low reproducibility.[2]

Another paper proposes an anomaly detection model based on Long Short-Term Memory to analyze the content of log messages in datasets. Two specific datasets, namely Knet2016 and R6.2 are used to train and test the proposed model. The results show that the model performed well and was able to identify exceptions within the log data. When compared to other existing methods (PCA, one-class SVM, and GMM), this model showed better results in detecting

anomalies. It is also noted that the training and testing should be carried out on datasets by using more data sources and isolation forest should be introduced as a baseline for comparative experiments. This paper also describes the possibilities of exploring models with more layers that could be useful for analyzing individual log messages. Finally, it has proposed ways to combine information from different parts of the logs to improve overall anomaly detection. Therefore, a better integration method could improve the performance of the model. [3]

This research proposes a new way to automatically detect anomalies in computer system logs. An unsupervised framework is used for real-time anomaly detection and does not require any pre-labeled data. This framework has two main stages. In the first stage, it builds a knowledge base of frequent patterns in the logs using clustering. Then a streaming anomaly detection phase is used to detect anomalies in real time. An HDFS log dataset was used in this experiment and an F-1 score of around 83% was obtained. Thus it is said to achieve a high accuracy score on a large log dataset. Our framework shows a novel perspective to anomaly detection in which, rather than alerting whenever an event trace is abnormal, we alert whenever an event seems abnormal in the context of the most recent events that occurred before it. [4]

In another paper, an approach analyzes the logs by combining a method of feature extraction with an anomaly detection algorithm from deep learning. For feature extraction, word2vec is used and it converts log messages into a format that an anomaly detection algorithm can understand. Then a deep autoencoder model with Long Short-Term Memory (LSTM) units is used for anomaly detection. To make the analysis of data easier, several techniques like data for dimension reduction are used. After detecting anomalies, they are further classified into different web attacks i.e. brute force, port scanning, and SQL injection. The experimental findings show the effectiveness of the proposed anomaly detection learning algorithm. [5]

A research paper has provided a detailed review and evaluation of six state-of-the-art log-based anomaly detection methods. These include three supervised methods and three unsupervised methods. These methods have been evaluated on two publicly available production log datasets, with a total of 15,923,592 log messages and 365,298 anomaly instances. The accuracy and efficiency of these models are compared. In addition, an open-source toolkit of these anomaly

detection methods is released for easy reuse and further study. To improve the efficiency of models, methods that could reflect the nature of anomalies are highly desired. Real-time log analysis is also required to be considered for future analysis.[6]

This paper demonstrates an idea of LogC. It is a new log-based anomaly detection approach with Component-aware analysis. LogC contains two phases. In the first phase log messages are converted into log template sequences and component sequences. These sequences are fed to train a combined LSTM model for detecting anomalous logs. LogC only needs normal log sequences to train the combined model. Two open-source log datasets, namely HDFS and ThunderBird are used for the evaluation. Experimental results show that LogC overall outperforms three baselines (i.e., PCA, IM, and DeepLog) in terms of three metrics (precision, recall, and F-measure). [7]

This paper explains how deep learning can be used to detect anomalies in the computer network. In order to maintain the network's security, they have designed and implemented deep learning methods, specifically, the LSTM algorithm and Attention mechanism to build a deep neural network model and enhance the model's performance. In handling the class-imbalance problem with the CSE-CIC-IDS 2018 dataset, researchers have employed two key strategies: Synthetic Minority Over-sampling Technique (SMOTE) and improved loss functions. As a result, it shows that the classification accuracy of the model reaches 96.2% which is higher than the machine learning algorithms. [8]

CHAPTER 3
EXISTING SYSTEM

The Existing system employs individual use of two distinct algorithms for anomaly detection, namely: Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). Convolutional Neural Network is also known as CNN or ConvNet. It is a type of deep learning model designed for tasks like image recognition and computer vision, it has undergone adaptation for the analysis of sequential data such as time series data or logs, by utilizing one-dimensional convolutions. These classification-based models play a pivotal role in identifying anomalies through the classification of log sequences.

On the other hand, Long Short-Term Memory is a specialized type of Recurrent Neural Network, specifically designed to overcome the challenges associated with RNN. An RNN can only memorize short-term information while LSTM can handle long time-series data. It excels in handling long-term dependencies, making it effective for sequential log data. It shows inherent proficiency in capturing contextual relationships between elements within a sequence. The effectiveness of LSTM networks depends on several factors, including the quality and quantity of training data, choice of architecture, and hyperparameter tuning.

Despite the incorporation of these advanced algorithms, the accuracy achieved in the current system has fallen short of the desired outcome. This difference highlights the need for a thorough examination of the system's details and refining the model. Regular evaluation and validation of the model's performance are crucial in identifying areas of improvement. By monitoring metrics like precision, recall, F1 score and conducting systematic experiments, it becomes possible to identify and address specific issues that may be hindering the desired accuracy. Such examination is crucial to bring out its full potential and attain the desired accuracy threshold.

3.1 Drawbacks of Existing System

Several drawbacks were encountered in the utilization of existing systems. Some of them are as follows:

- **Lack of sufficient labeled data:** CNN is a powerful model used for image and video processing. However, they do require a large amount of labeled data for training, which can be challenging to obtain in real-world scenarios, especially when manual labeling is involved.

- **Performance on Imbalanced data:** When dealing with imbalanced data, CNN may not be performed well and the accuracy of the model can decrease. An imbalance of data means the classification of a dataset with skewed class proportions. CNN can also struggle with noisy data. Noise data refers to irrelevant or misleading information in the dataset. When CNNs are presented with noisy data, it can affect their ability to capture patterns, resulting in reduced accuracy.

- **Computational Intensity:** LSTM is renowned for sequential data analysis. However, they can be computationally intensive, especially with large-scale datasets. Training and running LSTM models may require substantial computational resources and can be time-consuming.

- **Handling Long-Term dependencies:** LSTMs are effective in capturing long-term dependencies within sequential data. They may encounter challenges as the sequence length increases. As the sequence length becomes very long, LSTMs face difficulties in retaining relevant information from the beginning of the sequence. LSTM has fixed memory capacity and when presented with very long sequences, they may struggle to retain relevant context over extended periods.

3.2 Functional Workflow of Existing System

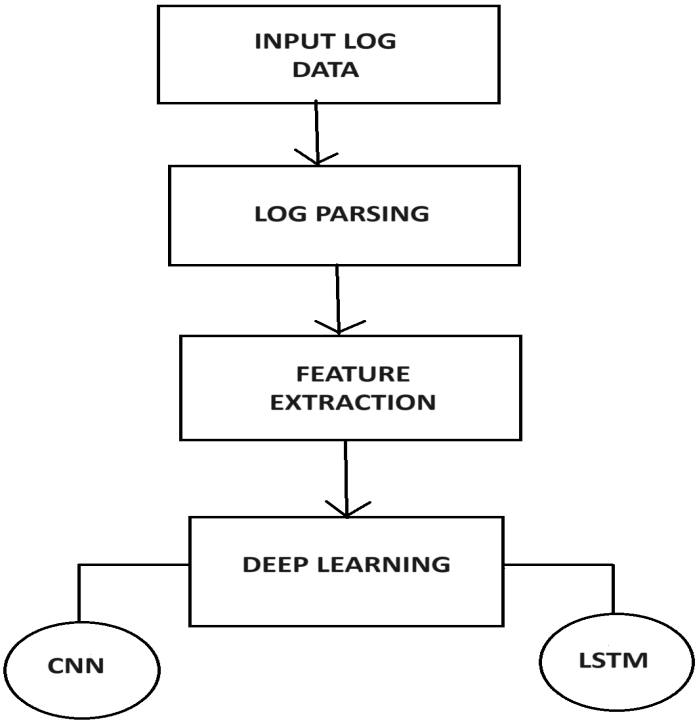


Fig 3.1 : Existing System

In the diagram:

- **Input log data:** This is the input data in the form of logs.
- **Log parsing:** The log data is parsed to extract relevant information
- **Feature Extraction:** Features are extracted from the parsed log data, which may include information relevant to the analysis.
- **Deep learning models:** The extracted features are used as input to deep learning models such as CNN and LSTM for further analysis, and prediction.
- **Output:** The output of the models depends on the taste, such as predictions, and classification derived from the log analysis.

CHAPTER 4
PROPOSED SYSTEM

The existing system is implemented to detect anomalies in a system using various deep learning models like RNN and CNN.Though these experiments provide good models for detecting log anomalies,there are still some challenges such as limited datasets,quality of data, prediction accuracy and effectiveness of models. We have proposed a hybrid deep learning model that uses CNN and LSTM algorithms to detect anomalies in a log dataset. The main objective of this system is to identify unusual events or patterns in log data by combining these two algorithms together, which helps in improving the accuracy of the system. The proposed model is created to capture spatial patterns within individual log entries and understand the temporal flow of log events.

The CNN algorithm is effective in capturing spatial patterns within log data while the LSTM model is ideal for capturing the temporal sequence of log data. The hybrid model of CNN-LSTM is a powerful algorithm as it combines the strength of CNN and LSTM models. In the hybrid model, data is processed in the CNN part and the result of this processing is fed into the LSTM model.

The experimental findings show that hybrid models have an overall good effect on the accuracy of the prediction.This ensures the robustness and high resilience of the trained model.The results show that many hybrid algorithms including CNN-LSTM may largely outperform the standalone DL models.It is known to achieve optimal performance when compared to other models by attaining better accuracy and thus produces high F measure.With the CNN-LSTM hybrid model, this project aims to create a reliable system that can accurately detect any unusual patterns or events based on the log data. The proposed model consists of a convolutional layer for feature extraction, followed by an LSTM layer for sequence processing, and finally, a dense layer for binary classification.This combination helps the model to tackle more complex tasks, deal with a variety of data types, and produce more accurate and efficient results.

To train the proposed model, a log dataset is being generated. This dataset consists of log entries that capture different types of patterns or events. After the phase of preprocessing and feature extraction, the model is being trained using the provided datasets.he accuracy of anomaly detection can improve. The proposed hybrid model was implemented in Python and its performance was evaluated on the dataset generated by the user.

Prediction is the core function of a trained model. Once the evaluation of the model is completed, it could be used to make predictions on entirely new data that the model has never seen before.
In this project, a web portal is developed so that users can upload log files in order to check if they contain anomalies or not. The web application will make use of the pre-trained model to analyze the uploaded files and predict the likelihood of anomalies. This way, users can get instant insights into potential issues within their systems without needing to write any code themselves.

4.1 Objectives of the proposed system

Introducing our proposed model, this advanced system aims to:

- **Normal flow of events:** To understand the normal flow of events and detect any deviations that may occur over time. It utilizes advanced algorithms to analyze large volumes of data, identify patterns, and establish what is considered “normal” behavior. Once the baseline has been established, the model can then identify any deviations from this norm.
- **Detect anomalies over time:** By processing log sequences in a time-ordered manner, the model can effectively identify anomalies over time. This allows for early detection and timely response to potential issues, reducing their impact on the performance and security of the system.
- **Reduce false positives:** An important objective of this model is to differentiate between normal variations and actual anomalies in the log data. By distinguishing between these two, they can reduce false positive alerts, and save time and resources for system administrators.
- **Improve accuracy of Anomaly detection:** Our model employs a combination of advanced techniques and algorithms specifically designed for anomaly detection in log data which helps in improving the accuracy of detecting anomalies in the log files.

CHAPTER 5
SYSTEM DESIGN ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

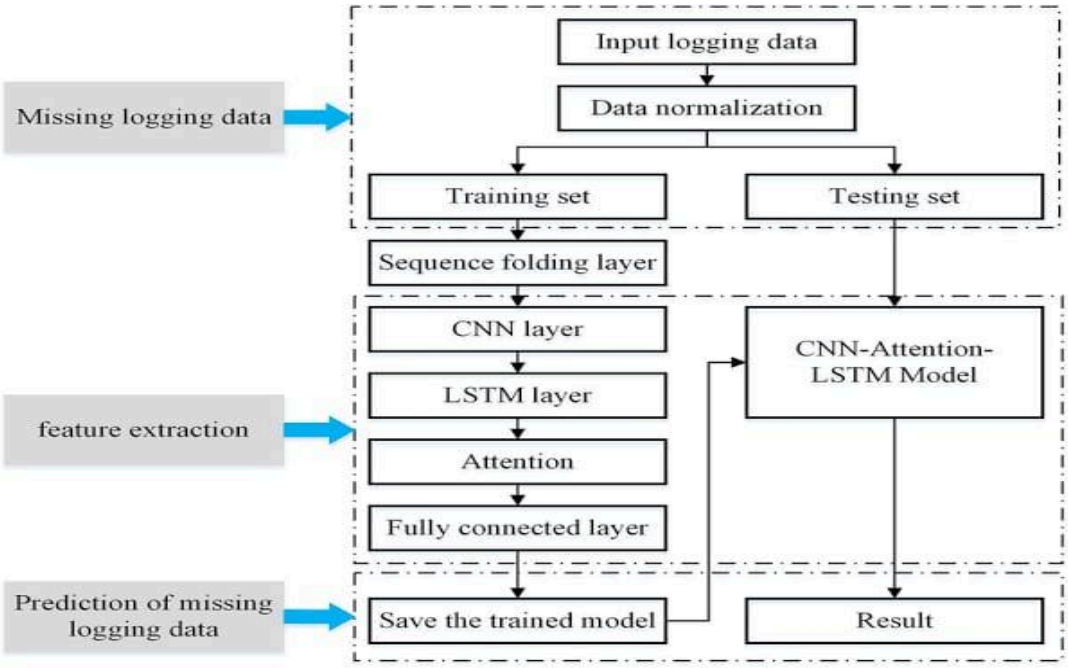


Fig 5.1: System Architecture

The process of log data analysis starts with input logging data, which collects log data from various activities or events in the system. However, the dataset may suffer from incomplete log entries, introducing the need for careful handling to ensure inclusive analysis.

To address this, data normalization is applied to standardize the numerical values within the dataset, optimizing the dataset for training. The next step includes dividing the dataset into two subsets: A training set and a testing set as shown in Figure 5.1. The training set is used to train the model, meanwhile, the testing set is used to evaluate the performance of the model. After the dataset is divided, the sequence folding layer is applied to reformat the input sequences. This layer helps to prepare the data for further processing and analysis.

The architecture implemented in log data analysis combines both CNN and LSTM algorithms. This combination of algorithms enables the model to capture temporal dependencies and patterns present in the log data. The CNN component helps extract local features from the data, while the LSTM component captures long-term dependencies within sequential data. Then it incorporates a CNN-LSTM-Attention mechanism, which focuses only on the important parts of the input sequence, enhancing the ability of the model to discern crucial information.

Finally, feature extraction plays a crucial role in the log analysis process. During this step, relevant information is identified and highlighted. The extracted features are linked to a fully connected layer, which prepares them for the prediction stage where the system predicts the missing logging data. Once the model completes training and evaluation, it is then saved for future use. The model's performance is based on the results obtained from the testing set. This evaluation ensures the accuracy and effectiveness of the model in predicting the missing logging data.

5.2 USE CASE DIAGRAM

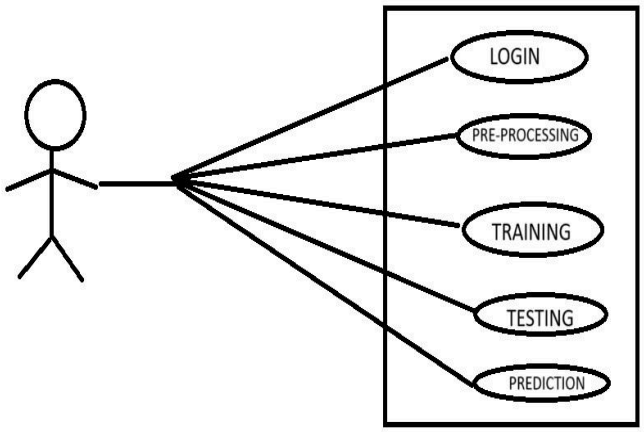


Fig 5.2: User-level functions

5.3 Level 0 DFD

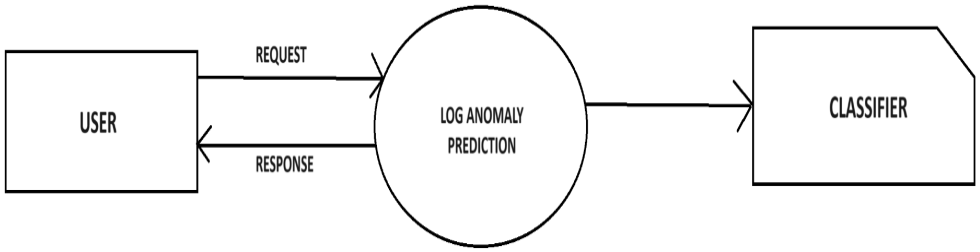


Fig 5.3 : Level 0 DFD

5.4 Level 1 DFD

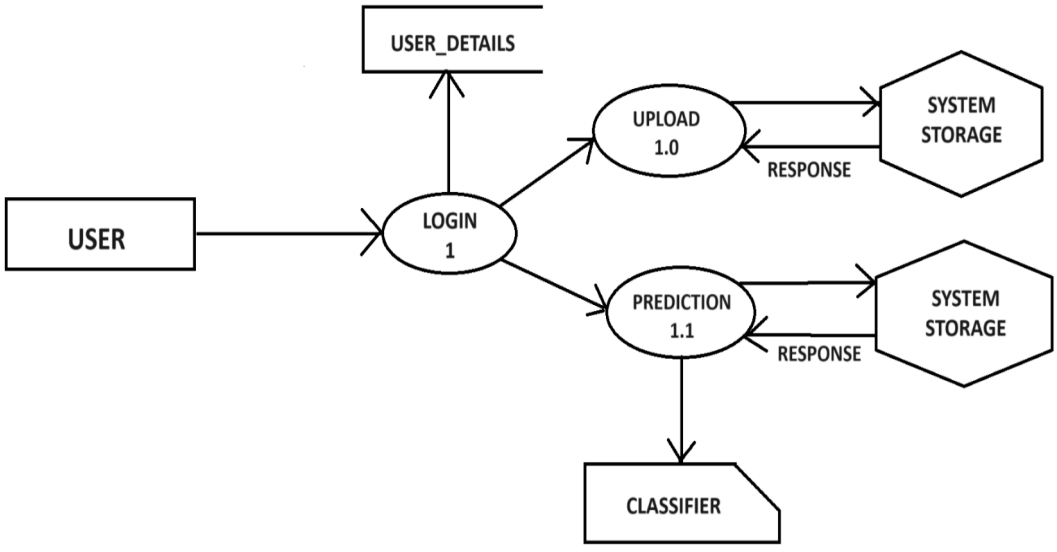


Fig 5.4 : Level 1 DFD

5.5 Level 2 DFD

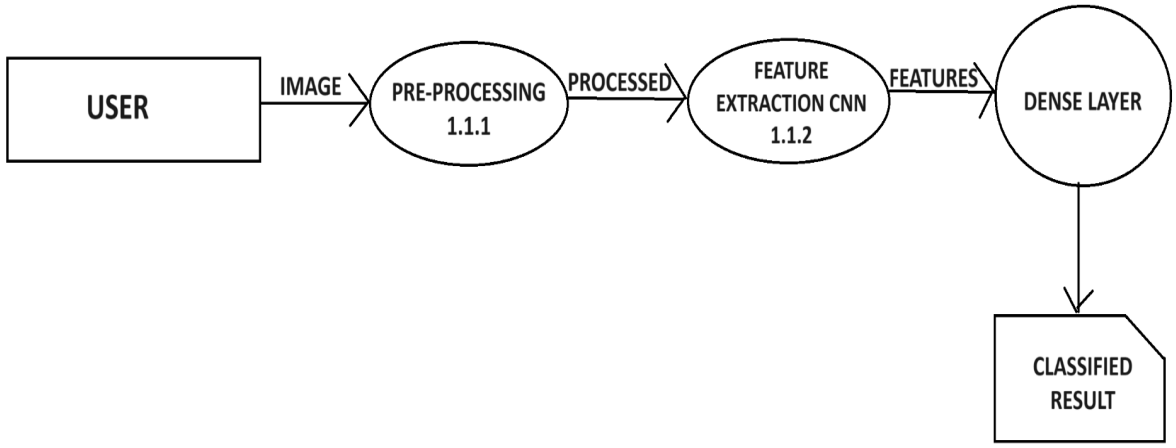


Fig 5.5 : Level 2 DFD

CHAPTER 6
SYSTEM REQUIREMENTS

6.1 SOFTWARE REQUIREMENTS:

i. Front end: HTML

ii. Back end:

- IDE: Google Colab
- Back-end Database: MySQL
- Back-end Scripting: Python
- Back-end Server: Flask

iii. Operating System:

- Client: Windows 10,11

6.2 HARDWARE REQUIREMENTS:

i. System:

- Windows 10 or 11 - PC/Laptop
- RAM: 4GB
- System-type:64-bit
- Processor: Intel Core i3-4100M or above
- Network Connectivity

CHAPTER 7
MODULE DESCRIPTION

The Log Based Anomaly Detection System consists of 5 modules:

1. Dataset Collection
2. Preprocessing and Feature Extraction
3. Model Training
4. Model Testing
5. Evaluation and Prediction

7.1 Module 1: Dataset Collection

A synthetic dataset of log data is being generated and used for further analysis. A log data consisting of a specified number of entries is generated. The dataset creation involves two main stages: the generation of normal log entries and the introduction of anomalies. Normal log entries are randomly generated with typical values and behaviors, while a subset of entries is marked as anomalies. Each log entry in the generated dataset consists of various attributes such as timestamp, source IP, event type, user agent, status, value, and duration. These anomalies are introduced by altering certain attributes such as the 'Value' and 'Event' columns, simulating abnormal behavior within the log data.

The resulting dataset, named 'log_data', combines both normal and anomalous log entries. It has a newly added 'Target' column that distinguishes anomalies (marked as 1) from normal entries (designated as 0). This dataset is intended for use in training anomaly detection models to identify and classify abnormal log entries within a system or network environment.

7.2 Module 2: Preprocessing and Feature Extraction

In this step, log data is organized and transformed into a suitable format for analysis. It is mainly done to handle missing data, standardize scales, address outliers, and generally ensure that the data is in a form that is conducive to accurate and effective analysis. Here, the algorithms used to train the required model require numerical inputs rather than categorical labels. The log data is divided into two distinct sets: numerical features, comprising attributes like 'Value' and 'Duration', and categorical features, encompassing 'SourceIP', 'Event', 'UserAgent', and 'Status'.

StandardScaler object is initialized, which will be used to standardize (or scale) numerical features in the dataset to have a mean of 0 and a standard deviation of 1. A OneHotEncoder object is initialized with sparse=False, which will be used to one-hot encode categorical features of the data. Categorical features are transformed into binary vectors through one-hot encoding, effectively converting them into a numerical format suitable for deep learning algorithms. The transformed categorical features are combined with the original data frame, and the original categorical columns are dropped to prevent redundancy. Finally, the preprocessed data is exported to a CSV file, facilitating further analysis in deep learning models. This preprocessing pipeline ensures that the data is appropriately formatted and scaled for effective model training and prediction.

Feature Selection: The log data is prepared by first removing certain columns, namely 'Timestamp' and 'Target', from the dataset.

Data Splitting: The dataset is then divided into training and testing subsets. One set is used for training the model and the other for evaluating its performance. Here, the test size is set to 0.2, indicating an 80-20 split and a random state of 42 is specified for reproducibility.

Reshaping for LSTM Input: The input data is reshaped into a 3D array, where the dimensions correspond to the number of samples, time steps (or features), and input dimensions. This reshaping prepares the data to be compatible with the input requirements of the CNN model, which expects input data in this specific format for sequential processing. This is necessary for consistency when giving data into the deep learning models. This format allows the model to understand and process the data properly during evaluation.

7.3 Module 3: Model Training

Training is the most important step in deep learning. A hybrid approach which combines CNN and LSTM models, is used to train the model using the preprocessed data in this module. This model consists of a convolutional layer for feature extraction, followed by an LSTM layer for sequence processing, and finally, a Dense layer for binary classification. The model is trained using the training data over 50 epochs, with a batch size of 32, and evaluated using the validation data. Throughout the training process, the model iteratively adjusts its parameters to minimize the defined loss function, aiming to optimize its predictive accuracy. This training regimen

enables the model to learn meaningful representations from the sequential input data and make accurate predictions based on the acquired knowledge of temporal patterns and relationships within the dataset.

7.4 Module 4: Model Testing

The trained model is tested using the testing set. The variable y_pred_probs contains the predicted probabilities for each instance in your test data test data (X_test) using a deep learning model (model) predicts the probabilities and this model also gives a probability score for each class. It is done when dealing with binary classification problems to convert probabilities into predicted classes (0 or 1).

The confusion matrix is a table that shows how many of the actual class labels were correctly or incorrectly predicted by the model. It is a way to evaluate the model's performance. The classification report provides a detailed summary of different performance metrics such as precision, recall, and F1-score for each class, as well as an average across all classes.

7.5 Module 5: Evaluation and Prediction

In this last phase,the model uses its learned patterns from the training data to predict class labels for the new data points. A user interface for real-time prediction is developed for the prediction of the model. Flask is a Python framework that allows you to build web applications. A web page is created using Flask where users can upload new log data. The uploaded log files are analyzed to check the presence of anomalies. Here,the Flask application acts as a bridge between the anomaly detection model and a user-friendly interface.

UI is the part that users interact with directly, typically a web browser.Users can upload log files through a web browser.The browser sends an HTTP request to the server.The Flask application on the server receives the uploaded log files from the client and then processes the data.It is the backend application that runs on a web server and handles data processing and communication.It is built using Flask and also interacts with other Python libraries.The server sends the prediction results from the anomaly detection model back to the browser as a response.The server sends the prediction results from the anomaly detection model back to the browser as a response.It may

also interact with a database to store historical data or logs for further analysis. The browser updates the UI based on the received response and thus displays the required result..

Synthetic data is generated for normal logs, Each log entry includes information like timestamp, user agent, source IP, value, duration, etc. These values have typical behavior. After creating synthetic data , you shuffle the dataset to introduce randomness. At the end, you add a binary target variable named 'Target.' For each log entry, if the event type is marked as 'Anomaly,' the target is set to 1, indicating an anomaly. Otherwise, it is set to 0, indicating a normal log entry. Finally, you save this synthetic test dataset to a CSV file named 'test_log_data.csv.' This file can be used to test and evaluate the performance of a machine-learning model.

Firstly, you assume that you have a previously trained model stored as a 'model.' Then,the test dataset is loaded. After preprocessing, you reshape the input data to fit the requirements of the LSTM model. The reshaped data is prepared to be fed into the trained model to predict probabilities for each log entry in the dataset. Finally, it is checked whether any anomalies are detected in the test dataset by examining the predicted classes. If the value 1 is present in the predicted classes, it indicates the presence of anomalies, and the code prints "The given file contains anomalies." Otherwise, it prints "The given file does not contain anomalies."

CHAPTER 8
IMPLEMENTATION

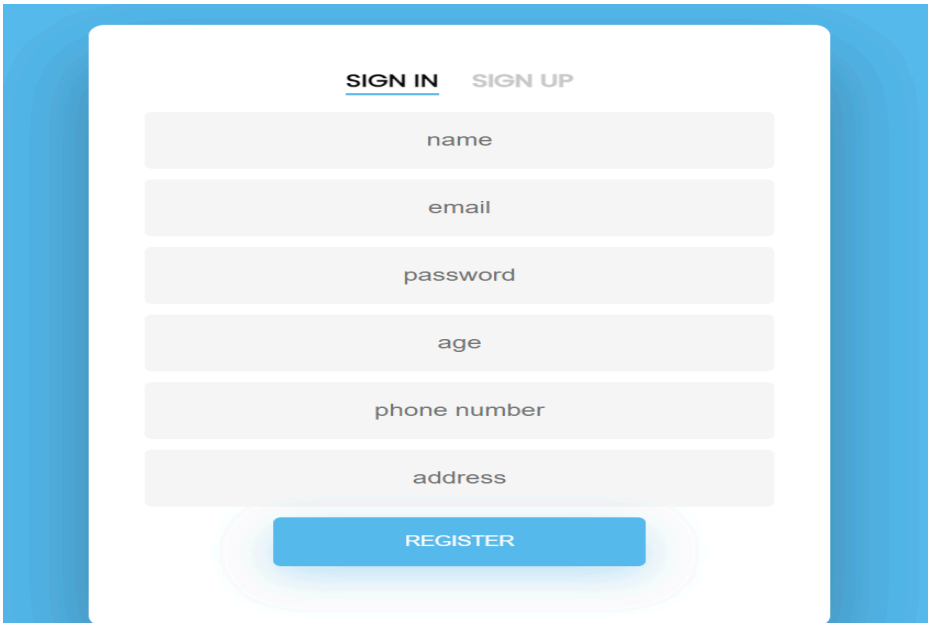
Log-based Anomaly Detection uses a hybrid deep learning model that combines CNN with LSTM to distinguish between anomalous and non-anomalous data in log files.

After the creation of the dataset, the phase of preprocessing and feature extraction is implemented to organize the data and then transform it into a suitable form for further analysis. The data is also divided into training as well as testing sets. Now, the data is trained by combining CNN with the LSTM algorithm. The evaluation of the trained model and the prediction phase is the last stage of this system.

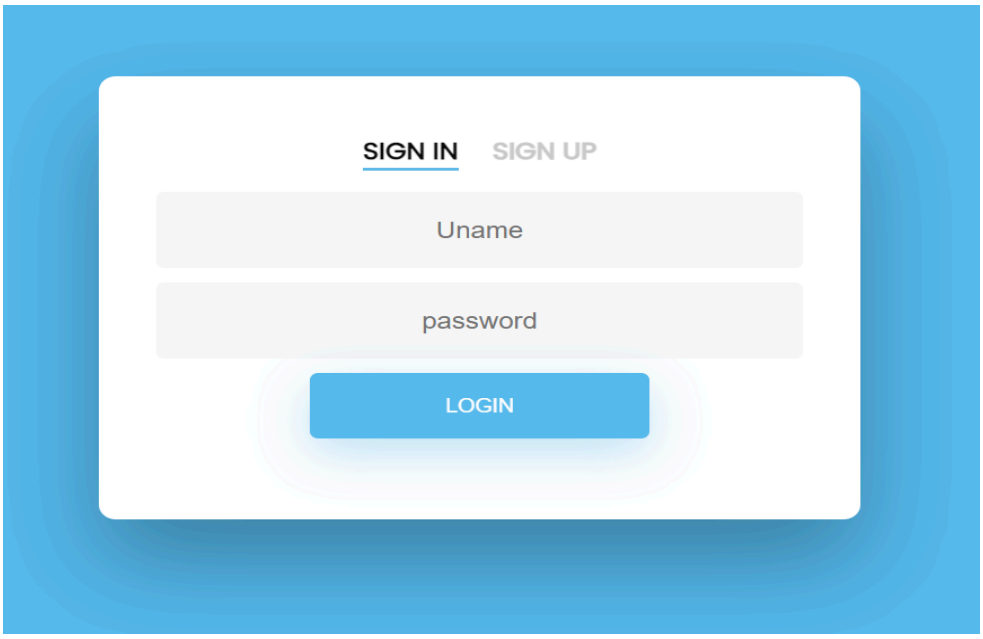
In the last phase, a portal for predicting the anomalies in log files is introduced. The user can register to enter the log anomaly detection website by adding various details like email, username, password, age, email ID, phone number, and address. On the login page, The user has to enter the username and password. If the login is successful, the screen displays a home page of the anomaly detection website. Click the option 'Detection' to upload a log file. To upload a log file, which is in the form of CSV, the user can click on the choose file option and then browse the file from its local storage. After selecting the file, click on the predict button. Now, the log file is uploaded to the anomaly detection website. The result is displayed under the heading of 'Output'. If the file contains anomalies, it will display that 'the given file contains anomalies'. Thus the user can get a prediction on whether the uploaded log file contains anomalies or not. There is also an option to download the normal as and anomalous files.

STEPS TO USE THE SYSTEM

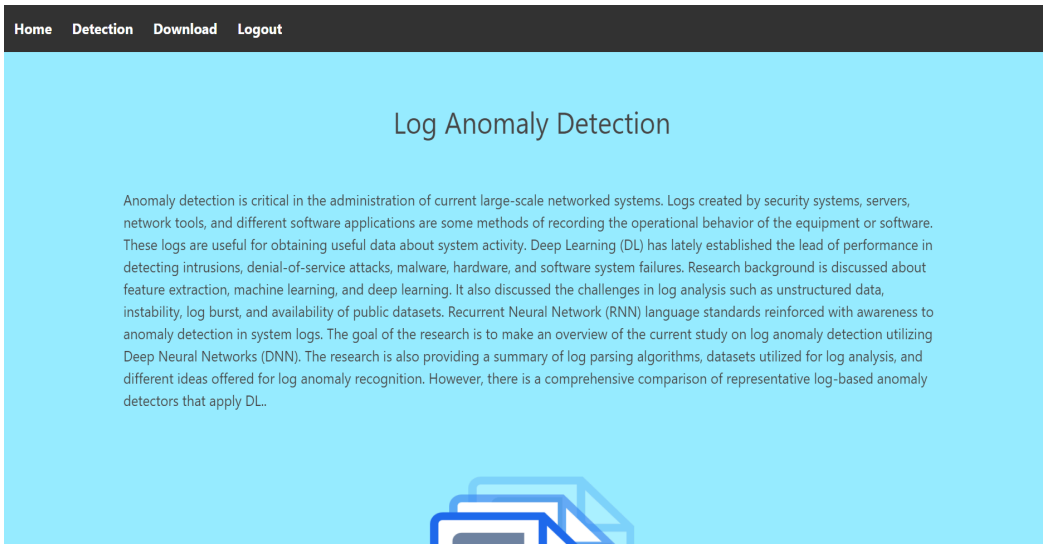
1. First, a registration page is displayed for new users. Here user enters the required details for further analysis:



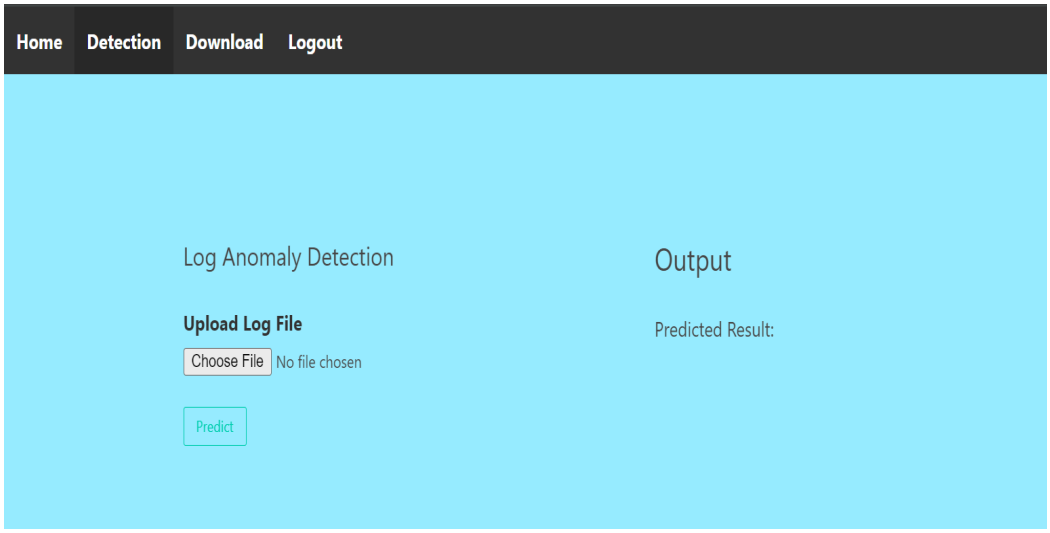
2. The login page has the option to log into the portal by entering a username and password.



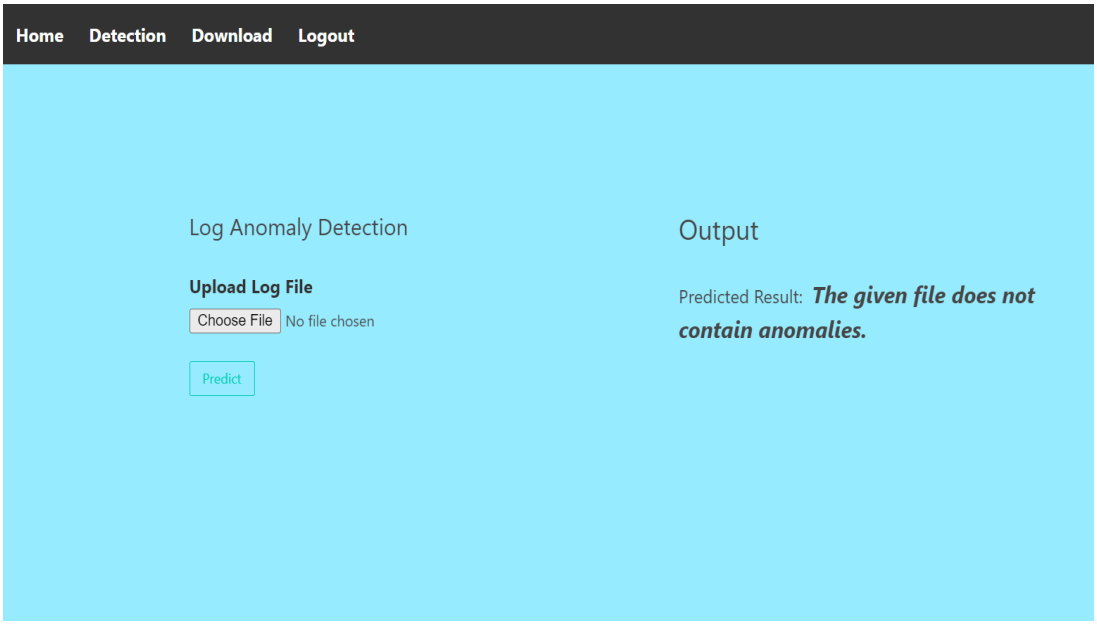
3. Once the user is logged into the anomaly detection website, a home page is displayed on the screen.



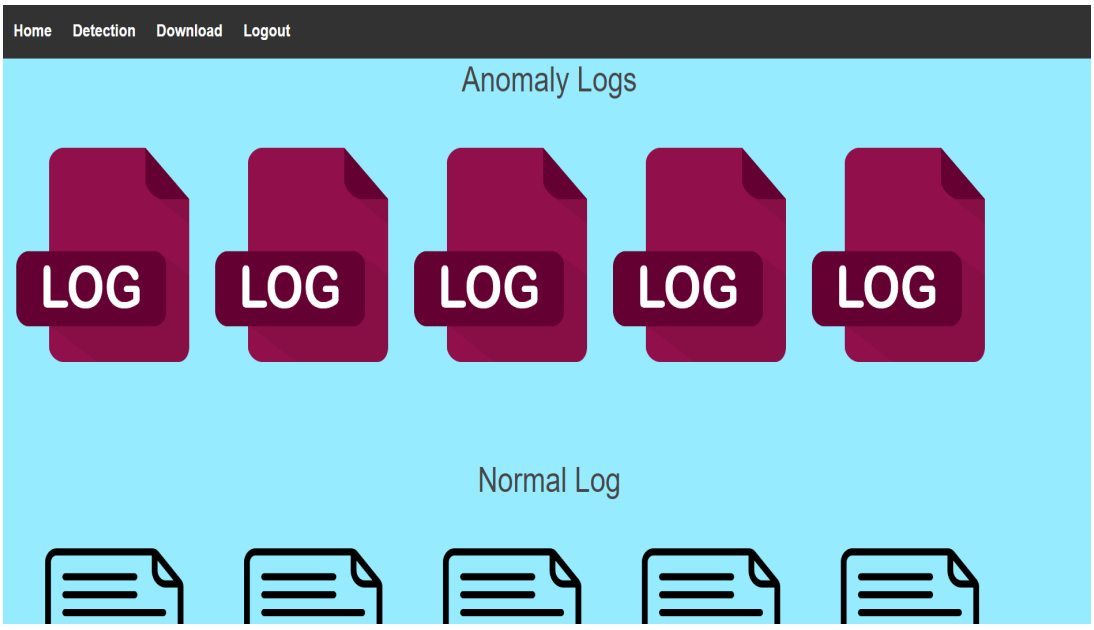
4. In the next step, the users can check log anomalies by uploading log files through the ‘Choose File’ option.



5. The log file is uploaded from the system's local storage. The accepted log files are in the CSV format.



6. There is also an option to download the anomaly and normal logs from the 'Download' option.



,

CHAPTER 9
RESULT ANALYSIS

9.1 Accuracy and Loss

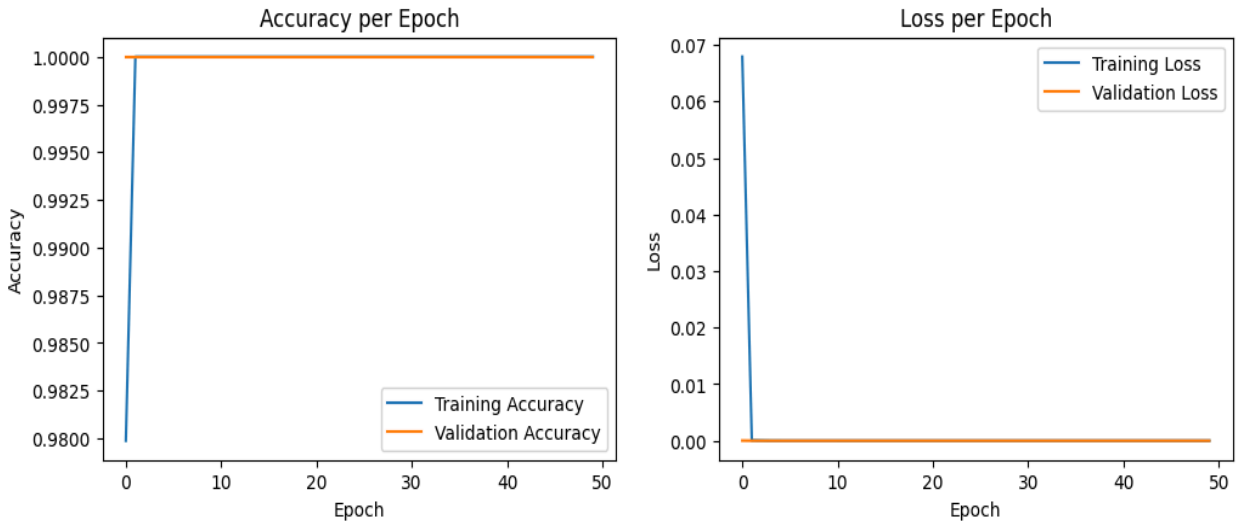


Fig 9.1: Accuracy and Loss Graph

- **Accuracy per Epoch:** The left graph shows the training accuracy and the validation accuracy over each epoch. It helps in visualizing how well the model is learning from the training data and generalizing to unseen validation data. The x-axis represents the epochs, while the y-axis represents the accuracy values. It is observed that the accuracy of the model improves as it learns from the training data over successive epochs. Initially, the training accuracy might start low and then increase steadily as the model learns patterns in the data.
- **Loss per Epoch:** The right subplot plots the training loss and the validation loss over each epoch. The x-axis represents the epochs, while the y-axis represents the loss values. Loss is a way of measuring how wrong the model's predictions are on average. The goal of this model is to minimize loss. A lower loss means the model's predictions are closer to the correct answers. In this graph,the loss starts high and trends downward as the number of epochs increases. This shows that the model is learning from the training data.

9.2 Confusion Matrix

The confusion matrix provided is a 2x2 matrix that summarizes the performance of a binary classification model.

TP (True Positive) is the number of abnormal log sequences that are correctly detected by the model. FP (False Positive) is the number of normal log sequences that are wrongly identified as anomalies. FN (False Negative) is the number of abnormal log sequences that are not detected by the model

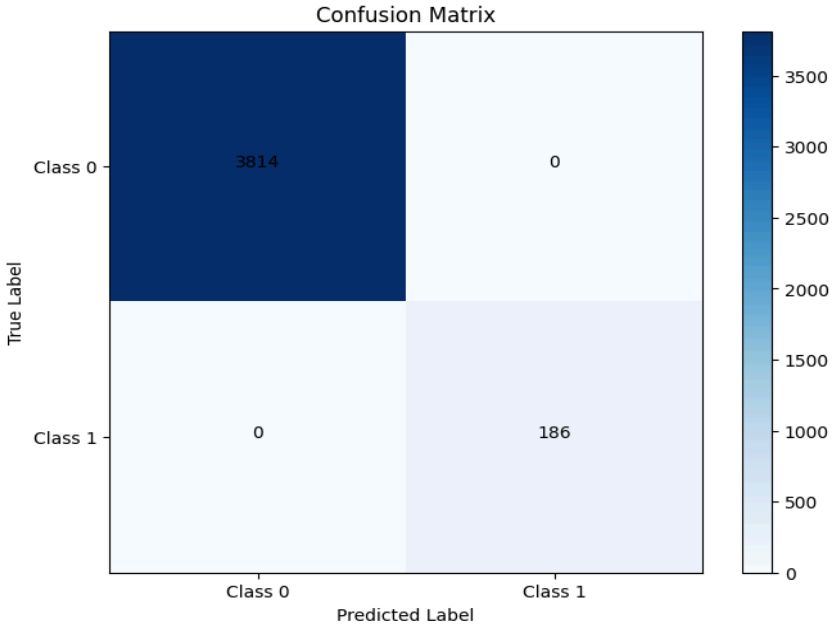


Fig 9.2: Confusion Matrix

In this case:

True Negatives (TN): 3814 instances were correctly predicted as the negative class (class 0).

False Positives (FP): 0 instances were incorrectly predicted as the positive class (class 1).

False Negatives (FN): 0 instances were incorrectly predicted as the negative class.

True Positives (TP): 186 instances were correctly predicted as the positive class.

The confusion matrix of this model indicates excellent performance, where the model made no errors in predicting negative instances and correctly identified all positive instances. Such a result suggests that the model has high precision and recall, indicating a strong ability to discriminate between the two classes.

9.3 Classification Report

	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	3814
1	1.00	1.00	1,00	186
Accuracy			1.00	4000

Fig 9.3 : Classification Report

- Precision: Precision measures the accuracy of positive predictions. In this case, both classes (0 and 1) have a precision of 1.00, indicating that all instances predicted as positive (1) are indeed positive. Similarly, for class 0, all instances predicted as negative (0) are correct.
- Recall: Both classes have a recall of 1.00 that means all positive instances were correctly identified.
- F1-score: The F1-score provides a balance between precision and recall. Since both precision and recall are 1.00 for both classes, the F1-score is also 1.00 for both.
- Support: Support is the number of actual occurrences of the class in the specified dataset. Here, it indicates how many instances belong to each class.
- Accuracy: Accuracy measures the overall correctness of the model across all classes. With an accuracy of 1.00, it means all predictions made by the model are correct.

This report suggests that the model has achieved an F1-score of 100 ,accurately predicting both classes without any errors.

CHAPTER 10
CONCLUSION

In this paper, we present a comprehensive review and evaluation of log anomaly detection methods that leverage a hybrid deep learning model. The proposed methodology combines the strength of two deep learning algorithms with other techniques to enhance the accuracy and efficiency of log analysis. To begin, we collect and preprocess a large dataset of log records from various sources in the system. This dataset is used to train the hybrid model, which is designed to learn the normal behavior and patterns of the system by analyzing the log data.

During the training process, the hybrid model uses a combination of supervised and unsupervised learning techniques. Unsupervised learning helps to detect anomalies in the log data, while supervised learning helps to label and classify the detected anomalies. Once the model was trained, we tested the model on the new log datasets to evaluate its accuracy and efficiency. We also measured the precision, recall, f1-score, etc to assess the performance of the model in detecting anomalies.

The result of the evaluation shows the effectiveness of the hybrid deep learning model in detecting anomalies from the log data. It exhibits higher accuracy as compared to the traditional manual inspection method, reducing the need for humans to manually analyze logs. The model also shows computational performance, making it suitable for monitoring and detecting anomalies in real time, especially in large-scale distributed systems. Overall, the model is accurate, effective, reduces human efforts, and can be used for real-time monitoring.

REFERENCES

[1] Le, Van-Hoang, and Hongyu Zhang. "Log-based anomaly detection with deep learning: How far are we?." In *Proceedings of the 44th international conference on software engineering*, pp. 1356-1367. 2022.

[2] Landauer, Max, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. "Deep learning for anomaly detection in log data: A survey." *Machine Learning with Applications* 12 (2023): 100470.

[3] Zhao, Zhijun, Chen Xu, and Bo Li. "A lstm-based anomaly detection model for log analysis." *Journal of Signal Processing Systems* 93, no. 7 (2021): 745-751.

[4] Zeufack, Vannel, Donghyun Kim, Daehee Seo, and Ahyoung Lee. "An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis." *High-Confidence Computing* 1, no. 2 (2021): 100030.

[5] Shahid, Nimra, and M. Ali Shah. "Anomaly detection in system logs in the sphere of digital economy." (2021): 185-190.

[6] He, Shilin, Jieming Zhu, Pinjia He, and Michael R. Lyu. "Experience report: System log analysis for anomaly detection." In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pp. 207-218. IEEE, 2016.

[7] Yin, Kun, Meng Yan, Ling Xu, Zhou Xu, Zhao Li, Dan Yang, and Xiaohong Zhang. "Improving log-based anomaly detection with component-aware analysis." In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 667-671. IEEE, 2020.

[8] Lin, Peng, Kejiang Ye, and Cheng-Zhong Xu. "Dynamic network anomaly detection system by using deep learning techniques." In *Cloud Computing–CLOUD 2019: 12th International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 12*, pp. 161-176. Springer International Publishing, 2019.

APPENDIX

SOURCE CODE:

#1

```
from flaskext.mysql import MySQL
from flask import (Flask, request, session, g, redirect, url_for, abort, render_template, flash, Response)
from werkzeug.utils import secure_filename
import os
import classifier
import re
regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

def check(email):

    if(re.fullmatch(regex, email)):
        return True
    else:
        return False

mysql = MySQL()
app = Flask(__name__,static_folder='static')
app.config['MYSQL_DATABASE_USER'] = 'root'
app.config['MYSQL_DATABASE_PASSWORD'] = ''
app.config['MYSQL_DATABASE_DB'] = 'log'
app.config['MYSQL_DATABASE_HOST'] = 'localhost'
mysql.init_app(app)
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = './static/upload/'
app.config['SECRET_KEY'] = 'd3Y5d5nJkU6CdwY'
if os.path.exists(app.config['UPLOAD_FOLDER']):
```

```
    print("directory exists")
else:
    os.makedirs(app.config['UPLOAD_FOLDER'])
    print("directory created")


@app.route('/', methods=["GET", "POST"])
def index():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        cursor = mysql.connect().cursor()
        cursor.execute("SELECT * from user_details where email='" + email + "' and password='"
+ password + "'")
        print("SELECT * from user_details where email='" + email + "' and password='" +
password + "'")
        data = cursor.fetchone()
        if data is None:
            return "Username or Password is wrong"
        else:
            session['email'] = data[2]
            session['user_id'] = data[0]
            return redirect(url_for('home'))

        return render_template('index.html')

    else:
        return render_template('index.html')

@app.route('/addUser',methods = ["GET","POST"])
def addUser():
    if request.method == 'POST':
        User_name = request.form['user_name']
        email = request.form['email']
```

```
        Phone_number = request.form['phone_number']
        age = request.form['age']
        Password = request.form['password']
        address = request.form['address']
        res=check(email)
        if User_name=="" or email=="" or phone_number=="" or age=="" or password=="" or
address=="":
            return "Please Fill all fileds"
        if not res:
            return "Please Enter a valid Email"
        if len(str(Phone_number)):
            return "Please Enter a valid Phone Number"
        qry = " INSERT INTO `user_details` ( user_name,email, Password, age,
phone_number,address ) values "
        qry += "('"+User_name +"','"+email +"','"+Password +"','"+age +"','"+Phone_number
+"','"+address +"')"
        conn = mysql.connect()
        cursor = conn.cursor()
        cursor.execute(qry)
        conn.commit()
        return redirect(url_for('index'))
    return render_template('addUser.html')

@app.route("/detect", methods=["GET", "POST"])
def detect():
    result = ""
    if request.method == "POST":
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
```

```
        # if user does not select file, browser also
        # submit an empty part without filename
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        filename = file.filename
        print(filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        result = classifier.classify(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    return render_template("detect.html", result=result, )
#
@app.route("/home",methods = ["GET","POST"])
def home():
    return render_template('home.html')
@app.route("/download",methods = ["GET","POST"])
def download():
    return render_template('images.html')

if __name__ == "__main__":  # on running python app.py
    app.run(debug=True)  # run the flask app
```

#2

```
import pandas as pd
import numpy as np
np.random.seed(42)
dataset_length = 10000

# Generate synthetic data for normal logs
normal_logs = pd.DataFrame({  'Timestamp': pd.date_range(start='2024-01-01',
periods=dataset_length,  freq='H'),   'SourceIP': np.random .choice(['192.168.1.1', '192.168.1.2',
'192.168.1.3'],  size=dataset_length),  'Event':  np.random.choice(['Login',  'Access',  'Error'],
size=dataset_length),  'UserAgent':  np.random.choice(['Chrome',  'Firefox',  'Safari',  'Edge'],
```

```
size=dataset_length), 'Status': np.random.choice(['Success', 'Failure'], size=dataset_length), 'Value': np.random.normal(loc=50, scale=10, size=dataset_length), 'Duration': np.random.uniform(low=1, high=10, size=dataset_length) })

# Introduce anomalies in the dataset
anomaly_indices =np.random.choice(dataset_length,size=int(0.1*dataset_length), replace=False)
anomaly_logs = normal_logs.copy()
anomaly_logs.loc[anomaly_indices,'Value'] = np.random.normal(loc=200, scale=50, size=len(anomaly_indices))
anomaly_logs.loc[anomaly_indices, 'Event'] = 'Anomaly'
anomaly_logs.loc[anomaly_indices, 'Duration'] = np.random.uniform(low=15, high=25, size=len(anomaly_indices))

log_data = pd.concat([normal_logs, anomaly_logs], ignore_index=True)
log_data = log_data.sample(frac=1, random_state=42).reset_index(drop=True)
log_data['Target'] = np.where(log_data['Event'] == 'Anomaly', 1, 0)
print(log_data.head())

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from keras.models import Sequential
from keras.layers import Conv1D, LSTM, Dense, Flatten, Dropout
from keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix

# Preprocess the data
scaler = StandardScaler()
encoder = OneHotEncoder(sparse=False)
numerical_features = ['Value', 'Duration']
categorical_features = ['SourceIP', 'Event', 'UserAgent', 'Status']
log_data[numerical_features] = scaler.fit_transform(log_data[numerical_features])
```

```
encoded_categories = encoder.fit_transform(log_data[categorical_features])
encoded_df = pd.DataFrame(encoded_categories,
columns=encoder.get_feature_names_out(categorical_features))
log_data = pd.concat([log_data, encoded_df], axis=1)
log_data.drop(categorical_features, axis=1, inplace=True)
print(log_data)
log_data.to_csv('feature_log_data.csv', index=False)


# Split into features and target
X = log_data.drop(['Timestamp', 'Target'], axis=1)
y = log_data['Target']
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train)
# Reshape input for LSTM
X_train = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
print(X_train)
# Build the model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1],
1)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,y_test))
y_pred_probs = model.predict(X_test)
y_pred_classes = np.round(y_pred_probs)
print(confusion_matrix(y_test, y_pred_classes))
print(classification_report(y_test, y_pred_classes))
```