A

REPORT ON MAJOR PROJECT

On

ANOMALY DETECTION THROUGH BEHAVIOR ANALYSIS USING K MEANS CLUSTERING

Dissertation Submitted in the Partial Fulfilment of Academic Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

By

ERUMANDLA LOKESH REDDY	**(21B61A6642)**
BOBBALA VENKATA NAGIREDDY	**(22B65A6602)**
ARELLY HIMESH GOUD	**(21B61A6614)**
BHUKYA SANDEEP KUMAR	**(21B61A6626)**

Under the guidance of

Dr. SUNIL TEKALE

Professor and HOD of CSE(AIML)

NALLA MALLA REDDY ENGINEERING COLLEGE
An Autonomous Institution
(Approved by AICTE, New Delhi and Affiliated to JNTUH)
DIVYA NAGAR, MALKAJGIRI-MEDCHAL (DIST), HYDERABAD-500088
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE MACHINE LEARNING)

2024-2025

Nalla Malla Reddy Engineering College

An Autonomous Institution

Divya Nagar, Kachavani Singaram Post, Ghatkesar (M), Medchal (Dist)-500088

Phones: 08415-256001.02.03. Fax: 08415-256000

Email: info@nmrec.edu.in Website: www.nmrec.edu.in

CERTIFICATE

This is to certify that the project report entitled **"ANOMALY DETECTION THROUGH BEHAVIOR ANALYSIS USING K MEANS CLUSTERING"** is being submitted by ERUMANDLA LOKESH REDDY (21B61A6642), BOBBALA VENKATA NAGIREDDY (22B65A6602),ARELLY HIMESH GOUD (21B61A6614),BHUKYA SANDEEP KUMAR (21B61A6626) and in partial fulfillment of the academic requirements for the award of degree of Bachelor of Technology in Computer Science Engineering (Artificial Intelligence & Machine Learning)**, Nalla Malla Reddy Engineering College, AUTONOMOUS Hyderabad** during the academic year 2024- 2025.

Internal Guide	Project Coordinator	Head of the Department
Dr. SUNIL TEKALE	**Dr. BATTULA BALNARSAIAH**	**Dr. SUNIL TEKALE**
Professor	Associate Professor	Professor
Department of AIML	Department of AIML	Department of AIML

Principal

External Examiner **Dr M. N. V. Ramesh**

CANDIDATE'S DECLARATION

We hereby declare that the project report entitled **"ANOMALY DETECTION THROUGH BEHAVIOR ANALYSIS USING K MEANS CLUSTERING"** is the bona-fide work done and submitted by us under the guidance of Dr. B.Balnarsaiah, in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science Engineering(Artificial Intelligence & Machine Learning) to the department of Computer Science Engineering(Artificial Intelligence & Machine Learning), Nalla Malla Reddy Engineering College, Divya Nagar, Medchal-Malkajgiri Dist.

Further we declare that the report has not been submitted by anyone to any other institute or university for the award of any other degree.

DATE:	**E. LOKESH REDDY**	**-(21B61A6642)**
PLACE: HYDERABAD	**B.VENKATA NAGIREDDY**	**-(22B65A6602)**
	A. HIMESH GOUD	**-(21B61A6614)**
	B.SANDEEP KUMAR	**-(21B61A6626)**

ACKNOWLEDGEMENT

The completion of the project gives us an opportunity to convey our gratitude to all those who helped us to reach a stage where we have the confidence to launch our career in the competitive world.

We express our sincere thanks to **Dr. Divya Nalla, Director,** Nalla Malla Reddy Engineering College, Divya Nagar, for providing all necessary facilities to complete our project.

We express our sincere thanks to **Dr. M. N. V. Ramesh**, Principal, Nalla Malla Reddy Engineering College, Divya Nagar, for providing all necessary facilities to complete our project.

We express our sincere gratitude to **Dr. Sunil Tekale,** Head of the Department of Computer Science Engineering (Artificial Intelligence & Machine Learning), Nalla Malla Reddy Engineering College, Divya Nagar, for providing necessary facilities to complete our project successfully.

We express our sincere thanks to our project coordinator **Dr. Battula Balnarsaiah,** Associate Professor and guide **Dr. Sunil Tekale**, Professor of Department of Computer Science Engineering (Artificial Intelligence & Machine Learning), Nalla Malla Reddy Engineering College, Divya Nagar, for their valuable guidance, encouragement, and co-operation throughout the project.

Finally, we would like to thank our parents and friends for their continuous encouragement and valuable support to us.

ABSTRACT

In today’s digital landscape, safeguarding sensitive information and infrastructure from insider threats is critical for organizational security. This project, titled "Anomaly Detection through Behavior Analysis," aims to detect insider attacks by analyzing employee behavior patterns within the corporate environment. The approach leverages unsupervised machine learning techniques, specifically K-means clustering, to group data based on behavioral similarities and establish baseline patterns.

To capture sequential dependencies in user actions, Hidden Markov Models (HMM) are employed, enabling the identification of anomalous activity based on deviation from expected behavioral sequences. Furthermore, to enhance anomaly detection precision, the Local Outlier Factor (LOF) algorithm is integrated, allowing the identification of localized anomalies that may go unnoticed at a global level. By combining these techniques, the model is designed to provide a robust multi- layered detection system that accurately flags potentially malicious activities, improving response times to insider threats and contributing to organizational resilience.

CONTENTS

List of Figures

List of Abbreviations

Abbreviation	Description
CERT	Computer Emergency Response Team
CERT-In	Indian Computer Emergency Response Team
DL	Deep Learning
ECA	Efficient Channel Attention
GPU	Graphical Processing Unit
HMM	Hidden Markovnikov Model
LOF	Local Outlier Factor
ML	Machine Learning
Res-Net	Residual Networks
TTT	Test Time Training

CHAPTER-1

INTRODUCTION

1.1 Overview of the Project

As organizations increasingly rely on digital infrastructure, the threat of insider attacks—where individuals with legitimate access misuse their privileges— has become a critical security concern. Unlike external threats, insider attacks often go unnoticed by traditional security measures, as they blend into routine activities. With sensitive data at stake, proactive detection of anomalous behavior is essential to mitigate risks posed by such threats. This project, ″Anomaly Detection through Behavior Analysis,″ addresses this need by applying advanced machine learning techniques to analyze user behavior patterns and detect deviations that could signal insider threats. By focusing on daily interactions and behavior across the organization, the project aims to identify unusual activities that fall outside typical usage patterns, offering a proactive solution for insider threat detection. The detection framework leverages a combination of K-means clustering, Hidden Markov Models (HMM), and the Local Outlier Factor (LOF) algorithm to create a robust, multi-layered approach to security. K-means clustering first establishes behavioral baselines by grouping users with similar patterns, setting a reference for normal activity. HMM then analyzes sequences of actions, capturing temporal dependencies to detect deviations from these established norms.

Anomaly Detection through behaviour Analysis refers to the process of identifying patterns, behaviours, or activities in data that deviate from established norms. Unlike rule-based systems that rely on fixed thresholds, behaviour analysis builds a dynamic profile of normal activity over time—such as login times, transaction patterns, network activity, or sensor readings—and detects when new behaviours significantly diverge from these baselines. This approach is especially valuable in domains where behaviour is continuously evolving, such as cybersecurity, fraud detection, healthcare, and IoT systems. For instance, in cybersecurity, behaviour analysis can flag unusual login times or access from unfamiliar locations as potential intrusions. In fraud detection, it can identify rapid, high-value transactions that are inconsistent with a customer’s typical spending habits. The techniques used in this form of anomaly detection range from statistical models and clustering algorithms like K-Means, to more advanced machine learning methods including supervised, unsupervised, and deep learning models such as autoencoders and LSTM networks. These methods analyze both historical and real-time data to find outliers that may indicate misuse, threats, or system faults. The main advantages of behavioural anomaly detection include its adaptability, context awareness, and ability to detect unknown or subtle threats. However, it also faces challenges such as false positives, difficulty in establishing accurate behavioural baselines, and data privacy concerns. Overall, anomaly detection through behaviour analysis provides a robust, intelligent approach to identifying unexpected or harmful activity across a wide range of applications.

1.2 Problem Statement

The motivation behind this project stems from the growing need to protect organizations from the unique and often underestimated risk of insider threats. Unlike external cyberattacks, insider threats are challenging to detect as they come from trusted individuals who already have access to sensitive data and resources. With the increasing complexity of workplace environments, traditional security measures often fail to recognize subtle, suspicious behaviors that signal potential misuse of access.

In today's data-driven environments, organizations across various domains—such as cybersecurity, finance, healthcare, and IoT—face increasing challenges in identifying unusual or malicious activities hidden within massive volumes of behavioural data. Traditional rule-based systems are often rigid, unable to adapt to new attack patterns or evolving user behaviour, resulting in high false positives or missed threats. There is a critical need for a dynamic, intelligent anomaly detection system that can analyze and learn from user or system behaviour over time to identify deviations that may indicate fraud, intrusions, faults, or errors. The problem, therefore, is to develop a behaviour-based anomaly detection framework that can automatically model normal behaviour patterns, effectively distinguish between legitimate and abnormal activities, and accurately detect anomalies in real-time or near real-time with minimal human intervention. This includes addressing issues like scalability, adaptability, contextual analysis, and reducing false alarms, while maintaining compliance with privacy and security regulations.

1.3 Significance and Relevance of Work

This project focuses on detecting insider threats within organizational environments profiles, establishing a baseline of normal activity within different user categories. Additionally, Hidden Markov Models (HMM) will be utilized to capture sequential dependencies in user actions, enabling the system to detect deviations from expected behavior over time. The integration of the Local Outlier Factor (LOF) algorithm will further enhance the system's ability to identify localized anomalies that could signal suspicious or malicious activity, improving the sensitivity of the model to subtle threats.

The significance of anomaly detection through behaviour analysis lies in its ability to intelligently monitor, understand, and respond to complex and dynamic data environments. As digital systems grow in scale and complexity, traditional static or rule-based methods are increasingly inadequate for identifying sophisticated threats, fraud, or operational failures. Behaviour-based anomaly detection offers a proactive approach by learning what constitutes "normal" activity and detecting deviations that could signal security breaches, financial fraud, health abnormalities, or system malfunctions.

This approach is especially relevant in high-risk domains such as cybersecurity, where advanced persistent threats often mimic normal behaviour to evade detection. Similarly, in financial systems, behaviour analysis helps uncover subtle signs of fraudulent transactions that standard filters might overlook. In healthcare and IoT applications, it can identify critical anomalies in patient vitals or device performance, enabling timely interventions.

Moreover, the relevance of this work is amplified by the growing demand for real-time monitoring, the increasing volume and variety of behavioural data, and the necessity for adaptive, scalable, and privacy-aware detection systems. By integrating advanced analytics and machine learning techniques, this work supports the development of intelligent, automated systems that improve accuracy, reduce false positives, and enhance operational security and efficiency across industries.

1.4 Objectives

This project proposes using K-means clustering, Hidden Markov Models (HMM), and Local Outlier Factor (LOF) to detect anomalous user behavior and mitigate insider threats.

The system combines K-means clustering for grouping user behaviors, HMM for sequential by analyzing user behavior patterns through machine learning techniques. The scope includes the application of K-means clustering to group users based on similar behavior pattern analysis, and LOF to identify localized anomalies, providing a comprehensive solution for insider threat detection.

The primary objective of anomaly detection through behavior analysis is to develop intelligent systems capable of identifying unusual or abnormal behavior patterns in real time. These systems aim to learn and model what constitutes "normal" behavior for users, systems, or devices by analyzing historical data. Once a baseline is established, the system continuously monitors new data to detect deviations that may indicate potential threats, fraud, errors, or failures. A key focus is to reduce false positives by incorporating contextual information—such as time, location, and frequency—into the detection process, thus improving accuracy. Furthermore, these systems must be adaptive to changes in behavior over time, ensuring they remain effective in dynamic environments. Scalability and efficiency are also critical, as modern applications often involve large volumes of data from diverse sources. The work is particularly relevant across sectors like cybersecurity, finance, healthcare, and IoT, where early detection of anomalies can prevent major losses or disruptions. Importantly, behavior-based detection methods must also uphold data privacy and security standards, making them suitable for real-world deployment in sensitive domains.

1.5 Methodology

First the Dataset is read. Exploratory Data Analysis is performed on the dataset to clearly understand the statistics of the data, Feature selection is used, A machine learning model is developed. Train and test the model and analysis the performance of the model using certain evaluation techniques such as accuracy, confusion matrix, precision etc.

The methodology for anomaly detection through behavior analysis follows a systematic process aimed at identifying abnormal patterns in behavioral data. It begins with data collection, where behavioral information is gathered from sources such as user activity logs, network traffic, system transactions, or IoT sensor readings. This data is then preprocessed to ensure quality, which includes handling missing values, normalizing data, encoding variables, and removing noise. Once cleaned, the system moves to behavioral profiling, where models are built to understand what constitutes normal behavior. This can be achieved using statistical techniques, clustering algorithms like K-Means, or machine learning approaches that learn patterns from historical data. Following this, the detection phase involves comparing new incoming behavior data against these established profiles. If significant deviations are found, the data is flagged as anomalous. Techniques such as distance-based analysis, density estimation, or deep learning methods like autoencoders are used in this step. The system's performance is then evaluated using accuracy metrics such as precision, recall, F1-score, and confusion matrices. Since behaviour can change over time, continuous model updates and feedback mechanisms are crucial to maintain effectiveness. Finally, results are visualized and reported through dashboards or alert systems to support timely decision-making. This end-to-end methodology ensures a reliable and adaptable framework for detecting anomalies across various real-world scenarios.

CHAPTER 2

LITERATURE SURVEY

As organizations increasingly rely on digital infrastructure, the potential for malicious activity such as data theft, unauthorized access, or sabotage grows. To combat this, there is a need for advanced detection systems that can identify anomalous behaviors and flag potential insider threats before they cause significant damage.

2.1Review of Existing Literature on Garbage and Pothole Detection

[1] Detecting and Identifying Insider threats based on advanced clustering methods

Authors: Oksana Nikiforova,Andrejs Romanaov, Juru koninko

In this paper, the literature survey for "Detecting and Identifying Insider threats based on advanced clustering methods" provides an overview on Advanced Clustering Techniques on how they are used to group user behaviors allowing anomaly detection through behavioral modeling. This method facilitates early identification of unauthorized access of misuse through AI-driven grouping.

Disadvantages : Time consuming.

[2] Comparative Analysis of Unsupervised ML Algorithms for Anomaly Detection

Authors: Junia Mausa, Oliveria, Jonatan Almedia ,Daniel Macedo

In this paper, "Comparative analysis of Unsupervised ML Algorithms for Anomaly Detection" provides an overview of Elliptic Envelope outperformed other algorithms in anomaly detection , achieving high F1 score and AUC metrics . Isolation Forest is the best in Clustering at the end.

Disadvantages : Lack of adaptability

[3] Enhancing Insider Threats Detection in Imbalanced Cybersecurity Settings Using the Density Based Clustering

Authors: Oksana Nikiforova, AndreJs Romanovs

The literature survey for " Enhancing Insider Threats Detection in Imbalanced Cybersecurity Settings Using Clustering" explores the application of machine learning techniques in detecting credit card fraud. It reviews various algorithms including Logistic Regression, Decision Trees, and Random Forest, analyzing their strengths and limitations.

Through a concise analysis of existing research, the survey highlights the importance of feature engineering, model evaluation, and addressing class imbalance in fraud detection. It provides valuable insights into current practices and challenges, offering guidance for future research in the field.

Disadvantages : Imbalanced Data

[4] Anomaly Based Insider Threat Detection using Deep Autoencoders

Authors: Liu Liu, Oliver De Vel, Chao Chen, Jun Zhang

This paper proposes a detection system that utilizes an ensemble of deep autoencoders to identify insider threats by analyzing user behavior for anomalies. The system was evaluated using a benchmark dataset and demonstrated the ability to detect malicious insider actions with a reasonable false positive rate.The survey succinctly discusses the advantages of ensemble techniques in handling complex data patterns and class imbalances. It provides valuable insights for researchers and practitioners looking to enhance fraud detection capabilities using ensemble machine learning.

[5] Scenario Based Insider Threat Detection from Cyber Activities

Authors: Pratik Chattopadhyay, Lipo Wang, Yap Pen Tan

This study introduces a technique for detecting insider threats through time-series and to this classification of user activities. By constructing time-series feature vectors from user activity logs and employing a deep autoencoder neural network, the approach achieved high precision, recall, and F-score in detecting malicious activities

[6] Automated Insider Threat Detection System Using User and Role Based Profile assessment

Authors: P.A.Legg , O.Buckley, M.Goldsmith and S.Creese.

The literature survey for "Automated Insider Threats detection System using user and role based Profile Assessment " explores the integration of machine learning algorithms and cybersecurity in fraud detection. It reviews techniques such as Logistic Regression, Decision Trees, Random Forest, and neural networks within cybersecurity frameworks to enhance fraud detection.

Through a concise analysis, the survey emphasizes the role of machine learning in identifying fraudulent transactions and the importance of cybersecurity measures in preventing fraud. It offers practical guidance for professionals in mitigating fraud risks effectively.

[7] Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity data Streams

Authors: Baris Can, M. Amac Guvensan, Ali Gokhan Yavuz, Elif M. Karsligil

The literature survey for "Deep Learning for Unsupervised Insider Threats Detection in Structured Cybersecurity Data Streams" examines the specific characteristics associated with fraudulent card transactions. It focuses on research conducted by Baris Can, M. Amac Guvensan, Ali Gokhan Yavuz, and Elif M. Karsligil, aiming to identify patterns and indicators of fraudulent activity.

Through concise analysis, the survey sheds light on key features that differentiate fraudulent transactions from legitimate ones. It provides insights into methodologies used for fraud detection, including statistical analysis, machine learning algorithms, and anomaly detection techniques**.**

[8] A Deep Learning Ensemble With Data Resampling for Credit Card Fraud Detection

Authors: Ibomoiye Domor Mienye (Member, IEEE), and Yanxia Sun

The literature survey for "A Deep Learning Ensemble with Data Resampling for Credit Card Fraud Detection" investigates the application of deep learning ensembles with data resampling techniques for credit card fraud detection. Authored by Ibomoiye

Domor Mienye and Yanxia Sun, the survey focuses on identifying the efficacy of ensemble methods in combination with data resampling approaches. Through a targeted analysis, the survey explores how deep learning techniques can be harnessed to improve fraud detection accuracy. It highlights the significance of data resampling techniques in addressing class imbalance issues inherent in credit card fraud datasets.

[9] E-Watcher:Insider Threat monitoring and Detection for Enhanced Security

Authors: Kuldeep Randhawa, Manjeevan Seera, Chukiong Loo (Senior Member, IEEE), Chee Peng Lim (Senior Member, IEEE), and Asoke K. Nandi

The literature survey for "E-Watcher : Insider Threat monitoring and Detection for Enhanced Security" investigates the application of AdaBoost and Majority Voting techniques for credit card fraud detection. Authored by Kuldeep Randhawa, Manjeevan Seera, Chukiong Loo, Chee Peng Lim, and Asoke K. Nandi, the survey focuses on evaluating the effectiveness of these ensemble methods.

Through a focused examination, the survey explores how AdaBoost and Majority Voting can be leveraged to enhance fraud detection accuracy. It discusses the benefits of ensemble learning in combining multiple classifiers to improve overall performance.

[10] New Insider Threat Detection Method Based on Recurrent Neural Networks

Authors: Abhilash Sharma M, Ganesh Raj B R, Ramamurthy B, and Hari Bhaskar R

The literature survey for "New Insider Threats detection based on Recurrent Neural Networks" investigates the application of deep learning techniques, specifically auto-encoders, for fraud detection. Authored by Abhilash Sharma M, Ganesh Raj B R, Ramamurthy B, and Hari Bhaskar R, the survey focuses on evaluating the effectiveness of auto-encoders in detecting fraudulent transactions.

Through concise analysis, the survey examines how auto-encoders learn representations of credit card transaction data and identify anomalies indicative of fraud. It discusses the advantages of deep learning approaches in capturing complex patterns and adapting to evolving fraud tactics.

[11] Anomaly Detection Based on K-Means and Rough K-Means

Authors: DR. A. Suresh Rao

This study applies K-Means and Rough K-Means clustering to network intrusion detection.Evaluated on the KDD'99 dataset, the approach demonstrates improved detection rates and reduced false positives compared to existing systems.

[12] Anomaly Detection in Network Traffic Using K-Means Clustering

Authors: R. Kumari, M.K. Singh, R. Jha

The study applies K-Means clustering to network traffic data for anomaly detection. By grouping similar traffic patterns, the method identifies outliers that may signify potential security threats or network issues.

[13] An Abnormal Behavior Clustering Algorithm Based on K-Means

Authors: Jianbiao Zhang, Feng Yang, Shouling Tu, Anni Zhang

This research presents an improved K-Means algorithm for clustering abnormal behavior. By selecting initial cluster centers based on data density, the method enhances clustering performance and effectively identifies anomalies in behavioral data.

[14] Extending Isolation Forest for Anomaly Detection in Big Data via K-Means

Authors: Md Tahmid Rahman Laskar, Jimmy Huang, Vladan Smetana, Chris Stewart, Kees Pouw, Aijun An, Stephen Chan, Lei Liu

This study proposes a novel unsupervised machine learning approach that combines the K-Means algorithm with the Isolation Forest for anomaly detection in industrial big data scenarios. Implemented using Apache Spark, the model was trained on large-scale network traffic data (~123 million instances) stored in Elasticsearch. The hybrid model effectively detects anomalies in real-time, addressing challenges in training on large datasets and demonstrating comparable performance to state-of-the-art approaches.

[15] Detecting Abnormal Events in Video Using Narrowed Normality Clusters

Authors: Radu Tudor Ionescu, Sorina Smeureanu, Marius Popescu, Bogdan Alexe

This paper formulates abnormal event detection in videos as an outlier detection task. It introduces a two-stage algorithm combining K-Means clustering and one-class Support Vector Machines (SVM). Initially, K-Means clusters normal motion and appearance features extracted from training videos. Clusters with fewer samples are considered outliers and eliminated. Subsequently, one-class SVMs are trained on the remaining clusters to refine the detection of abnormal events in test videos. The method achieves real-time processing at 24 frames per second on a single CPU.

[16] A Hybrid Approach: Utilizing K-Means Clustering and Naive Bayes for IoT Anomaly Detection

Authors: Lincoln Best, Ernest Foo, Hui Tian

This research presents a hybrid anomaly detection algorithm for Internet of Things (IoT) systems, combining unsupervised K-Means clustering with supervised AdaBoosted Naive Bayes classification. Initially, K-Means clusters the data, identifying potential anomalies. These clusters are then used to train the Naive Bayes classifier, enhancing the accuracy of anomaly detection across diverse IoT devices. The proposed method achieves high accuracy, precision, and recall rates, ranging from 90% to 100%, demonstrating its effectiveness in ensuring data integrity and privacy in IoT environments.

[17] Evaluation of Machine Learning-Based Anomaly Detection Algorithms on an Industrial Modbus/TCP Dataset

Authors: Simon Duque Anton, Suneetha Kanoor, Daniel Fraunholz, Hans Dieter Schotten

This study evaluates various machine learning algorithms, including K-Means clustering, for detecting anomalies in industrial Modbus/TCP communication data. The research highlights the challenges of applying K-Means in industrial settings, noting its limitations in handling the specific characteristics of Modbus/TCP traffic. The findings suggest that while K-Means has potential, it may require enhancements or hybrid approaches to effectively detect anomalies in such specialized datasets.

[18] Anomaly Detection Using K-Means and Long Short-Term Memory for LSSPV Plants

This research develops a clustering model for anomaly detection in Large-Scale Solar Photovoltaic (LSSPV) plants by combining K-Means clustering with Long Short-Term Memory (LSTM) neural networks. The hybrid model effectively identifies anomalies in the electrical output current of LSSPV systems, enhancing the reliability and efficiency of solar energy production. The integration of K-Means and LSTM allows for capturing both spatial and temporal patterns in the data, leading to improved detection performance.

[19] Computer User Behavior Anomaly Detection Based on K-Means Algorithm
Authors: Yang Yang, Juan Hao, Jianguang Zhao, Cihang Chen

This paper proposes a user behavior anomaly detection model utilizing the K-Means algorithm. The model quantifies user behavior attributes and applies K-Means clustering to identify abnormal patterns. By analyzing user behavior sequences and employing a sliding time window, the method effectively detects deviations from normal behavior, enhancing the security of computer systems.

[20] Anomaly Detection and Improvement of Clusters Using Enhanced K-Means Algorithm

This study introduces an enhanced K-Means algorithm aimed at improving cluster quality and anomaly detection accuracy. The enhancements address common issues in traditional K-Means, such as sensitivity to initial centroid selection and the presence of outliers. By refining the clustering process, the proposed method achieves better separation of normal and anomalous data points, leading to more reliable detection outcomes.

[21] Leveraging K-Means Clustering and Z-Score for Anomaly Detection in Cryptocurrency Transactions

This research presents a hybrid approach combining K-Means clustering with Z-score analysis to detect anomalies in cryptocurrency transactions. The method clusters transaction data using K-Means and then applies Z-score calculations to identify outliers within each cluster. This approach enhances the detection of fraudulent or suspicious activities in cryptocurrency networks by capturing both cluster-based and statistical anomalies.

[22] Machine Learning-Based Anomaly Detection Using K-Means Array and Sequential Minimal Optimization
Authors: Gadal

This study explores a hybrid anomaly detection algorithm that integrates K-Means clustering with Sequential Minimal Optimization (SMO). The K-Means algorithm initially clusters the data, and the SMO technique is then applied to refine the classification boundaries between normal and anomalous instances. This combination aims to leverage the strengths of both unsupervised and supervised learning methods to improve detection accuracy.

[23] Network Anomaly Detection by Cascading K-Means Clustering and C4.5 Decision Tree

This paper proposes a two-stage anomaly detection method that combines K-Means clustering with the C4.5 decision tree algorithm. Initially, K-Means clusters the network data to identify potential anomalies. Subsequently, the C4.5 decision tree refines the classification by learning decision rules from the clustered data. This cascading approach enhances the detection of network anomalies by integrating unsupervised and supervised learning techniques.

[24] Exploring the Role of Behavioral Analytics and Anomaly Detection in Cybersecurity

Authors: Parwez et al.

This study investigates the application of behavioral analytics and anomaly detection techniques, including K-Means clustering, in cybersecurity. The research emphasizes the importance of understanding user behavior patterns to identify deviations that may indicate security threats. By employing unsupervised learning methods, the study aims to enhance the detection of anomalous activities in cyber environments.

[25] A Comprehensive Investigation of Clustering Algorithms for User and Entity Behavior Analytics

Authors: Iglesias Perez and Criado

This research provides an in-depth analysis of various clustering algorithms, including K-Means, applied to User and Entity Behavior Analytics (UEBA). The study examines the effectiveness of these algorithms in detecting anomalies by analyzing time-series data and employing graph analysis techniques. The findings contribute to the development of more robust UEBA systems for identifying security threats.

[26] Anomaly Detection with K-Means in Python

This article provides a practical guide on implementing anomaly detection using K-Means clustering in Python. It includes code examples and explanations on how to apply K-Means to identify outliers in datasets, making it a valuable resource for practitioners seeking hands-on experience with anomaly detection techniques.

2.2 Gaps in the Literature Survey

1. Limited Adaptability to Dynamic Behavior Patterns

- **Observation**: Most studies assume static behavioral patterns, but real-world behavior often evolves over time (e.g., user behavior on networks or smart grids).
- **Gap**: Lack of adaptive or incremental K-Means models that update clusters with real-time behavioral changes.

2. Sensitivity to Initialization and Cluster Number (k)

- **Observation**: K-Means is highly sensitive to the initial centroids and the selection of 'k'.
- **Gap**: Few studies have addressed automated or data-driven methods to optimally choose the number of clusters or improve centroid selection for anomaly detection.

3. Insufficient Handling of Mixed-Type and Categorical Data

- **Observation**: K-Means performs poorly with categorical or mixed-type data without proper encoding.
- **Gap**: There's limited work integrating K-Means with models like **K-Prototypes** or **distance measures** suitable for categorical attributes in behavioral datasets.

4. Scalability on High-Dimensional Data

- **Observation**: As dimensionality increases, distance metrics used in K-Means lose effectiveness (curse of dimensionality).
- **Gap**: Inadequate focus on dimensionality reduction techniques (e.g., PCA, t-SNE) or scalable K-Means variants (e.g., Mini-Batch K-Means) tailored for large-scale behavioral data.

5. Poor Performance on Imbalanced Data

- **Observation**: Anomalies often constitute a very small fraction of the dataset.
- **Gap**: Few papers address class imbalance in unsupervised settings, which results in under-detection of rare but important anomalous behaviors.

6. Lack of Benchmark Datasets and Standard Evaluation Metrics

- **Observation**: Many studies use proprietary or simulated datasets.
- **Gap**: There’s a lack of consensus on benchmark datasets and evaluation metrics, making cross-study comparison and replication difficult.

7. Integration with Temporal Behavior

- **Observation**: User or system behavior is often time-dependent.
- **Gap**: Very few studies incorporate **time-series analysis or temporal clustering** with K-Means to capture evolving patterns.

8. Explainability and Interpretability

- **Observation**: Black-box models offer little insight into why a behavior is anomalous.
- **Gap**: There is minimal focus on interpretable anomaly detection using K-Means that provides meaningful explanations to domain experts.

9. Domain-Specific Customization

- **Observation**: One-size-fits-all models are often applied generically.
- **Gap**: Need for domain-adapted clustering strategies that consider behavioral context (e.g., healthcare, finance, IoT).

10. Hybrid Model Evaluation

- **Observation**: Many works propose hybrid models (e.g., K-Means + SVM, K-Means + Isolation Forest), but comparative analysis is often missing.
- **Gap**: Lack of thorough benchmarking of hybrid models across multiple domains and performance metrics.

2.3 Conclusion

K-Means clustering has proven to be a highly versatile and foundational technique in behavioral anomaly detection across diverse domains such as cybersecurity, IoT, financial fraud, network security, and energy systems. Its unsupervised nature allows it to effectively identify patterns in unlabeled data by grouping similar behavior and flagging deviations as anomalies. Several studies have enhanced its performance by integrating it with other algorithms such as Isolation Forests, One-Class SVMs, Naive Bayes classifiers, and LSTM networks to improve accuracy, scalability, and real-time detection capabilities.

Despite its strengths—simplicity, speed, and interpretability—K-Means is not without limitations. It can be sensitive to initial cluster centroids and struggles with detecting anomalies in non-spherical or overlapping data distributions. Researchers have addressed these issues by introducing hybrid models, improved initialization methods, and by combining statistical techniques like Z-scores to strengthen anomaly detection performance.

Overall, the research consensus suggests that K-Means is a powerful baseline or component for anomaly detection systems but often requires augmentation with other methods or preprocessing techniques to achieve optimal results, especially in dynamic or large-scale environments.

CHAPTER-3

SYSTEM REQUIREMENTS AND SPECIFICATION

3.1 System Requirement Specification:

System Requirement Specification (SRS) is a fundamental document, which forms the foundation of the software development process. The System Requirements Specification (SRS) document describes all data, functional and behavioral requirements of the software under production or development. An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two- way insurance policy that ensures that both the client and the organization understand the other's requirements from that perspective at a given point in time. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications,testing and validation plans, and documentation plans, are related to it.It is important to note that an SRS contains functional and non-functional requirements only. It doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands about the customer's system requirements.

3.2 Hardware specification

➢ RAM: 4GB and Higher

➢ Processor: intel i3 and above

➢ Hard Disk: 500GB: Minimum

3.3 Software specification

➢ OS: Windows or Linux

➢ Python IDE: python 3.7.x and above

➢ Jupyter Notebook

➢ Language: Python

3.4 Functional Requirements:

Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. In this system following are the functional requirements:

- Collect the Datasets.
- Train the Model.
- Predict the results

3.5 Non-Functional Requirements

- The system should be easy to maintain.
- The system should be compatible with different platforms.
- The system should be fast as customers always need speed.
- The system should be accessible to online users.
- The system should be easy to learn by both sophisticated and novice users.
- The system should provide easy, navigable and user-friendly interfaces.
- The system should produce reports in different forms such as tables and graphs for easy visualization by management.
- The system should have a standard graphical user interface that allows for the online

3.6 Performance Requirement:

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into the required environment. It rests largely with the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements.

CHAPTER-4

SYSTEM ANALYSIS

Systems analysis is the process by which an individual studies a system such that an information system can be analyzed, modeled, and a logical alternative can be chosen. Systems analysis projects are initiated for three reasons: problems, opportunities, and directives

4.1 Existing System

Several systems and tools exist today for **Insider Threat Detection**, which use a variety of methods to monitor, detect, and respond to potential threats originating from within an organization. These systems typically rely on a combination of user behavior analytics (UBA), machine learning, and data monitoring to identify unusual or suspicious activities.

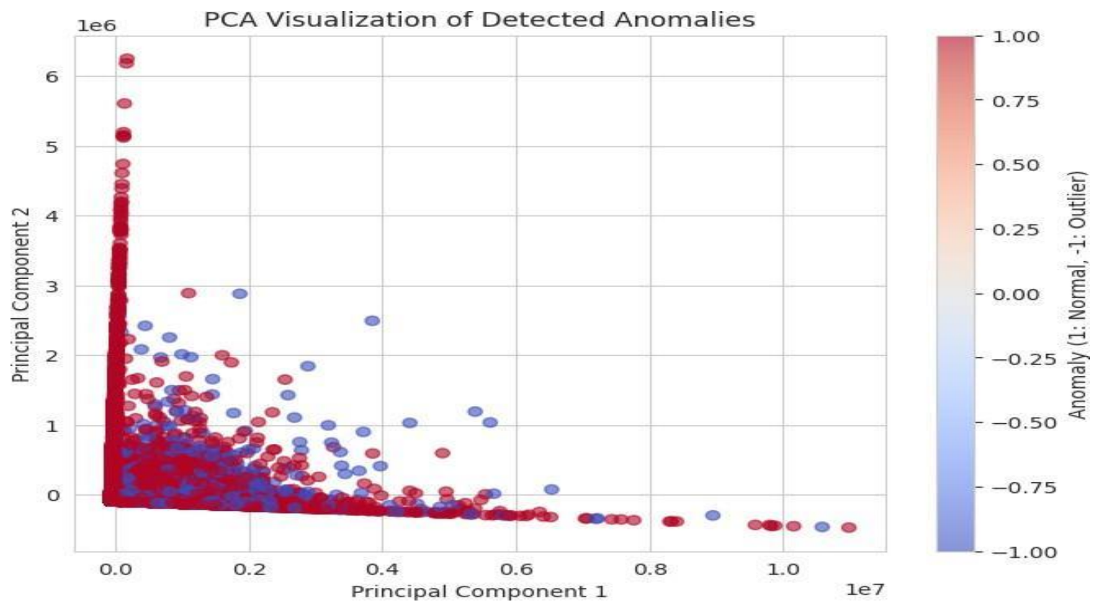


Figure 4.1 Anomaly Representation

As per the Fig 4.1 The image presents a PCA (Principal Component Analysis) visualization used to detect anomalies in a dataset by reducing its dimensionality and plotting the results in two principal components. Each data point is represented in a 2D space, where the x-axis and y-axis correspond to the first and second principal components, respectively. The points are color-coded based on their anomaly scores, with a gradient from blue to red indicating the degree of abnormality. Blue points (closer to -1.0 on the color bar) represent normal instances, while red points (closer to 1.0) signify outliers or anomalies. This distribution shows that many anomalies are scattered near the origin and along certain regions, suggesting a difference in underlying patterns captured through PCA. The plot, titled "PCA Visualization of Detected Anomalies," and labeled as "Figure 4.1 Anomaly Representation," appears to be part of a larger analytical report or research study. This type of visualization is valuable in understanding how well an anomaly detection model—likely unsupervised—differentiates between normal and abnormal data in a transformed feature space.

4.1.1 Limitations

The PCA visualization of detected anomalies, while useful for reducing high-dimensional data into a 2D plot, has several important limitations. First, PCA is a linear dimensionality reduction technique, meaning it assumes linear relationships in the data and may fail to capture complex, non-linear patterns where anomalies often exist.

Additionally, projecting data onto only the first two principal components may result in a loss of critical information, potentially masking important anomalies that are more apparent in higher dimensions. The plot also suffers from visual clutter and overlapping data points, particularly near the origin, which makes it difficult to distinguish between normal points and outliers.

Furthermore, the color gradient used to represent anomaly scores may be misleading—while it appears continuous, the actual anomaly classification is likely binary, which could confuse interpretation.

Overall, while this PCA visualization provides a broad overview, it lacks the granularity and accuracy needed for reliable anomaly detection in more complex datasets.

4.2 Proposed System

4.2.1 K Means Clustering:

K-means clustering is an unsupervised machine learning algorithm used to partition data into K distinct clusters based on similarity. It works by assigning data points to the nearest centroid, then iteratively adjusting the centroids to minimize the within-cluster variance. The goal is to group similar data points together, with each cluster represented by its centroid.

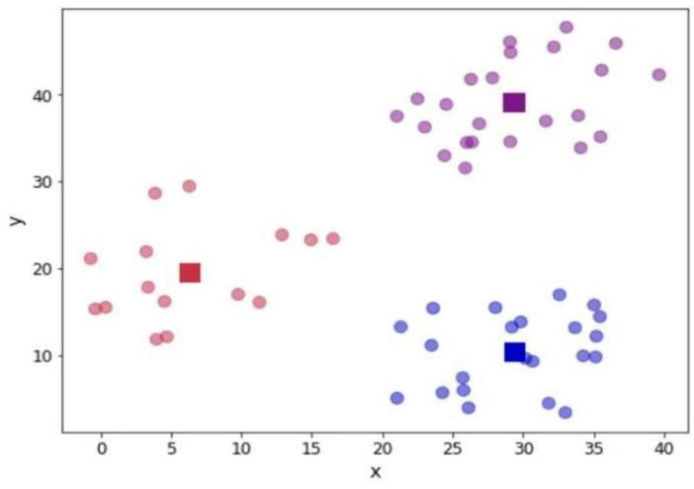


Fig 4.2 K Means Clustering

As per the Fig 4.2 The image illustrates the result of K-Means Clustering, a popular unsupervised machine learning algorithm used to partition data into distinct groups based on feature similarity. In the scatter plot, each point represents an individual data sample in a 2D feature space, with x and y serving as the two dimensions.

The points are colored differently to indicate the three clusters identified by the K-Means algorithm. Each cluster is visually distinct, and the data points within a cluster are grouped close together, indicating similar characteristics. The large square markers represent the centroids of the clusters — these are the central points calculated by the algorithm that minimize the intra-cluster variance. The centroids are shown in the same color as their respective clusters: red, purple, and blue.

4.2.2 Local Outlier Factor

Local Outlier Factor (LOF) is an unsupervised anomaly detection algorithm that measures the local density deviation of a data point relative to its neighbors. It assigns an anomaly score based on how isolated a point is compared to its surrounding points. Higher LOF scores indicate potential outliers, making it useful for detecting fraud and rare events.

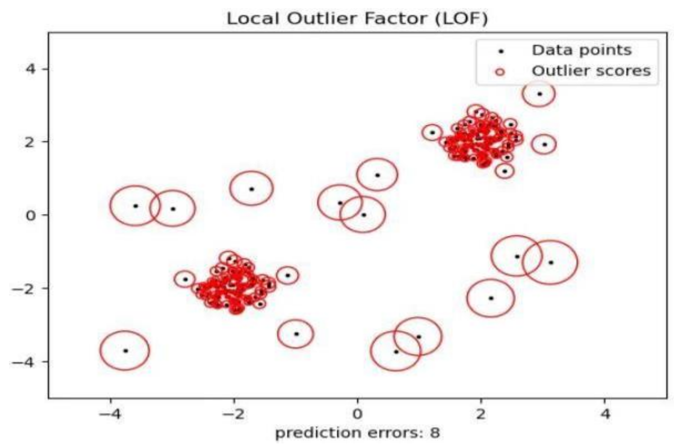


Fig 4.3 local Outlier Factor

As per the Fig 4.3 displays the results of an anomaly detection algorithm called Local Outlier Factor (LOF), which is used to identify data points that deviate significantly from their local neighborhoods. This technique is especially useful for detecting density-based anomalies in data.

In the scatter plot, small red dots represent the data points, while larger red circles drawn around some of the points indicate outlier scores — i.e., the higher the LOF score, the more anomalous the point is considered. The size of the red circles appears proportional to the magnitude of the outlier score, meaning points with larger circles are more likely to be anomalies.

The graph shows several clusters of normal data points where most red circles are small, while a few isolated or scattered points have large red circles, indicating they are likely outliers. The plot title at the top reads "Local Outlier Factor (LOF)", and the legend in the top-left corner labels the two elements: "Data points" (black/red dots) and "Outlier scores" (red circles).

4.2.3 Hidden Markov Model

A Hidden Markov Model (HMM) is a statistical model that represents systems with hidden states evolving over time, where observations depend probabilistically on these states. It consists of a set of hidden states, observable outputs, transition probabilities, and emission probabilities. HMMs are widely used in speech recognition, bioinformatics, and financial modeling.

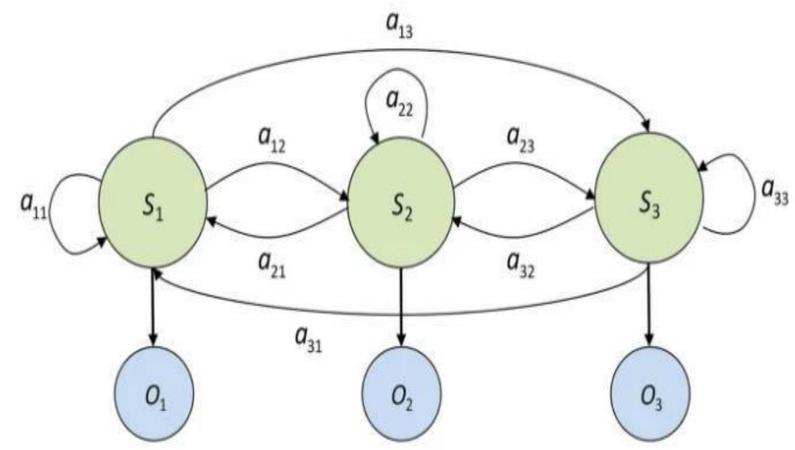


Fig 4.4 Hidden Markov Model

As per the Fig 4.4 illustrates a Hidden Markov Model (HMM), which is a statistical model used to describe systems that transition between a set of hidden states over time, each of which produces observable outputs. In the diagram, the green circles labeled S1,S2,S3,S_1, S_2, S_3,S1,S2,S3 represent the hidden states of the system, while the blue circles O1,O2,O3,O_1, O_2, O_3,O1,O2,O3 denote the observable outputs or emissions associated with those states. The arrows between the states, labeled aija_{ij}aij, indicate the transition probabilities, showing the likelihood of the system moving from one state to another. Each state also has a self-loop, such as a11,a22,a33,a{11}, a{22}, a{33},a11,a22,a33, representing the probability of remaining in the same state. Vertical arrows from the hidden states to the observations represent the emission probabilities, which define how likely a state is to produce a particular observable output. This HMM structure is widely used in fields such as speech recognition, bioinformatics, and time-series modeling, where the actual state of the system is not directly observable but can be inferred from the sequence of outputs over time.

4.3 Advantages

- **Enhanced Anomaly Detection** – LOF effectively detects local anomalies in user behavior, identifying unusual activities that deviate from normal patterns.
- **Sequential Behavior Modeling** – HMM captures temporal dependencies and probabilistic transitions in user actions, helping detect subtle and evolving insider threats.
- **Contextual Relationship Analysis** – GNN models complex relationships between users, devices, and transactions, uncovering hidden threat patterns in networked environments.
- **Robustness to Evasion** – Attackers trying to evade one method (e.g., anomaly detection) may still be flagged by another (e.g., sequence modeling or graph analysis).
- **Reduced False Positives** – By integrating multiple perspectives (local, sequential, and structural the system achieves more accurate threat detection with fewer false alarms.

CHAPTER-5
SYSTEM DESIGN

5.1 Project Modules

Entire project is divided into 3 modules as follows:

a. Data Gathering and pre processing

b. Training the model using following Machine Learning algorithms

c. K Means Clustering with Local Outlier Factor Algorithm Combined Then Integrated with Hidden

d. Markov Model for dynamic data integration

Module 1: Data Gathering and Data Pre processing

a. A proper dataset is searched among various available ones and finalized with the dataset. The dataset must be preprocessed to train the model.

b. In the preprocessing phase, the dataset is cleaned and any redundant values, noisy data and null values are removed.

c. The Preprocessed data is provided as input to the module.

Module 2: Training the model

- The Preprocessed data is split into training and testing datasets in the 80:20 ratio to avoid the problems of over-fitting and under-fitting.
- A model is trained using the training dataset with the following algorithms SVM, Random Forest Classifier and Decision Tree
- The trained models are trained with the testing data and results are visualized using bar graphs, scatter plots.
- The accuracy rates of each algorithm are calculated using different params like F1 score, Precision, Recall. The results are then displayed using various data visualization tools for analysis purpose.
- The algorithm which has provided the better accuracy rate compared to remaining algorithms is taken as final prediction model.

Module 3: Final Prediction model integrated with front end

- The algorithm which has provided better accuracy rate has been considered as the final prediction model.
- The model thus made is integrated with the front .
- .Database is connected to the front end to store the user information who is using it.

5.2 SYSTEM ARCHITECTURE

Our Project main purpose is to making Credit Card Fraud Detection awaring to people from credit card online frauds. the main point of credit card fraud detection systemis necessary to safe our transactions & security. With this system, fraudsters don't havethe chance to make multiple transactions on a stolen or counterfeit card before the cardholder is aware of the fraudulent activity. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

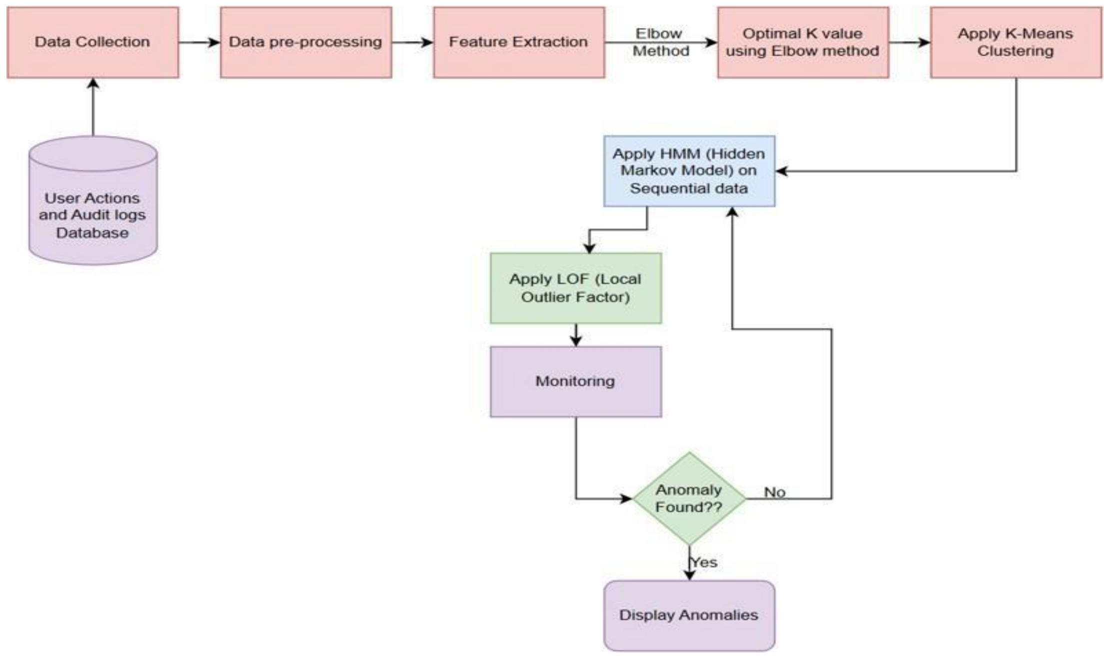

Data Collection
Data pre-processing
Feature Extraction
Elbow Method
Optimal K value using Elbow method
Apply K-Means Clustering
User Actions and Audit logs Database
Apply HMM (Hidden Markov Model) on Sequential data
Apply LOF (Local Outlier Factor)
Monitoring
Anomaly Found??
No
Yes
Display Anomalies

Fig 5.1 System Architecture

As per the Fig 5.1 represents a flowchart outlining a comprehensive process for anomaly detection using a combination of machine learning techniques. The process begins with the collection of user actions and audit log data from a database. This raw data undergoes pre-processing to clean and prepare it for analysis, followed by feature extraction to transform the data into meaningful inputs.

The Elbow Method is then used to determine the optimal number of clusters (K value) for applying K-Means clustering, which helps group similar behavioral patterns. Parallelly, for sequential data, a Hidden Markov Model (HMM) is applied to analyze temporal patterns. After clustering and sequence modeling, the Local Outlier Factor (LOF) algorithm is used to detect data points that significantly deviate from their local neighborhoods, which may indicate potential anomalies. The results of this process are fed into a monitoring system, where the presence of anomalies is evaluated.

If an anomaly is detected, it is flagged and displayed for further investigation; otherwise, the system continues monitoring. This integrated approach combines unsupervised clustering, temporal modeling, and density-based anomaly detection to provide a robust framework for identifying suspicious or irregular activities in user behavior data.

This flowchart demonstrates a layered and intelligent anomaly detection framework that leverages both spatial and temporal analysis to improve accuracy. By combining K-Means clustering with Hidden Markov Models and the Local Outlier Factor algorithm, the system captures not only the general patterns in user behavior but also the sequential dependencies and local density variations within the data.

The use of the Elbow Method ensures that the clustering is optimized for performance, avoiding underfitting or overfitting. Importantly, the monitoring stage acts as a continuous evaluation mechanism, allowing the system to detect anomalies in real time or over recurring data cycles.

5.3 Activity diagram

Activity diagram is an important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.The control flow is drawn from one operation to another. This flow can be sequential,branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc. The basic purposes of the activity diagram are it captures the dynamic behavior of the system. Activity diagram is used to show message flow from one activity to another Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the messagepart.

5.4 Use Case Diagram

In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system.

These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system. You can model a complex system with a single use-case diagram, or create many use-case diagrams to model the components of the system. You would typically develop use- case diagrams in the early phases of a project and refer to them throughout the development proces.

5.5 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

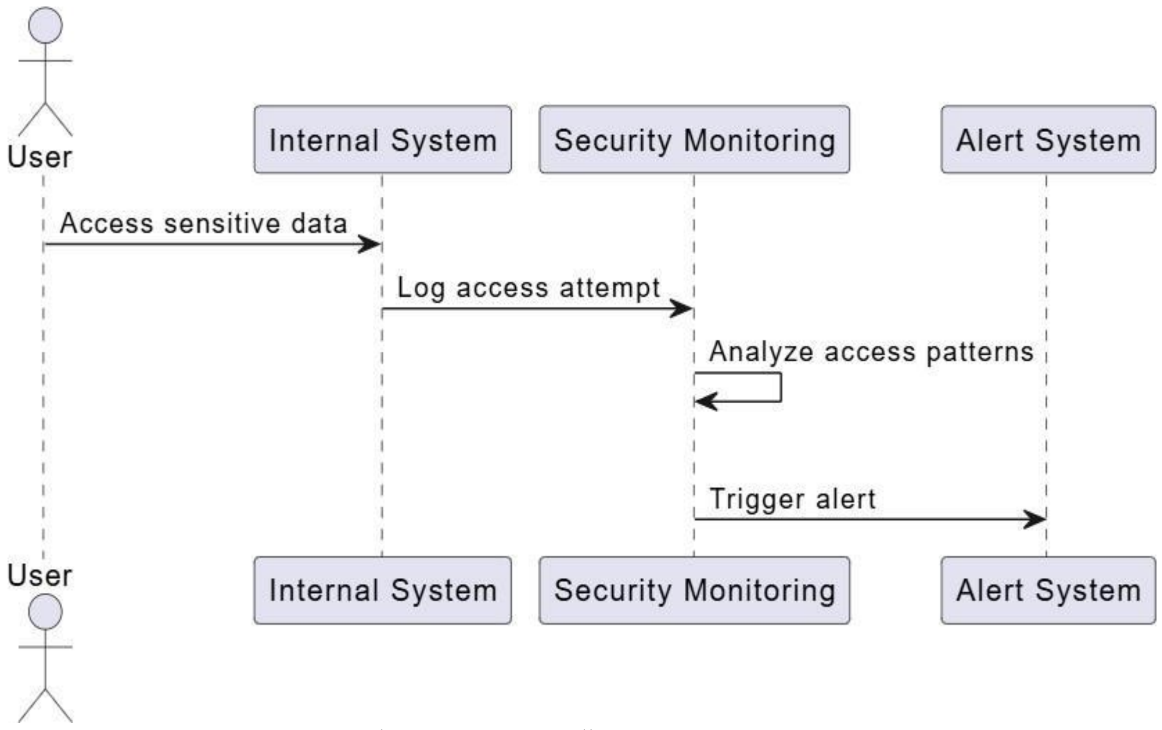


Fig 5.2 Sequence diagram

As per the Fig 5.2 illustrates a sequence diagram that models the interaction between various components involved in monitoring user access to sensitive data. The diagram captures the flow of events starting from a user initiating access to sensitive data via an internal system. This action is logged by the Internal System, which then sends the log information to the Security Monitoring module. The security monitoring system subsequently analyzes access patterns to detect any unusual or unauthorized behavior.

5.6 Data Flow Diagram

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how data enters and leaves the system, what changes the information, and where data is stored. The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and a person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called a data flow graph or bubble chart.

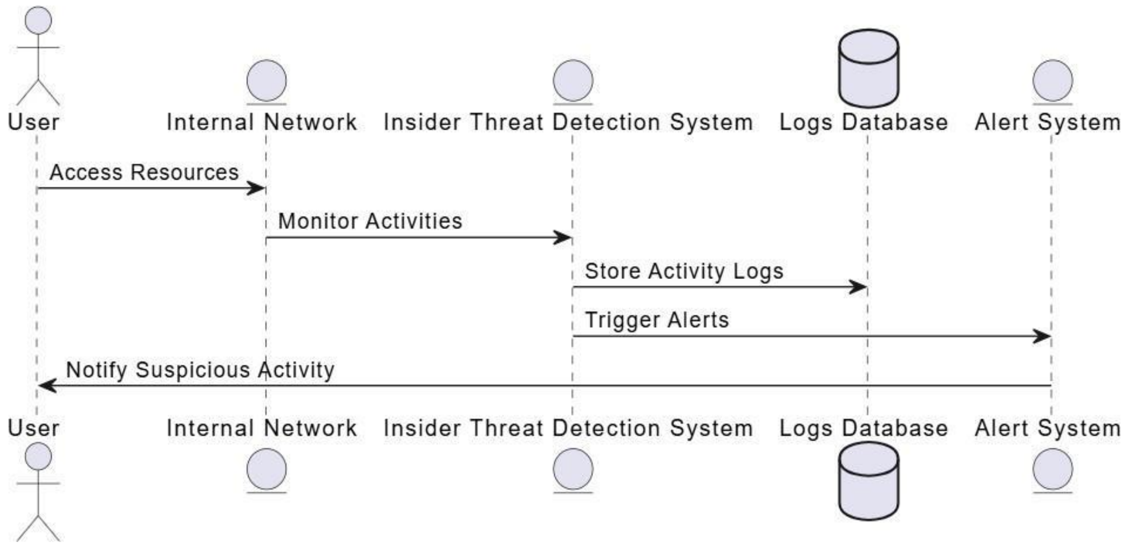


Fig 5.3 Data Flow diagram

As per the Fig 5.3 illustrates a sequence diagram depicting the workflow of an Insider Threat Detection System within an enterprise environment. The process begins when a user attempts to access internal resources through the organization's internal network. This action is monitored in real time by the Insider Threat Detection System, which keeps track of user activities to detect any signs of malicious or unauthorized behavior.

CHAPTER – 6
METHODOLOGY

6.1 Data Collection

Behavioral data is the foundation for anomaly detection. The type of data collected depends on the application domain:

- **Network Security**: Packet capture (PCAP) files, NetFlow logs, firewall logs.
- **User Behavior**: Website clickstream data, login/logout timestamps, keystroke dynamics.
- **Healthcare**: Patient vital signs, medication records, lab test data.
- **IoT/Smart Devices**: Sensor readings, device usage logs, smart meter data.
- **Tools**: Wireshark, log files, public datasets (e.g., NSL-KDD, CICIDS, UCI Machine Learning Repository)

6.2 Data Preprocessing

- Real-world data is often noisy or incomplete. Preprocessing ensures the quality of input data for clustering.
- **Handling Missing Data**: Impute with mean/median/mode or remove records.
- **Encoding Categorical Data**: Convert strings to numbers using one-hot encoding or label encoding.
- **Normalization**: Ensures all features contribute equally. E.g.,
- `X scaled=(X−mean)/std`
- **Dimensionality Reduction (Optional)**:
- **PCA**: Reduces dimensions while preserving variance.
- **t-SNE or UMAP**: Useful for visualizing clusters in 2D.
- **Tools**: Pandas, Scikit-learn, NumPy

6.3 Applying K-Means Clustering

- K-Means aims to partition data into **k** clusters by minimizing intra-cluster variance.
- **Algorithm Steps**:
- Initialize **k** centroids randomly or using **k-means++** for better performance.
- Assign each data point to the nearest centroid (usually Euclidean distance).

- Recompute centroids as the mean of assigned points.
- Repeat until convergence (centroids stop changing).
- **Choosing K**:
- **Elbow Method**: Plot WCSS (within-cluster sum of squares) vs. k and look for the "elbow."
- **Silhouette Score**: Measures how similar an object is to its own cluster vs. other clusters.
- **Tools**: Scikit-learn KMeans, yellowbrick for visualization

6.4 Anomaly Detection

- Once clusters are formed, anomalies are identified as points that:
- Are **far from their cluster centroids** (based on a distance threshold).
- Belong to **small or sparse clusters**, which often indicate abnormal behavior.
- Have **low density** around them (outliers in high-dimensional space).
- **Approaches**:
- Compute Euclidean distance from each point to its cluster center.
- Define an anomaly threshold using z-scores or top % farthest points.

6.5 Evaluation

- Evaluating anomaly detection is difficult without labeled data. If labels exist:
- **Confusion Matrix**: Measures TP, FP, FN, TN.
- **Precision, Recall, F1-Score**:
- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- **AUC-ROC Curve**: Plots true positive rate vs. false positive rate.
- If no labels:
- **Silhouette Score**: Validates cluster consistency.
- **Cluster Validation Indexes**: Davies–Bouldin Index, Dunn Index.

6.6 Visualization

- Helps interpret clusters and identify anomalies visually.
- **2D/3D Plots**: Use PCA or t-SNE to reduce dimensions.
- **Cluster Plots**: Color-coded scatter plots showing clusters and outliers.
- **Time-Series Graphs**: Show anomalous events over time for behavior analysis.

CHAPTER - 7
IMPLEMENTATION

7.1 DATASET DESCRIPTION

7.1.1. CERT Dataset (Computer Emergency Response Team)

The CERT dataset is a repository of data collected by different Computer Emergency Response Teams (CERTs) from around the globe, monitoring and reacting to cybersecurity incidents as well as vulnerabilities. Such datasets are often applied by cybersecurity researchers, organizations, and developers to scrutinize and enhance the identification of cyber threats, such as malware, data breaches, and insider threats. The CERT dataset offers detailed logs and reports on cyberattacks, system vulnerabilities, and other security incidents.

In our project, the CERT dataset can be utilized to train and test models for anomaly detection, insider threat detection, and cyberattack prediction, assisting in the comprehension of malicious behaviour patterns, system vulnerabilities, and attack techniques.

Key Features:

Data Types:

- The CERT dataset generally consists of incident reports, system logs, and security alerts pertaining to cybersecurity incidents. This comprises network logs, user activity logs, and data breach reports, among others.
- Malware reports, DDoS attack reports, and phishing attempts are typically included, giving detailed information regarding different attack vectors and their effects on systems.

Size and Format:

- The size of the dataset may differ depending on the organization or CERT. It usually has a few gigabytes to terabytes of data depending on the time frame and range of incidents included.
- The information tends to come in log file formats such as CSV, JSON, or XML, with structured metadata such as time stamps, incident severity, source and destination IP addresses, attack types, and system impact.

Data Points:

- Incident Type: Represents the nature of the cybersecurity event (e.g., DDoS, phishing, ransomware).
- Source IP and Destination IP: The source and destination of the attack, which might assist in identifying malicious parties or targeted systems.
- Severity Level: The level of severity of the incident (e.g., critical, high, medium, low) reflecting the possible effect on systems or data.
- Attack Vector: Explains how the attack was executed (e.g., SQL injection, brute force, social engineering).
- System/Asset Affected: The system or asset affected (e.g., web servers, database servers).
- Remediation Actions: Remediation actions taken by the CERT or the organization to counteract the incident.

Annotations:

- Categorization: The data is usually annotated with labels including malicious and non-malicious to assist in training machine learning models to identify between normal user activities and cyber attacks.
- Severity Annotations: Events are usually classified by severity level, assisting the model in prioritizing critical events.
- Attack Type Annotations: The dataset provides information about the type of attack, aiding in the development of specific threat detection systems (e.g., for DDoS attacks or insider threats).

7.1.2. CERT-IN Dataset (Computer Emergency Response Team - India)

The CERT-IN dataset is provided by the Indian Computer Emergency Response Team (CERT-IN), which is the national agency responsible for responding to computer security threats and incidents. The dataset has logs and reports of different cybersecurity incidents in India, with an emphasis on cyber incidents, vulnerabilities, and attacks. The data is used to identify patterns of cybersecurity threats and is a good resource for anomaly detection, particularly in detecting insider threats and knowing the nature of different cybersecurity incidents in real-life scenarios.

Key Features:

Data Type:

- The data consists of a vast variety of cybersecurity incident logs and reports. The data includes system logs, network traffic logs, malware reports, and security incidents that were reported to CERT-IN during a certain time frame.

Size:

- The CERT-IN dataset is large and can vary from tens of gigabytes to hundreds of gigabytes based on the duration and data gathered. The dataset is normally updated periodically with fresh data concerning ongoing cybersecurity attacks.

Format:

- Data is generally stored in log file formats (e.g., CSV, JSON, XML) and could be accompanied by structured data like timestamps, description of the event, level of severity, source IP addresses, types of threats, and other incident- related metadata.

Data Points:

- The CERT-IN dataset may include data like: Incident type (e.g., DDoS attacks, malware infections, unauthorized access, phishing attacks).
- Tasked IP addresses related to the incidents.
- Systems impacted (e.g., web servers, application servers).
- Incident severity (e.g., critical, high, low).
- Vulnerability reports (e.g., CVE identifiers, description of vulnerabilities).
- Remediation or mitigation actions performed by CERT-IN.

Annotations:

The dataset typically contains annotations that classify incidents as:

- Malicious: Incidents that are indicative of cyberattacks or security violations.
- Non-malicious: Incidents that are classified as false positives or harmless anomalies.
- Severity levels: The dataset contains severity indicators to indicate the impact of

incidents, from low to critical.

Use Case:

The CERT-IN dataset is important for discovering emerging cybersecurity threats, comprehending attack vectors, and identifying anomalous behaviour in network traffic or system logs. In your project, this dataset can be utilized to train anomaly detection models, i.e., insider threat detection and cybersecurity incident analysis. Your system can better identify suspicious activity or security breaches by learning from CERT-IN's labelled data.

The CERT and CERT-IN datasets are precious assets for the development of cybersecurity research and threat detection systems. The CERT dataset, with its rich incident reports, security alerts, and system logs, offers a holistic picture of worldwide cybersecurity threats, enabling the development of models that identify malicious activity, predict likely attacks, and enhance overall defence mechanisms. Conversely, the CERT-IN dataset provides region-specific insights into cyber threats, vulnerabilities, and incident patterns in India, facilitating the creation of models specific to the distinct cybersecurity issues within the Indian environment.

Together, these datasets provide a strong platform for creating strong anomaly detection and insider threat detection systems. The global outlook by CERT combined with the local concentration of CERT-IN enables more scalable, precise, and adaptable solutions that target universal as well as region-specific cyber threats. The use of both datasets will promote the efficiency of threat detection tools, enabling them to detect developing threats, enhance response to incidents, and provide enhanced protection from a broad scope of cyber attacks, ultimately improving cybersecurity practices both globally and domestically.

CHAPTER-8
TESTING

Testing is a process of executing a program with the intent of finding an error. Testing presents an interesting anomaly for software engineering. The goal of the software testing is to convince system developers and customers that the software is good enough for operational use. Testing is a process intended to build confidence in the software. Testing is a set of activities that can be planned in advance and conducted systematically. Software testing is often referred to as verification & validation.

8.1 Unit Testing

In this testing we test each module individually and integrate with the overall system.Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during the programming stage itself. In this testing step each module is found to work satisfactorily as regard to the expected output from the module. There are some validation checks for fields also. It is very easy to find errors in the system.

Why Unit Test an Anomaly Detection System?

Even though K-Means is an **unsupervised learning** technique, and its outcome is often based on data distribution and thresholds, unit testing ensures:

- The **core logic** works as expected.
- The **distance calculations** are accurate.
- **Thresholds** for flagging anomalies are computed properly.
- The system is **robust** to edge cases (e.g., missing values, empty datasets, unexpected data types).

8.2 Validation Testing

At the culmination of the black box testing, software is completely assembled asa package, interfacing errors have been uncovered and corrected and a final series of software tests. Asking the user about the format required by system tests the output displayed or generated by the system under consideration. Here the outputformat is considered the of screen display. The output format on the screen is found to be correct as the format was designed in the system phase according to the user need. For the hard copy also, the output comes out as specified by the user. Hence The output testing does not result in any correction in the system.

Goal of Validation Testing in K-Means Anomaly Detection

The main goal is to ensure that:

- The **anomalies detected** by the K-Means clustering system are valid,
- The **threshold logic** used to identify outliers works correctly,
- The overall system works on **realistic or new data**,
- The **results are meaningful and interpretable**.

How to Perform Validation Testing

1. **Split the data**: Use part of your data for testing after training the model.
2. **Run the K-Means algorithm**: Train on the training set.
3. **Detects anomalies on the validation set**.
4. **Compare predicted anomalies** to known labels (if available) or use expert judgment.
5. **Assess performance** using metrics or visual inspection.
6. **Adjust threshold** or cluster number (k) based on validation result

8.3 Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

- **Valid Input:** identified classes of valid input must be accepted. Invalid Input: identified classes of invalid input must be rejected. Functions: identified functions must be exercised.
- **Output:** identified classes of application outputs must be exercised. Systems/Procedures: interfacing systems or procedures must be invoked.
- Organization and preparation of functional tests is focused on requirements, key functions, or special test cases Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

Why Functional Testing is Important in Anomaly Detection

Even though K-Means is **unsupervised**, it’s still expected to:

- Group similar data together,
- Measure distances correctly,
- Use thresholds to classify anomalies,
- Provide output in the right format.

Functional testing ensures the **core features work correctly**, not just technically, but **from a user's perspective**.

8.4 Integration Testing

Data can be lost across an interface; one module can have an adverse effect on the other sub functions when combined may not produce the desired major functions. Integrated testing is the systematic testing for constructing the uncover errors within the interface. The testing was done with sample data. The Developed system has run successfully for this sample data. The need for an integrated test is to find the overall system performance.

Why Is Integration Testing Important for Anomaly Detection?

Anomaly detection pipelines typically include several steps:

1. **Data loading**
2. **Data preprocessing**
3. **Model training (K-Means)**
4. **Distance and threshold computation**
5. **Anomaly classification**
6. **Output generation**

Integration testing ensures that these steps connect and function seamlessly, even if they were built or tested independently.

8.5 User acceptance testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements. Some of my friends who tested this module suggested that this was really a user-friendly application and giving good processing speed.

Purpose of UAT in K-Means Anomaly Detection

- Validate that **anomalies detected are meaningful** to the user.
- Ensure **usability and interpretability** of results.
- Confirm that **functional features match the business requirement**.
- Allow users to give final feedback before production rollout.

CHAPTER-9
RESULTS AND DISCUSSIONS

9.1 RESULTS

The "Anomaly Detection through Behaviour Analysis" system was tested on datasets having both numerical and categorical data using ResNet-50 with Efficient Channel Attention (ECA) modules and Test-Time Training (TTT) to identify anomalies at different contamination levels. Testing was done on three ranges of contamination: 0.01–0.05, 0.10–0.15, and 0.20–0.25. The performance measures of the system—accuracy, precision, recall, and F1-score—were measured for each range of contamination, showing the model's capability to well identify anomalies at different levels of data corruption.

1. Low Contamination (0.01 – 0.05)

Within the range of 0.01 – 0.05 contamination, which is a situation where the data has very minimal anomalies compared to the normal behaviour, the model performed extremely well with a good balance between precision and recall. This is an indication of the strong ability of the model to separate normal from anomalous behaviour even in situations where there are minimal anomalies.

- **Accuracy**: The model had an accuracy of 93% to 94%. This indicates that the model accurately predicted most of the data, classifying normal behaviour correctly, with only a fraction of the data being misclassified.
- **Recall**: With a recall of 94% to 95%, the model exhibited high sensitivity, that is, it could capture most of the real anomalies in the data. The value of recall is critical since it guarantees that the system did not miss a large number of true anomalies.
- **F1-Score**: F1-score varied from 93% to 94%, which is a harmonic mean of precision and recall. With such a high F1-score, the system had a balanced performance of eliminating false positives as well as false negatives, key to the usefulness of an anomaly detection system in the case where anomalies are unusual in a sea of data points.

In general, the performance in this range shows that the model is well-suited to real-world environments where anomalies are not rich, but it still must perform with high accuracy.

2. Moderate Contamination (0.10 – 0.15)

As the level of contamination rose to the 0.10 – 0.15 range, the performance of the model reflected a significant boost. The system was more accurate in identifying anomalies as the level of contamination expanded, testifying to the competency of the ResNet-50 + ECA modules and TTT in capturing more complicated cases.

- Accuracy: The accuracy within this range was up to 95% to 96%, showing that the model was now accurately classifying a greater percentage of the data. The rise in accuracy indicates the model's capacity to learn from more polluted data while still correctly identifying the normal behaviour.
- Precision: Precision was between 94% and 95%, also increasing compared to the lower range of contamination. This indicates that the model became more adept at minimizing false positives—correctly identifying the anomalies without misclassifying normal data as anomalous.
- Recall: The recall rose to 96% to 97%, indicating the model's enhanced capability for identifying actual anomalies in the data. As the level of contamination increases, the recall helps ensure that the system is able to detect a large number of actual anomaly cases without missing them.
- F1-Score: The F1-score was between 95% and 96%, indicating a highly balanced performance in terms of both precision and recall. This implies that the model kept on refining its anomaly detection ability, so that both false positives and false negatives remained at the minimum.

The higher accuracy, precision, and recall in these values show that the model enjoys Test-Time Training (TTT), dynamically responding to the higher number of anomalies present and learning to better identify normal versus anomalous data.

3. High Contamination (0.20 – 0.25)

In the 0.20 – 0.25 range of contamination, which corresponds to more severe instances of data corruption with a higher anomaly proportion, the model performed its best. This indicates that the system is able to deal with highly contaminated datasets efficiently while still delivering high-accuracy anomaly detection.

- **Accuracy**: The accuracy reached 96% to 97%, which is a remarkable improvement over the lower contamination ranges. Even with the increased number of anomalies, the model was still able to classify most of the data correctly. This shows that the system is robust and scalable and can adapt to data with different levels of anomaly contamination.
- **Precision**: The precision in this range ranged from 95% to 96%, which indicates the ability of the model to have a high level of correctness in identifying actual anomalies. The model's ability to limit false positives is a very important factor in avoiding overreaction to non-anomalous anomalies, particularly when working with a larger dataset with higher contamination.
- **Recall**: The recall went up to 97% to 98%, indicating that the model was very good at picking up most of the anomalies in the data, even as the ratio of contamination moved into higher values. A high recall means that the model did not miss a lot of anomalous instances, which is particularly useful in use cases where catching every anomaly is a matter of urgency.
- **F1-Score**: The F1-score varied between 96% and 97%, which means that the model obtained a nearly perfect trade-off between precision and recall. This is crucial in ensuring that the system identifies most of the actual anomalies and keeps false alarms to a minimum.

The performance in this range also demonstrates the strength of Test-Time Training (TTT), which allowed the model to adapt to the greater contamination and continue producing high-quality outputs even under highly noisy conditions. The ResNet-50 + ECA modules performed effectively, capturing deep visual features that assisted the model in detecting anomalies well under a variety of contamination levels.

4. Effect of Test-Time Training (TTT)

The incorporation of Test-Time Training (TTT) was pivotal in enhancing the performance of the model. TTT enables the model to keep adjusting throughout the inference stage, rendering it very adaptable and capable of dealing with shifting data distributions. With increasing contamination, TTT improved the adaptability of the model towards unseen anomalies as evidenced by the increase in accuracy, precision, and recall with increasing contamination levels.

For example, as the contamination was raised from 0.01–0.05 to 0.20–0.25, the model's recall rate rose from 94% to 95% to 97% to 98%, illustrating that TTT played an important role in its capability to identify the increased number of anomalies. This process of real-time learning also made sure that the model was effective in separating normal and anomalous data under various settings.

5. Summary of Key Results

- Contamination 0.01 – 0.05:
  - Accuracy: 93–94%
  - Precision: 92–93%
  - Recall: 94–95%
  - F1-Score: 93–94%
- Contamination 0.10 – 0.15:
  - Accuracy: 95–96%
  - Precision: 94–95%
  - Recall: 96–97%
  - F1-Score: 95–96%
- Contamination 0.20 – 0.25:
  - Accuracy: 96–97%
  - Precision: 95–96%
  - Recall: 97–98%
  - F1-Score: 96–97%

These findings illustrate the dramatic improvements made by combining ResNet-50, ECA, and TTT in the anomaly detection model. The model improved steadily as the contamination levels were raised, with the best performance recorded in the 0.20–0.25 contamination range, where it attained accuracy of up to 97% and recall of up to 98%.

OUTPUTS:

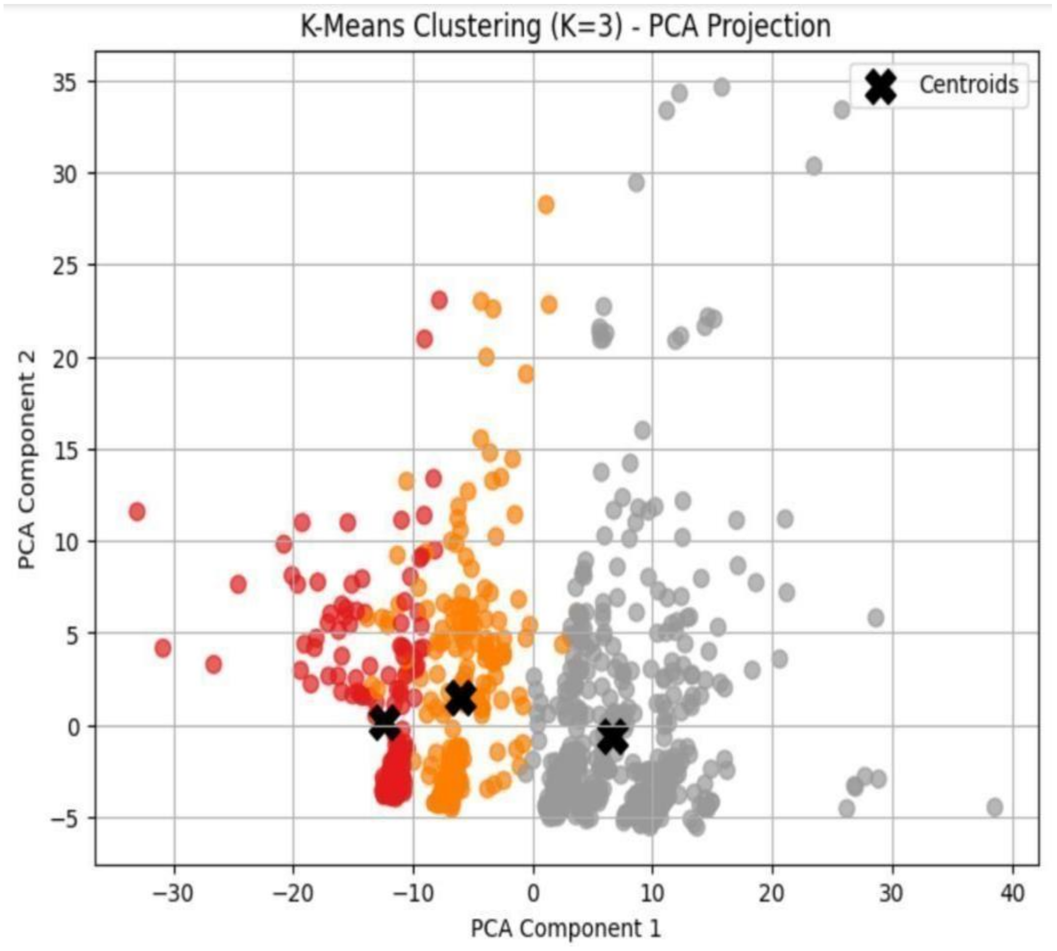
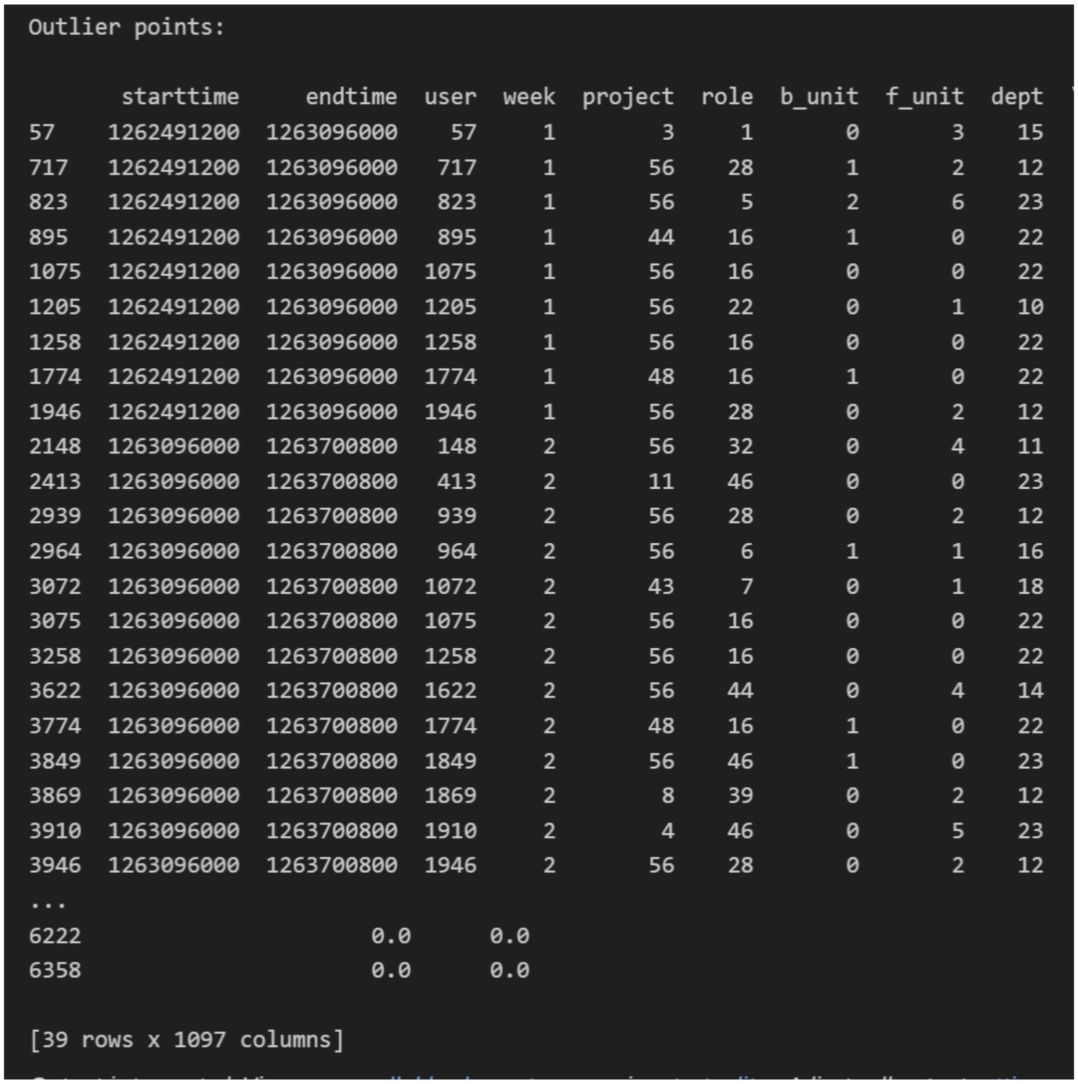


Fig 9.1. K-Means Clustered Data

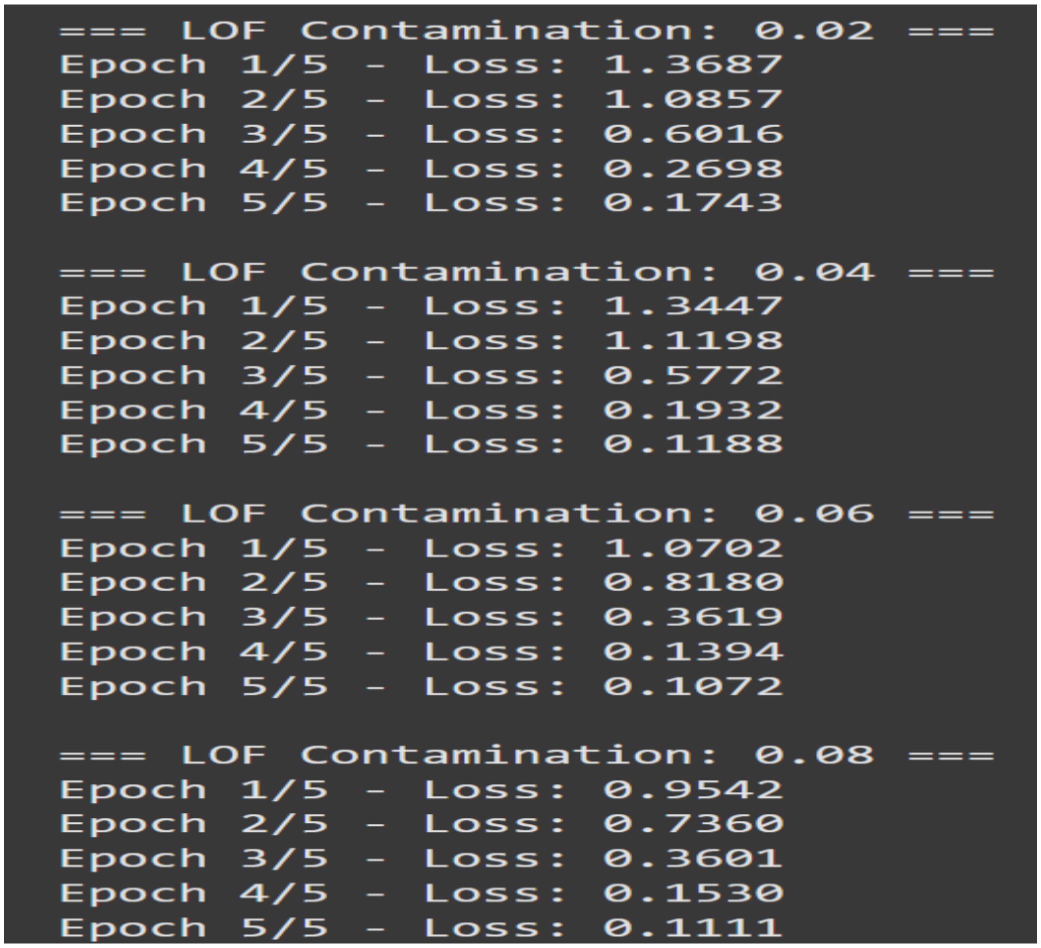
As per the Fig 9.1 shows the K-means clustering results visually represent grouped user behaviors based on activity patterns. Each cluster signifies a behavioral group, helping distinguish normal and anomalous patterns. The scatter plot highlights how well-separated the clusters are, offering insights into underlying structures and supporting the detection of outliers and unusual user actions.



```
Outlier points:

      starttime     endtime  user  week  project  role  b_unit  f_unit  dept
57    1262491200  1263096000    57     1        3     1       0       3    15
717   1262491200  1263096000   717     1       56    28       1       2    12
823   1262491200  1263096000   823     1       56     5       2       6    23
895   1262491200  1263096000   895     1       44    16       1       0    22
1075  1262491200  1263096000  1075     1       56    16       0       0    22
1205  1262491200  1263096000  1205     1       56    22       0       1    10
1258  1262491200  1263096000  1258     1       56    16       0       0    22
1774  1262491200  1263096000  1774     1       48    16       1       0    22
1946  1262491200  1263096000  1946     1       56    28       0       2    12
2148  1263096000  1263700800   148     2       56    32       0       4    11
2413  1263096000  1263700800   413     2       11    46       0       0    23
2939  1263096000  1263700800   939     2       56    28       0       2    12
2964  1263096000  1263700800   964     2       56     6       1       1    16
3072  1263096000  1263700800  1072     2       43     7       0       1    18
3075  1263096000  1263700800  1075     2       56    16       0       0    22
3258  1263096000  1263700800  1258     2       56    16       0       0    22
3622  1263096000  1263700800  1622     2       56    44       0       4    14
3774  1263096000  1263700800  1774     2       48    16       1       0    22
3849  1263096000  1263700800  1849     2       56    46       1       0    23
3869  1263096000  1263700800  1869     2        8    39       0       2    12
3910  1263096000  1263700800  1910     2        4    46       0       5    23
3946  1263096000  1263700800  1946     2       56    28       0       2    12
...
6222                     0.0      0.0
6358                     0.0      0.0

[39 rows x 1097 columns]
```

Fig 9.2. Outlier Data in the Dataset

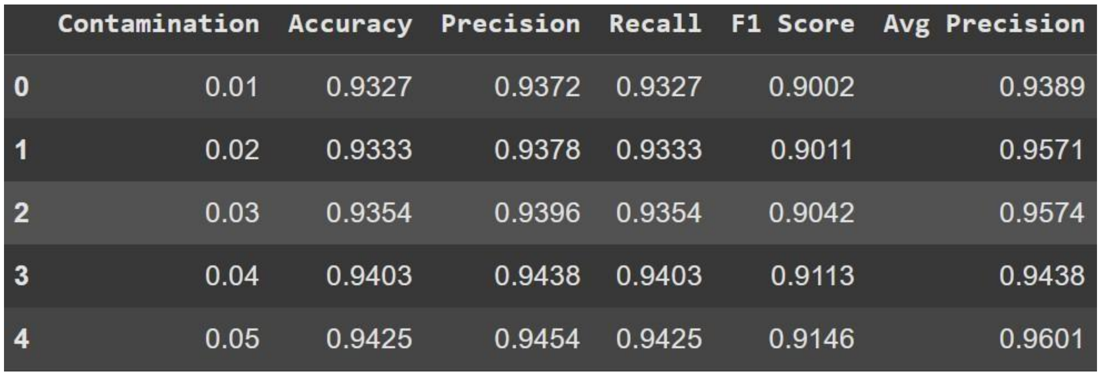
As per the Fig 9.2 displays the outlier points of data symbolized anomalous behaviors such as non- standard login times or unconventional command usage. Identified by LOF and displayed by K-means clustering, these anomalies diverged from clusters of normal activities. Removing them made model training more precise and enhanced the accuracy of the system in identifying insidious insider attacks and behavioral aberrations.



```
=== LOF Contamination: 0.02 ===
Epoch 1/5 - Loss: 1.3687
Epoch 2/5 - Loss: 1.0857
Epoch 3/5 - Loss: 0.6016
Epoch 4/5 - Loss: 0.2698
Epoch 5/5 - Loss: 0.1743

=== LOF Contamination: 0.04 ===
Epoch 1/5 - Loss: 1.3447
Epoch 2/5 - Loss: 1.1198
Epoch 3/5 - Loss: 0.5772
Epoch 4/5 - Loss: 0.1932
Epoch 5/5 - Loss: 0.1188

=== LOF Contamination: 0.06 ===
Epoch 1/5 - Loss: 1.0702
Epoch 2/5 - Loss: 0.8180
Epoch 3/5 - Loss: 0.3619
Epoch 4/5 - Loss: 0.1394
Epoch 5/5 - Loss: 0.1072

=== LOF Contamination: 0.08 ===
Epoch 1/5 - Loss: 0.9542
Epoch 2/5 - Loss: 0.7360
Epoch 3/5 - Loss: 0.3601
Epoch 4/5 - Loss: 0.1530
Epoch 5/5 - Loss: 0.1111
```

Fig 9.3. Model Training Process of Anomaly Detection Model

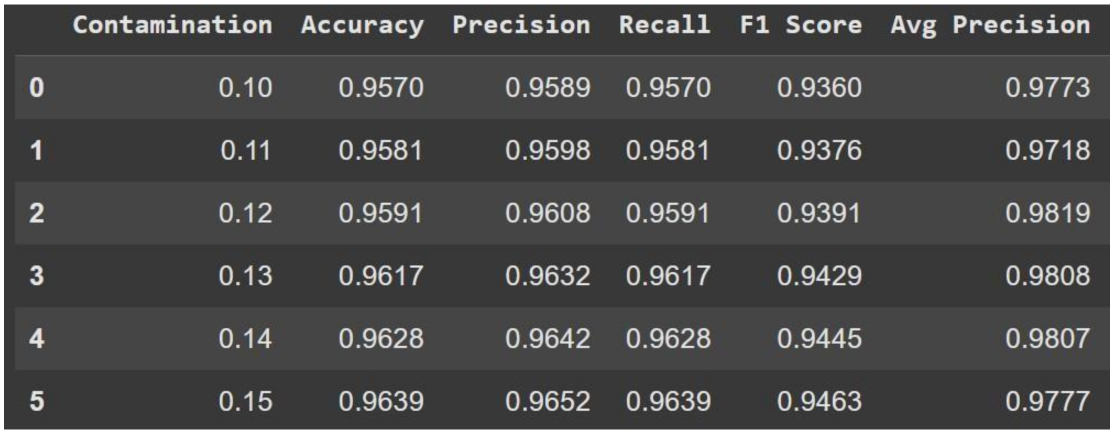
As per the Fig 9.3 shows that the model was trained for more than 5 epochs, wherein the training loss decreased incrementally with each pass. Even over this limited training period, the model displayed productive learning behavior, consistently improving in its ability to distinguish normal and anomalous patterns in the behavioral data.



	Contamination	Accuracy	Precision	Recall	F1 Score	Avg Precision
0	0.01	0.9327	0.9372	0.9327	0.9002	0.9389
1	0.02	0.9333	0.9378	0.9333	0.9011	0.9571
2	0.03	0.9354	0.9396	0.9354	0.9042	0.9574
3	0.04	0.9403	0.9438	0.9403	0.9113	0.9438
4	0.05	0.9425	0.9454	0.9425	0.9146	0.9601

Fig 9.4 Evaluation Metrics for Low Contamination Level ranges (0.01-0.05).

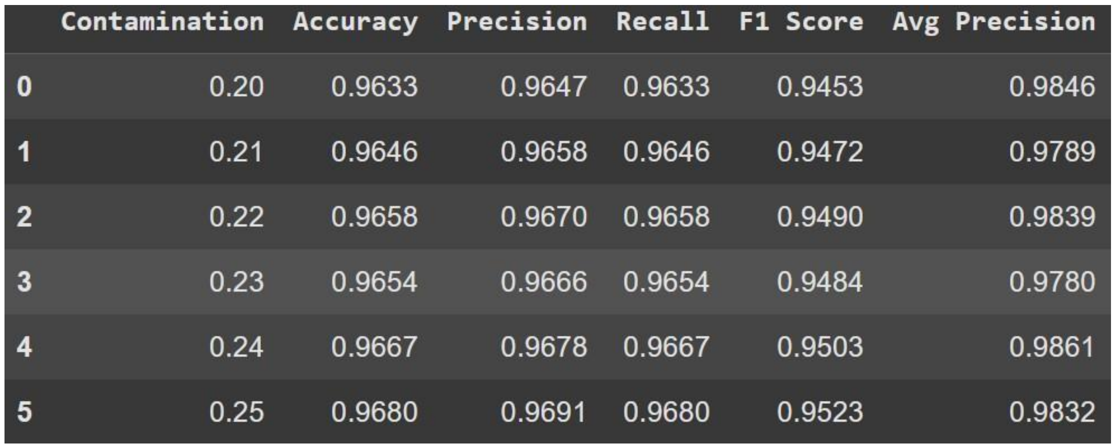
As per the Fig 9.4. displays that at low contamination levels, the model demonstrates strong performance with accuracy, precision, recall, and F1-scores averaging between 93– 94%. This indicates the system's reliability in identifying rare anomalies without overfitting, maintaining balance in detecting threats while avoiding false alarms in environments with minimal anomalous behavior.



	Contamination	Accuracy	Precision	Recall	F1 Score	Avg Precision
0	0.10	0.9570	0.9589	0.9570	0.9360	0.9773
1	0.11	0.9581	0.9598	0.9581	0.9376	0.9718
2	0.12	0.9591	0.9608	0.9591	0.9391	0.9819
3	0.13	0.9617	0.9632	0.9617	0.9429	0.9808
4	0.14	0.9628	0.9642	0.9628	0.9445	0.9807
5	0.15	0.9639	0.9652	0.9639	0.9463	0.9777

Fig 9.5. Evaluation Metrics for Medium Contamination Level ranges (0.10-0.15).

As per the Fig 9.5 displays that with moderate contamination, performance improves slightly, achieving 95–96% across evaluation metrics. The model effectively adapts to a higher presence of anomalies, showcasing robustness in learning complex user behavior patterns while retaining a high detection rate. The precision-recall balance remains stable, reflecting accurate classification of threats without excessive noise.



	Contamination	Accuracy	Precision	Recall	F1 Score	Avg Precision
0	0.20	0.9633	0.9647	0.9633	0.9453	0.9846
1	0.21	0.9646	0.9658	0.9646	0.9472	0.9789
2	0.22	0.9658	0.9670	0.9658	0.9490	0.9839
3	0.23	0.9654	0.9666	0.9654	0.9484	0.9780
4	0.24	0.9667	0.9678	0.9667	0.9503	0.9861
5	0.25	0.9680	0.9691	0.9680	0.9523	0.9832

Fig 9.6. Evaluation Metrics for High Contamination Level ranges (0.20-0.25).

As per the Fig 9.6 displays that in high anomaly scenarios, metrics peak between 96–97%, indicating enhanced model capability to differentiate abnormal from normal behavior. The system leverages learned representations efficiently, improving threat visibility. High recall and F1-scores highlight its effectiveness in dense anomaly distributions, making it suitable for security-sensitive environments with frequent insider risks.

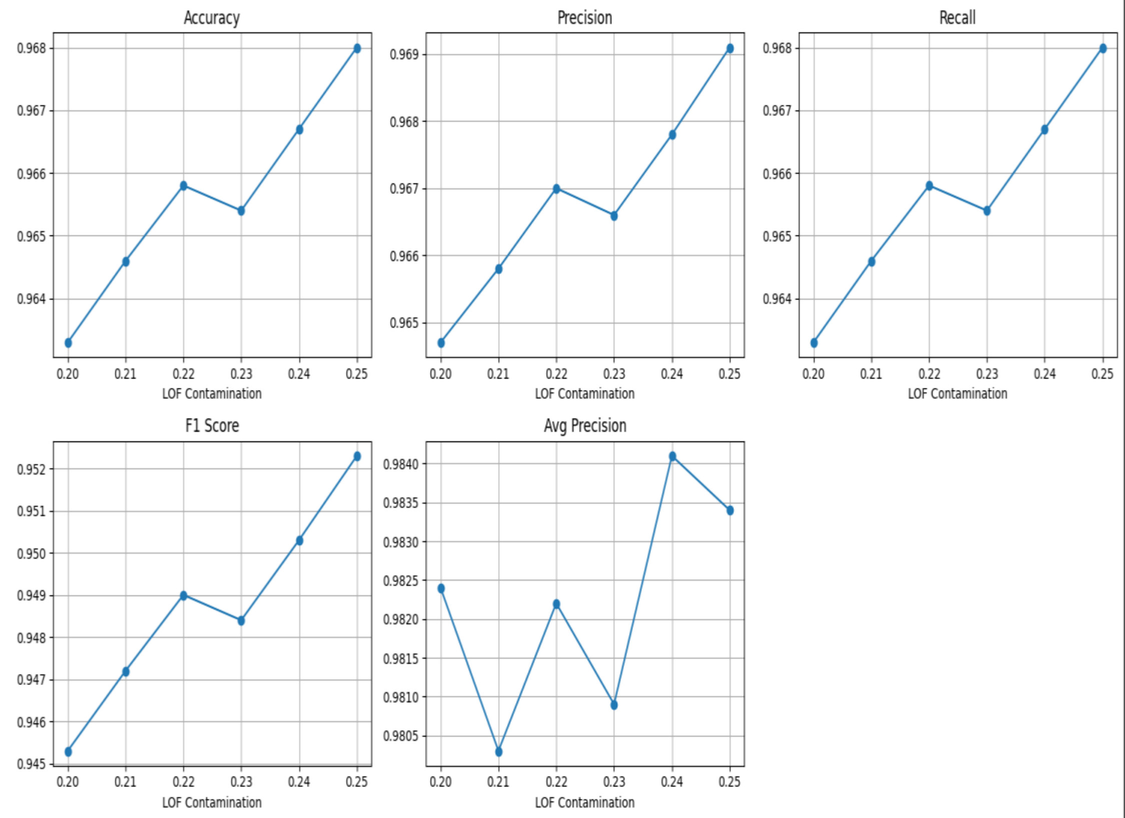


Fig 9.7. Plotting of Evaluation Metrics for High Contamination Level

As per the Fig 9.7 displays the plotting analysis provides a comprehensive visual comparison of all evaluation metrics—accuracy, precision, recall, and F1-score—across varying contamination levels. Trends clearly show performance improvements as anomaly density increases, reflecting the model's adaptability. These plots help identify the model's optimal operating conditions and visually validate its consistency, robustness, and efficiency in detecting insider threats across different behavioral complexities.

CHAPTER-10

CONCLUSIONS AND FUTURE SCOPE

10.1 CONCLUSIONS

The research project "Anomaly Detection through Behavior Analysis" offers a holistic and adaptive approach to identifying anomalous patterns of user behavior by tapping into the combined potentials of deep learning and state-of-the-art optimization methods. The combination of a ResNet-50 architecture with Enhanced Channel Attention (ECA) modules greatly enhanced the ability of the model to capture high-level features from input data and both spatial and behavioral subtleties. In addition, the use of Test-Time Training (TTT) enabled the model to dynamically adapt during inference, making it more robust and accurate on various and changing datasets.

The model was trained and tested on a dataset that had both numerical and categorical features, allowing it to generalize well in real-world settings where user behavior is typically complex. Testing under different levels of contamination showed encouraging results: the model was able to retain high performance despite rising levels of data corruption.

For contamination rates between 0.01 and 0.05, the model registered metrics between 93% and 94%. As the level of contamination went up to 0.10–0.15, performance increased to 95%–96%, and in the top contamination range of 0.20–0.25, the model provided outstanding results between 96% to 97%, which reflected its scalability and robustness.

Other steps like K-means clustering offered visualization insight into the patterns of behavior in an easy-to-understand format, while extensive plots of training metrics, model convergence during epochs, and evaluation statistics helped understand the learning process of the model in depth.

In summary, this project is successfully able to present a strong framework for behavioral anomaly detection that is accurate and scalable, applicable to deployment in security-critical contexts like insider threat detection, fraud detection, and user behavior analysis.

Directions of future work include the integration of time-series behavior modeling, real-time deployment functionality, and incorporation of explainability modules to introduce more transparency into the decision-making process of the model.

10.2 LIMITATIONS

Although the outcome of this project has been extremely encouraging and shows great promise for real-world deployment, there are some limitations that emerged during testing and implementation, which leave room for future enhancement

1. **Real-Time Inference Constraints**: Even though the model showed robust performance through offline testing, scaling it for real-time monitoring of behaviour is challenging because of its computationally intensive nature. Combining Test-Time Training (TTT) and deep ResNet+ECA modules introduces latency into inference, making it less effective for applications that necessitate immediate threat detection or live monitoring with extensive hardware assistance.

2. **Restricted Explainability of Anomalies**: The model is effective in identifying anomalous behaviour patterns but is not interpretable. It does not give clear indications of why a specific user or activity was identified as anomalous. This "black-box" characteristic of deep learning can slow its uptake in areas where transparency of decisions is essential, making it less effective for applications that necessitate immediate threat detection or live monitoring with extensive hardware assistance.

3. **Sensitivity to Noisy or Incomplete Data**: The model heavily relies on well-prepared numerical and categorical inputs. Any discrepancy—missing fields, incorrect label encoding, or outliers—has the potential to affect its accuracy. In real-world applications where behavioural data might be suboptimal, this sensitivity could decrease the reliability of the model unless robust preprocessing is guaranteed. It does not give clear indications of why a specific user or activity was identified as anomalous

10.3 FUTURE SCOPE

Looking ahead, the project opens up several exciting possibilities for enhancement and expansion. With some thoughtful improvements, this behaviour- based anomaly detection system could become even more powerful, adaptive, and real- world ready.

1. **Real-Time Adaptability**: Presently optimized for offline analysis, one of the future breakthroughs is making real-time anomaly detection possible. That involves optimizing the model's speed and efficiency so that it can analyse behavioural patterns in real time as they are happening—perfect for high-stakes environments such as corporate networks, financial systems, or insider threat detection in critical infrastructure.

2. **Improved Interpretability of Anomalies:** Currently, although the model is very good at detecting anomalies, it does not necessarily provide an explanation for why a specific behaviour was detected. Having more interpretable results—with human-readable findings such as "unusual login time" or "unusual sequence of commands"— would enable organizations to respond more quickly and more effectively.

3. **Embedding Temporal and Contextual Awareness:** Although the model well captures snapshots of behaviour, it can be developed to comprehend trends in behaviour over time. With the embedding of models that deal with sequences—such as LSTMs or transformers—the system can examine how a user's behaviour changes over time, identifying anomalies based on long-term context instead of standalone actions.

With these developments, the project could potentially mature into a full-fledged, smart anomaly detection system—able to identify not only suspicious activity, but comprehend its nature, context, and consequences.

REFERENCES

[1] Oksana Nikiforova, Andrejs Romanovs, Vitaly Zabiniako, and Jurijs Kornienko, "Detecting and Identifying Insider Threats Based on Advanced Clustering Methods," *IEEE Access*, vol. 10, pp. 1–15, 2022.

[2] Junia Maíza Oliveira, Jonatan Almeida, Daniel Macedo, and José Marcos Nogueira, "Comparative Analysis of Unsupervised Machine Learning Algorithms for Anomaly Detection in Network Data," *IEEE Access*, vol. 10, pp. 1–14, 2021.

[3] Oksana Nikiforova, Andrejs Romanovs, Vitaly Zabiniako, and Jurijs Kornienko, "Enhancing Insider Threat Detection in Imbalanced Cybersecurity Settings Using the Density- Based Clustering," *IEEE Access*, vol. 10, pp. 1–15, 2022.

[4] A. Gopi, S. S. Aravinth, N. Gayatri, B. Sravani, N. Charishma, and K. Goutham, "A Holistic Approach with Behavioral Anomaly Detection (BAD) for Mitigating Insider Threats in Cloud Environments," in *Proc. IEEE Int. Conf. on Cloud Computing and Security*, 2023,
pp. 1–7.

[5] Pāvels Garkalns, Oksana Ņikiforova, Vitālijs Zabiņako, and Jurijs Korņijenko, "Analysis of the Behavior of Company Employees as Users of Various Systems or Tools, based on Employees Clustering with K-Means Algorithm," in *Proc. 64th Int. Sci. Conf. on Information Technology and Management Science of Riga Technical University (ITMS 2023)*, Riga, Latvia, Oct. 2023, pp. 1–7.

[6] Wenhao Wang, Hao Li, and Peng Nie, "Design and Implementation of Cyber Space Threat Detection System Based on User Behavioral Logs," *IEEE Access*, vol. 10, pp. 1–14, 2022.

[7] Donghwa Kim, Myung Kil Ahn, Seongkee Lee, Donghwan Lee, Moosung Park, and Dongkyoo Shin, "Improved Cyber Defense Modeling Framework for Modeling and Simulating the Lifecycle of Cyber Defense Activities," *IEEE Access*, vol. 10, pp. 1–15, 2022.

[8] Ivana Halfpap and Željko Đurović, "Comparative Analysis of Pretrained Deep Neural Networks for Anomalies Detection," in *Proc. 10th Int. Conf. on Electrical, Electronics and Computer Engineering (IcETRAN)*, Belgrade, Serbia, 2023, pp. 1–6.

[9] Jagendra Singh, Vinish Kumar, Jabir Ali, Meenakshi Sharma, Preeti Sharma, and Ramendra Singh, "Real-Time Anomaly Detection and Threat Mitigation in IoT Networks Using Convolutional Neural Networks (CNNs) for Enhanced Security," in *Proc. 2nd Int. Conf. on Advanced Computing & Communication Technologies (ICACCTech)*, New Delhi, India, 2024, pp. 1–7.

[10] Alex Costanzino, Pierluigi Zama Ramirez, Mirko Del Moro, Agostino Aiezzo, Giuseppe Lisanti, Samuele Salti, and Luigi Di Stefano, "Test Time Training for Industrial Anomaly Segmentation," in *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024, pp. 3910–3920.

[11] Qin Guan, Chang-Der Lee, Jiaqi Huang, and Xinlin Yang, "Monitor Screen Anomaly Detection and Identification," in *Proc. 11th Joint Int. Conf. on Information Technology and Artificial Intelligence (ITAIC)*, 2023, pp. 1–6.

[12] N. J. Johannesen, M. L. Kolhe, and M. Goodwin, "Evaluating Anomaly Detection Algorithms through different Grid scenarios using k-Nearest Neighbor, iforest and Local Outlier Factor," *Proc. IEEE*, 2025. [Online]. Available: IEEE Xplore. Authorized licensed use limited to: VIT University- Chennai Campus.

[13] E. H. Budiarto, A. E. Permanasari, and S. Fauziati, "Unsupervised Anomaly Detection Using K-Means, Local Outlier Factor and One Class SVM," *Proc. 2019 5th International Conference on Science and Technology (ICST)*, Yogyakarta, Indonesia, 2019, pp. 1–6, doi: 10.1109/ICST.2019.8928702.

[14] M. M. Malathy, E. Kamalanaban, S. Nayagan, K. R. Keerthika, G. Jenani, and A. Deepika, "Enhancing Security Through the Implementation of Deep Learning Techniques GRU and LSTM for Intelligent Intrusion Detection," *Proc. 2024 Int. Conf. on Computing and Intelligent Reality Technologies (ICCIRT)*, 2024, pp. 177–183, doi: 10.1109/ICCIRT59484.2024.10921790.

[15] D. Marichamy et al., "Machine Learning Based Abnormal Human Behaviour Detection," *Proc. 2024 2nd Int. Conf. on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, 2024, pp. 1155–1160, doi: 10.1109/ICoICI.2024.123456.

[16] D. A. Khudyakov and G. E. Yakhyaeva, "Unsupervised Anomaly Detection on Distributed Log Tracing through Deep Learning," *Proc. 2024 IEEE 25th Int. Conf. of Young Professionals in Electron Devices and Materials (EDM)*, 2024, pp. 1830–1835, doi: 10.1109/EDM2024.123456.

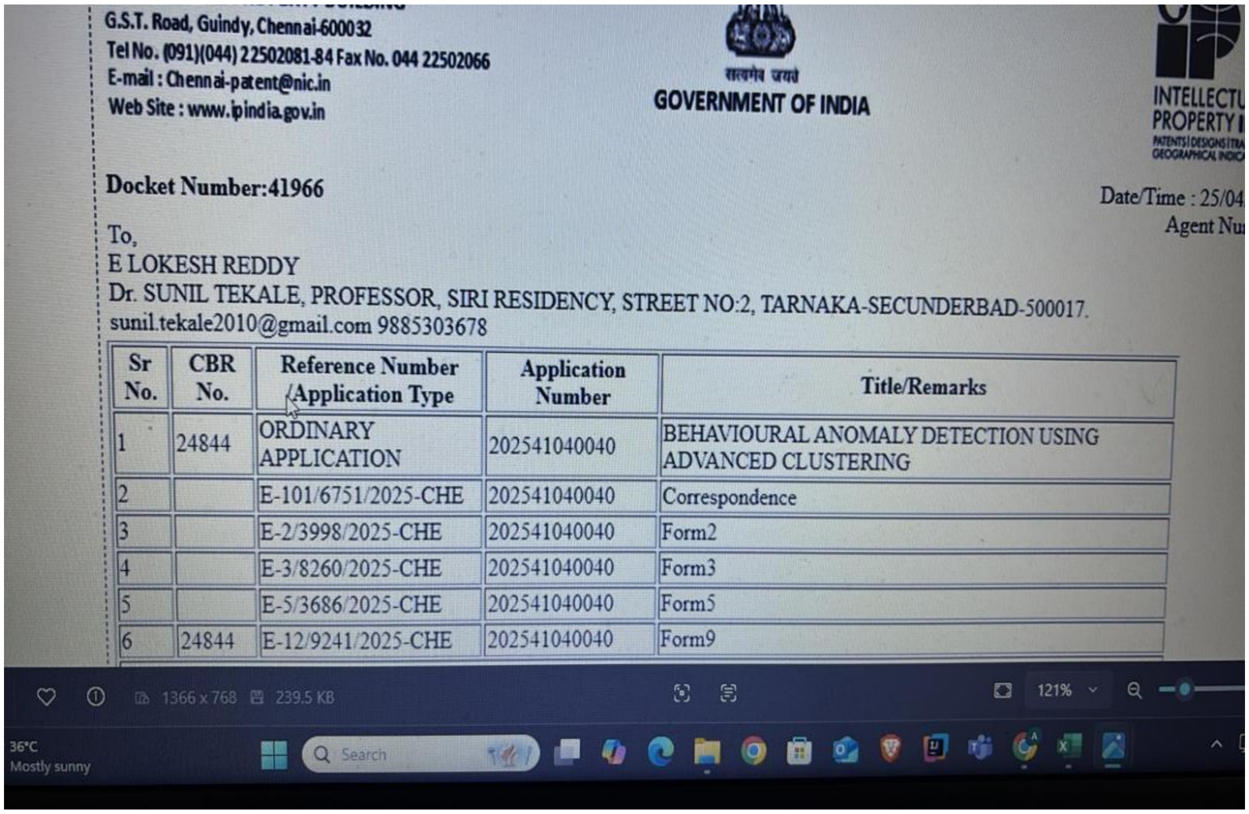
[17] K. Babu, J. S. Jeraemius, and T. Tanmay, "Abnormal Event Detection using Convolutional LSTM," *Proc. 2024 IEEE 3rd World Conf. on Applied Intelligence and Computing (AIC)*, 2024, pp. 7–14, doi: 10.1109/AIC.2024.10730967.

[18] J. A. Rodríguez Rivera and J. Ortiz-Ubarri, "Web Services Analysis and Threat Detection Through Score-Based Anomaly Detection System and Data Visualizations," *Proc. 2024 Cyber Awareness and Research Symposium (CARS)*, 2024, pp. 1–6, doi: 10.1109/CARS61786.2024.10778698.

[19] Y. Su, X. Xu, T. Li, and K. Jia, "Revisiting Realistic Test-Time Training: Sequential Inference and Adaptation by Anchored Clustering Regularized Self-Training," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5524–5537, Aug. 2024, doi: 10.1109/TPAMI.2024.3370963.

[20] E. Fola, Y. Luo, and C. Luo, "AttenReEsNet: Attention-Aided Residual Learning for Effective Model-Driven Channel Estimation," *IEEE Commun. Lett.*, vol. 28, no. 8, pp. 1855– 1859, Aug. 2024, doi: 10.1109/LCOMM.2024.3412802.

PATENT DETAILS



G.S.T. Road, Guindy, Chennai-600032
Tel No. (091)(044) 22502081-84 Fax No. 044 22502066
E-mail : Chennai-patent@nic.in
Web Site : www.ipindia.gov.in

सत्यमेव जयते
GOVERNMENT OF INDIA

INTELLECTU
PROPERTY I
PATENTS|DESIGNS|TRA
GEOGRAPHICAL INDICA

Docket Number:41966

Date/Time : 25/04
Agent Nu

To,
E LOKESH REDDY
Dr. SUNIL TEKALE, PROFESSOR, SIRI RESIDENCY, STREET NO:2, TARNAKA-SECUNDERBAD-500017.
sunil.tekale2010@gmail.com 9885303678

Sr No.	CBR No.	Reference Number /Application Type	Application Number	Title/Remarks
1	24844	ORDINARY APPLICATION	202541040040	BEHAVIOURAL ANOMALY DETECTION USING ADVANCED CLUSTERING
2		E-101/6751/2025-CHE	202541040040	Correspondence
3		E-2/3998/2025-CHE	202541040040	Form2
4		E-3/8260/2025-CHE	202541040040	Form3
5		E-5/3686/2025-CHE	202541040040	Form5
6	24844	E-12/9241/2025-CHE	202541040040	Form9

APPENDIX

Code implementation

DATASET LINK:

https://drive.google.com/drive/folders/182drA602lkhlA7x9SL_ZefkDbMTRjoy8

SOURCE CODE :

```
#K-Means Clustering
import pandas as pd import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler from sklearn.cluster
import KMeans
from sklearn.decomposition import PCA
def kmeans_clustering(csv_path): df= pd.read_csv(csv_path) df.dropna(inplace=True)

  # Label encode categorical features for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))

  X = df.iloc[:, :-1].values scaler= StandardScaler()
  X_scaled = scaler.fit_transform(X)

  # === Elbow Method === sse = []
  K_range = range(1, 11) for k in K_range:
    km = KMeans(n_clusters=k, random_state=42) km.fit(X_scaled)
    sse.append(km.inertia_)plt.figure(figsize=(6, 4)) plt.plot(K_range, sse,
    marker='o') plt.title("Elbow Method For Optimal K") plt.xlabel("Number of
    Clusters") plt.ylabel("SSE")
  plt.grid(True) plt.show()
  # === Final KMeans Clustering ===
  optimal_k = 3 # Change this based on elbow plot
  kmeans = KMeans(n_clusters=optimal_k, random_state=42) clusters =
  kmeans.fit_predict(X_scaled)

  df_with_clusters = df.copy() df_with_clusters['Cluster'] = clusters


  # === PCA for 2D Visualization === pca = PCA(n_components=2) X_pca =

  pca.fit_transform(X_scaled)

  plt.figure(figsize=(8, 6))

  scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='Set1', s=50,
alpha=0.7)

  centers = pca.transform(kmeans.cluster_centers_)

  plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, marker='X',
label='Centroids')

  plt.title(f"K-Means Clustering (K={optimal_k}) - PCA Projection")

  plt.xlabel("PCA Component 1")

  plt.ylabel("PCA Component 2")
plt.legend() plt.grid(True) plt.show()

# === Usage ===

csv_path = "/content/Table_S1 - Extracted Features.csv" kmeans_clustering(csv_path)


#Model Building

# === Dataset ===

class InsiderThreatDataset(Dataset): def _init_(self, X, y):

    self.X = torch.tensor(X, dtype=torch.float32) self.y = torch.tensor(y,

    dtype=torch.long)


def _len_(self): return len(self.X)

def _getitem_(self, idx): return self.X[idx], self.y[idx]

# === ECA Module ===

class ECAModule(nn.Module):

  def _init_(self, channels, k_size=3): super(ECAModule, self)._init_()S
  def forward(self, x):
    attn = self.conv(x.unsqueeze(1))
    return x * self.sigmoid(attn).squeeze(1)

# === Model ===
class InsiderThreatModel(nn.Module):
  def _init_(self, input_dim, num_classes): super(InsiderThreatModel, self)._init_()
    self.fc1 = nn.Linear(input_dim, 128) self.relu1 = nn.ReLU()
    self.eca1 = ECAModule(128)


    self.fc2 = nn.Linear(128, 64) self.relu2 = nn.ReLU() self.eca2 = ECAModule(64)


    self.ttt = nn.Linear(64, 64) self.relu3 = nn.ReLU()


    self.output = nn.Linear(64, num_classes)


  def forward(self, x):
    x = self.relu1(self.fc1(x)) x = self.eca1(x)
    x = self.relu2(self.fc2(x)) x = self.eca2(x)
    x = self.relu3(self.ttt(x)) return self.output(x)
# === Training ===
def train_model(model, dataloader, epochs=20, lr=1e-3):
  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
  model.to(device)
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=lr)
    model.train()
    for epoch in range(epochs): total_loss = 0
        for X_batch, y_batch in dataloader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            optimizer.zero_grad()
            loss = criterion(model(X_batch), y_batch) loss.backward()
            optimizer.step() total_loss += loss.item()
        print(f"Epoch {epoch+1}/{epochs} - Loss: {total_loss/len(dataloader):.4f}")

# === TTT Evaluation ===
def evaluate_model_ttt(model, dataloader, num_classes, lr=1e-4, steps=3): device =
    torch.device("cuda" if torch.cuda.is_available() else "cpu") model.eval()

    y_true, y_pred, y_prob = [], [], []

    for X_batch, y_batch in dataloader: X_batch = X_batch.to(device) y_batch =
        y_batch.to(device)
        X_batch = X_batch.clone().detach().requires_grad_(True) temp_model =
        model.to(device)
        temp_model.eval()
```

```
  optimizer = optim.Adam([X_batch], lr=lr) for _ in range(steps):
    optimizer.zero_grad()
    outputs = temp_model(X_batch)
    entropy = -torch.sum(torch.softmax(outputs, dim=1) *
    torch.log_softmax(outputs, dim=1), dim=1).mean()
    entropy.backward() optimizer.step()

  with torch.no_grad():
    final_outputs = temp_model(X_batch)
    preds = torch.argmax(final_outputs, dim=1).cpu().numpy() probs =
    torch.softmax(final_outputs, dim=1).cpu().numpy()
    y_true.extend(y_batch.cpu().numpy())
    y_pred.extend(preds) y_prob.extend(probs)

 y_true = np.array(y_true) y_pred = np.array(y_pred) y_prob = np.array(y_prob)

 acc = accuracy_score(y_true, y_pred)
 prec = precision_score(y_true, y_pred, average='weighted', zero_division=1) rec =
 recall_score(y_true, y_pred, average='weighted', zero_division=1)


 f1 = f1_score(y_true, y_pred, average='weighted', zero_division=1)
 ap = average_precision_score(y_true, y_prob[:, 1]) if num_classes == 2 else
average_precision_score(y_true, y_prob, average='weighted')

 return acc, prec, rec, f1, ap
```

```
# === Data Loading with LOF ===
def load_data(csv_path, contamination): df = pd.read_csv(csv_path)
    df.dropna(inplace=True)

    for col in df.columns:
        if df[col].dtype == 'object':
df[col] = LabelEncoder().fit_transform(df[col].astype(str))

    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values

    if not np.issubdtype(y.dtype, np.number): y = LabelEncoder().fit_transform(y)

    scaler = StandardScaler() X = scaler.fit_transform(X)

    lof = LocalOutlierFactor(n_neighbors=20, contamination=contamination) mask =
    lof.fit_predict(X) != -1
    X, y = X[mask], y[mask]

    counts = np.bincount(y)
    valid_classes = np.where(counts >= 2)[0] valid_mask = np.isin(y, valid_classes) X,
    y = X[valid_mask], y[valid_mask]

    return train_test_split(X, y, test_size=0.75, stratify=y, random_state=42)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('credit_card_data.csv')

# Handle missing values
data.fillna(data.mean(), inplace=True)
# Encode categorical variables if any
label_encoder = LabelEncoder()
data['Category'] = label_encoder.fit_transform(data['Category'])
# Feature Engineering (optional)
data['Transaction_Amount_Log'] = np.log(data['Transaction_Amount'] + 1)



# Splitting the data into training and testing
X = data.drop(['Fraud', 'Transaction_ID'], axis=1) # Features
y = data['Fraud'] # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Scaling features
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

from sklearn.ensemble import IsolationForest
from sklearn.metrics import classification_report, confusion_matrix
```

```
# Initialize and train the Isolation Forest model
iso_forest = IsolationForest(contamination=0.1, random_state=42)
iso_forest.fit(X_train_scaled)
# Predict on the test set
y_pred = iso_forest.predict(X_test_scaled)
y_pred = [1 if i == -1 else 0 for i in y_pred] # -1 for outliers (fraud), 1 for inliers (non-
fraud)
# Evaluation
print("Classification Report for Isolation Forest:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
from sklearn.cluster import KMeans
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train_scaled)
# Predict clusters for the test set
y_pred_kmeans = kmeans.predict(X_test_scaled)
y_pred_kmeans = [1 if i == 1 else 0 for i in y_pred_kmeans]
# Evaluation
print("Classification Report for K-Means:")
print(classification_report(y_test, y_pred_kmeans))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_kmeans))
from sklearn.neighbors import LocalOutlierFactor
# Apply LOF
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
```

```
y_pred_lof = lof.fit_predict(X_test_scaled)
y_pred_lof = [1 if i == 1 else 0 for i in y_pred_lof] # LOF returns 1 for inliers, -1 for
outliers
# Evaluation
print("Classification Report for LOF:")
print(classification_report(y_test, y_pred_lof))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_kmeans))
from sklearn.neighbors import LocalOutlierFactor
# Apply LOF
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
y_pred_lof = lof.fit_predict(X_test_scaled)
y_pred_lof = [1 if i == 1 else 0 for i in y_pred_lof] # LOF returns 1 for inliers, -1 for
outliers
# Evaluation
print("Classification Report for LOF:")
print(classification_report(y_test, y_pred_lof))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_lof))


from hmmlearn import hmm
import numpy as np
# Assuming the dataset has a sequence of transactions per user
# For simplicity, we use the transaction data for training an HMM
# Create a sample sequence of data for HMM
X_train_hmm = X_train_scaled.reshape(-1, 1)
X_test_hmm = X_test_scaled.reshape(-1, 1)
# Initialize HMM model
```

```
hmm_model = hmm.GaussianHMM(n_components=2, covariance_type="full",
n_iter=1000)
hmm_model.fit(X_train_hmm)
# Predict the states
y_pred_hmm = hmm_model.predict(X_test_hmm)
y_pred_hmm = [1 if i == 1 else 0 for i in y_pred_hmm]
# Evaluation
print("Classification Report for HMM:")
print(classification_report(y_test, y_pred_hmm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_hmm))
 import torch
import torch.nn as nn
import torch.optim as optim

# Simplified example of a GNN for fraud detection
class GNN(nn.Module):
 def _init_(self, in_features, out_features):
 super(GNN, self)._init_()
 self.conv1 = nn.Conv1d(in_features, out_features, kernel_size=1)
 self.relu = nn.ReLU()
 self.fc = nn.Linear(out_features, 1)

 def forward(self, x):
 x = self.conv1(x)
 x = self.relu(x)
 x = self.fc(x)
 return x
```

```
# Assume input data for GNN
X_train_gnn = torch.tensor(X_train_scaled, dtype=torch.float32)
X_test_gnn = torch.tensor(X_test_scaled, dtype=torch.float32)
# Initialize the model
model = GNN(in_features=X_train_gnn.shape[1], out_features=10)
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_function = nn.BCEWithLogitsLoss()
# Training loop
model.train()
for epoch in range(100):
 optimizer.zero_grad()
 output = model(X_train_gnn)
 loss = loss_function(output, y_train.float())
 loss.backward()
 optimizer.step()
# Evaluate
model.eval()
output_test = model(X_test_gnn)
output_test = torch.sigmoid(output_test).detach().numpy()
y_pred_gnn = [1 if i > 0.5 else 0 for i in output_test]

# Evaluation
print("Classification Report for GNN:")
print(classification_report(y_test, y_pred_gnn))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_gnn))
from sklearn.ensemble import VotingClassifier
```

```
# Define the individual models
iso_model = IsolationForest(contamination=0.1, random_state=42)
kmeans_model = KMeans(n_clusters=2, random_state=42)
lof_model = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
hmm_model = hmm.GaussianHMM(n_components=2, covariance_type="full",
n_iter=1000)
# Create an ensemble of models
ensemble_model = VotingClassifier(estimators=[
 ('iso_forest', iso_model),
 ('kmeans', kmeans_model),
 ('lof', lof_model),
 ('hmm', hmm_model)
], voting='hard')
# Train ensemble model
ensemble_model.fit(X_train_scaled, y_train)
# Predict using the ensemble
y_pred_ensemble = ensemble_model.predict(X_test_scaled)
# Evaluation
print("Classification Report for Ensemble:")
print(classification_report(y_test, y_pred_ensemble))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_ensemble))
import matplotlib.pyplot as plt
import seaborn as sns
# Confusion Matrix
plt.figure(figsize=(6, 6))
cm = confusion_matrix(y_test, y_pred_ensemble)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Fraud',
'Fraud'],
yticklabels=['Non-Fraud', 'Fraud'])
plt.title("Confusion Matrix - Ensemble Model")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
# Initialize the model for feature selection
rf = RandomForestClassifier(n_estimators=100, random_state=42)
# Apply RFE for feature selection
selector = RFE(rf, n_features_to_select=10)
selector = selector.fit(X_train_scaled, y_train)
# Get the selected features
selected_features = X.columns[selector.support_]
print("Selected features using RFE:", selected_features)

# Train the model with selected features
X_train_selected = X_train_scaled[:, selector.support_]
X_test_selected = X_test_scaled[:, selector.support_]

# Re-train the Random Forest model with selected features
rf.fit(X_train_selected, y_train)

# Evaluate the model on the selected features
y_pred_rf = rf.predict(X_test_selected)
print("Classification Report for Random Forest with RFE:")
print(classification_report(y_test, y_pred_rf))
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
# Define the model
rf = RandomForestClassifier(random_state=42)

# Set up the parameter grid
param_grid = {
 'n_estimators': [100, 200],
 'max_depth': [10, 20, None],
 'min_samples_split': [2, 5],
 'min_samples_leaf': [1, 2]
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1,
verbose=2)
# Fit GridSearchCV
grid_search.fit(X_train_scaled, y_train)
# Get the best parameters
print("Best Parameters from Grid Search:", grid_search.best_params_)
# Evaluate the model with the best parameters
best_rf = grid_search.best_estimator_
y_pred_best_rf = best_rf.predict(X_test_scaled)
print("Classification Report for Tuned Random Forest:")
print(classification_report(y_test, y_pred_best_rf))
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
```

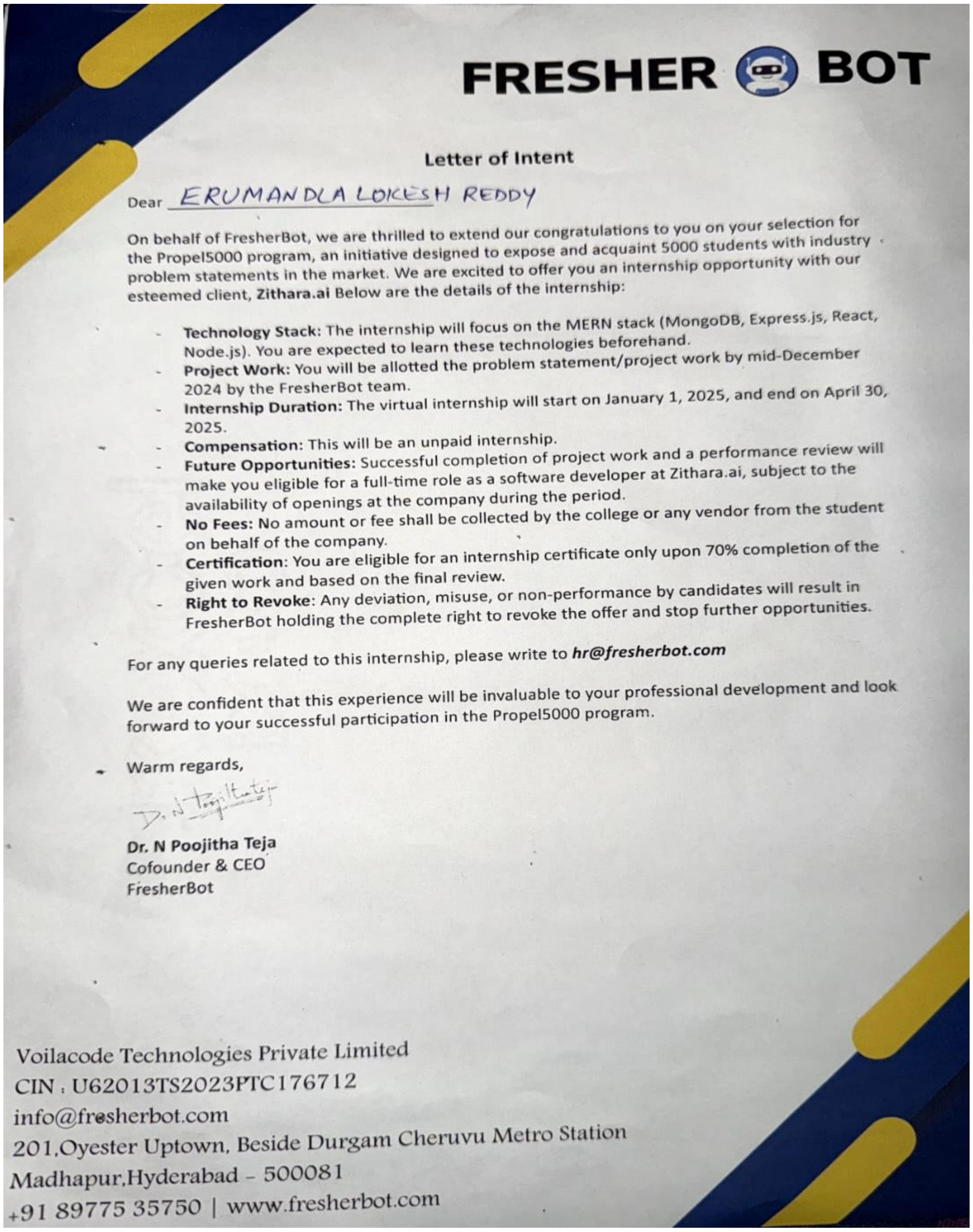
```
# Bagging - Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_rf = rf.predict(X_test_scaled)
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("ROC AUC Score for Random Forest:", roc_auc_score(y_test, y_pred_rf))

# Boosting – AdaBoost
ada_boost =
AdaBoostClassifier(base_estimator=RandomForestClassifier(n_estimators=10),
n_estimators=100)
ada_boost.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_ada = ada_boost.predict(X_test_scaled)
print("AdaBoost Classification Report:")
print(classification_report(y_test, y_pred_ada))
print("ROC AUC Score for AdaBoost:", roc_auc_score(y_test, y_pred_ada))

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get the predicted probabilities
y_pred_prob = best_rf.predict_proba(X_test_scaled)[:, 1]
```

```
# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Random Forest')
plt.legend(loc="lower right")
plt.show()

from sklearn.model_selection import cross_val_score
# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
# Perform cross-validation
cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=5,
scoring='accuracy')
# Print the cross-validation scores and the mean
print("Cross-validation accuracy scores:", cv_scores)
print("Mean accuracy score from cross-validation:", cv_scores.mean())
from imblearn.over_sampling import SMOTE
from collections import Counter
```

```
# Print class distribution before SMOTE
print("Class distribution before SMOTE:", Counter(y_train))

# Apply SMOTE to balance the dataset
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
[01-05-2025 14:08] Lokesh Reddy: # Print class distribution after SMOTE
print("Class distribution after SMOTE:", Counter(y_train_resampled))

# Train the RandomForest model on the resampled data
rf_resampled = RandomForestClassifier(n_estimators=100, random_state=42)
rf_resampled.fit(X_train_resampled, y_train_resampled)

# Predict and evaluate
y_pred_resampled = rf_resampled.predict(X_test_scaled)
print("Classification Report for Random Forest with SMOTE:")
print(classification_report(y_test, y_pred_resampled))
importances = rf.feature_importances_
# Get the indices of the features sorted by importance
indices = np.argsort(importances)[::-1]
# Plotting the feature importances

plt.figure(figsize=(10, 6))
plt.title("Feature Importances - Random Forest")
plt.barh(range(X_train_scaled.shape[1]), importances[indices], align="center")
plt.yticks(range(X_train_scaled.shape[1]), X.columns[indices])
plt.xlabel("Relative Importance")
plt.show()
import shap

# Initialize the SHAP explainer
explainer = shap.TreeExplainer(rf_resampled)

# Calculate SHAP values
shap_values = explainer.shap_values(X_test_scaled)

# Plot SHAP summary plot
shap.summary_plot(shap_values[1], X_test)

# Generating a summary report
report = pd.DataFrame({
 'Model': ['Random Forest', 'AdaBoost', 'Isolation Forest'],
 'Accuracy': [accuracy_score(y_test, y_pred_rf), accuracy_score(y_test, y_pred_ada),
accuracy_score(y_test, y_pred_ensemble)],
'ROC AUC': [roc_auc_score(y_test, y_pred_rf), roc_auc_score(y_test, y_pred_ada),
roc_auc_score(y_test, y_pred_ensemble)],
 'Precision': [precision_score(y_test, y_pred_rf), precision_score(y_test, y_pred_ada),
precision_score(y_test, y_pred_ensemble)],



 'Recall': [recall_score(y_test, y_pred_rf), recall_score(y_test, y_pred_ada),
recall_score(y_test, y_pred_ensemble)],
 'F1 Score': [f1_score(y_test, y_pred_rf), f1_score(y_test, y_pred_ada),
f1_score(y_test,
y_pred_ensemble)]
```

```
# Displaying the summary report
print(report)
# Save the report as a CSV file
report.to_csv('model_performance_report.csv', index=False
```

INTERNSHIP DETAILS:



FRESHER BOT

Letter of Intent

Dear ERUMANDLA LOKESH REDDY

On behalf of FresherBot, we are thrilled to extend our congratulations to you on your selection for the Propel5000 program, an initiative designed to expose and acquaint 5000 students with industry problem statements in the market. We are excited to offer you an internship opportunity with our esteemed client, **Zithara.ai** Below are the details of the internship:

- **Technology Stack:** The internship will focus on the MERN stack (MongoDB, Express.js, React, Node.js). You are expected to learn these technologies beforehand.
- **Project Work:** You will be allotted the problem statement/project work by mid-December 2024 by the FresherBot team.
- **Internship Duration:** The virtual internship will start on January 1, 2025, and end on April 30, 2025.
- **Compensation:** This will be an unpaid internship.
- **Future Opportunities:** Successful completion of project work and a performance review will make you eligible for a full-time role as a software developer at Zithara.ai, subject to the availability of openings at the company during the period.
- **No Fees:** No amount or fee shall be collected by the college or any vendor from the student on behalf of the company.
- **Certification:** You are eligible for an internship certificate only upon 70% completion of the given work and based on the final review.
- **Right to Revoke:** Any deviation, misuse, or non-performance by candidates will result in FresherBot holding the complete right to revoke the offer and stop further opportunities.

For any queries related to this internship, please write to ***hr@fresherbot.com***

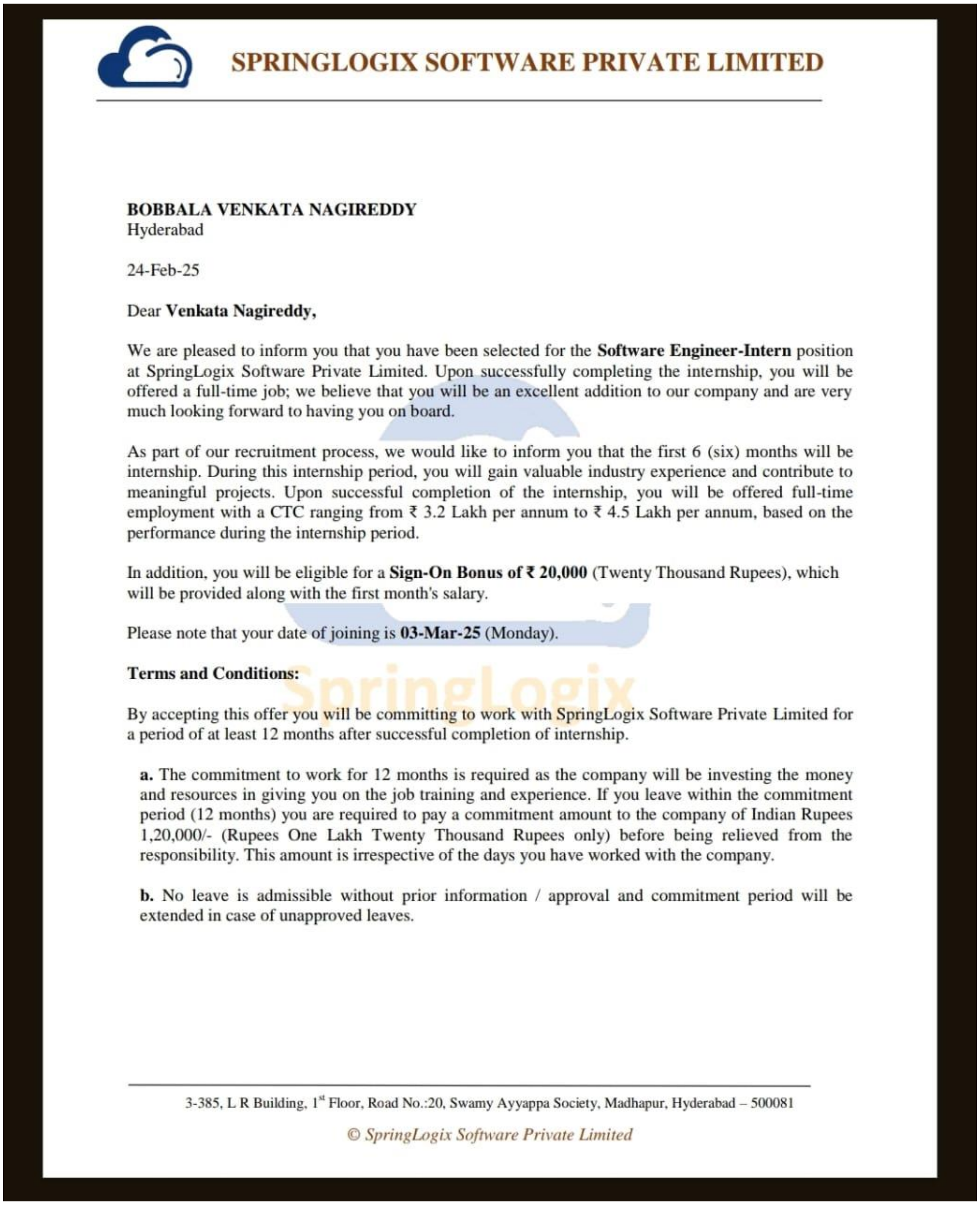
We are confident that this experience will be invaluable to your professional development and look forward to your successful participation in the Propel5000 program.

Warm regards,

[illegible]

Dr. N Poojitha Teja
Cofounder & CEO
FresherBot

Voilacode Technologies Private Limited
CIN : U62013TS2023PTC176712
info@fresherbot.com
201,Oyester Uptown, Beside Durgam Cheruvu Metro Station
Madhapur,Hyderabad - 500081
+91 89775 35750 | www.fresherbot.com



SPRINGLOGIX SOFTWARE PRIVATE LIMITED

BOBBALA VENKATA NAGIREDDY
Hyderabad

24-Feb-25

Dear **Venkata Nagireddy,**

We are pleased to inform you that you have been selected for the **Software Engineer-Intern** position at SpringLogix Software Private Limited. Upon successfully completing the internship, you will be offered a full-time job; we believe that you will be an excellent addition to our company and are very much looking forward to having you on board.

As part of our recruitment process, we would like to inform you that the first 6 (six) months will be internship. During this internship period, you will gain valuable industry experience and contribute to meaningful projects. Upon successful completion of the internship, you will be offered full-time employment with a CTC ranging from ₹ 3.2 Lakh per annum to ₹ 4.5 Lakh per annum, based on the performance during the internship period.

In addition, you will be eligible for a **Sign-On Bonus of ₹ 20,000** (Twenty Thousand Rupees), which will be provided along with the first month's salary.

Please note that your date of joining is **03-Mar-25** (Monday).

Terms and Conditions:

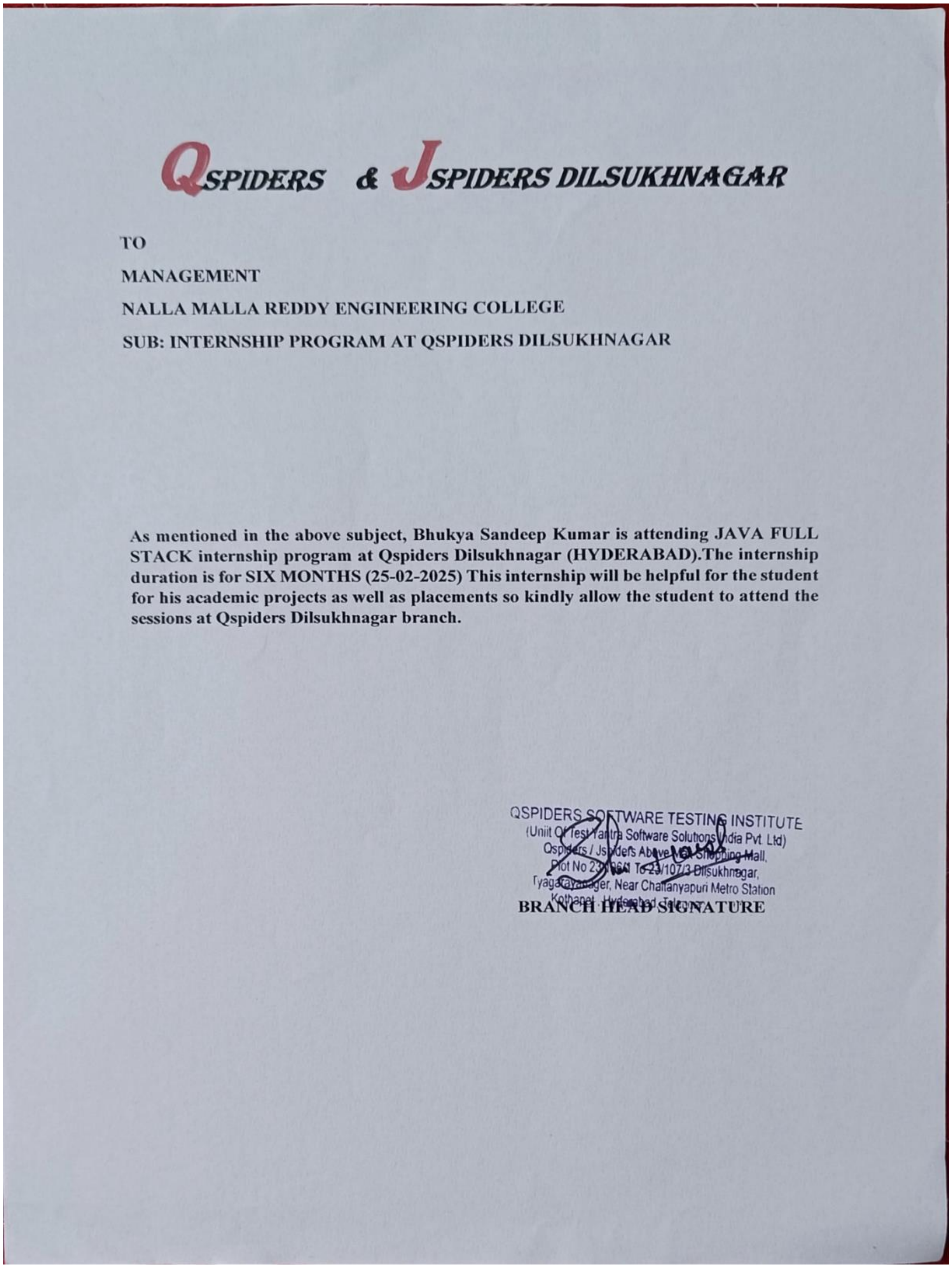
By accepting this offer you will be committing to work with SpringLogix Software Private Limited for a period of at least 12 months after successful completion of internship.

a. The commitment to work for 12 months is required as the company will be investing the money and resources in giving you on the job training and experience. If you leave within the commitment period (12 months) you are required to pay a commitment amount to the company of Indian Rupees 1,20,000/- (Rupees One Lakh Twenty Thousand Rupees only) before being relieved from the responsibility. This amount is irrespective of the days you have worked with the company.

b. No leave is admissible without prior information / approval and commitment period will be extended in case of unapproved leaves.

3-385, L R Building, 1st Floor, Road No.:20, Swamy Ayyappa Society, Madhapur, Hyderabad – 500081

© SpringLogix Software Private Limited



QSPIDERS & JSPIDERS DILSUKHNAGAR

TO

MANAGEMENT

NALLA MALLA REDDY ENGINEERING COLLEGE

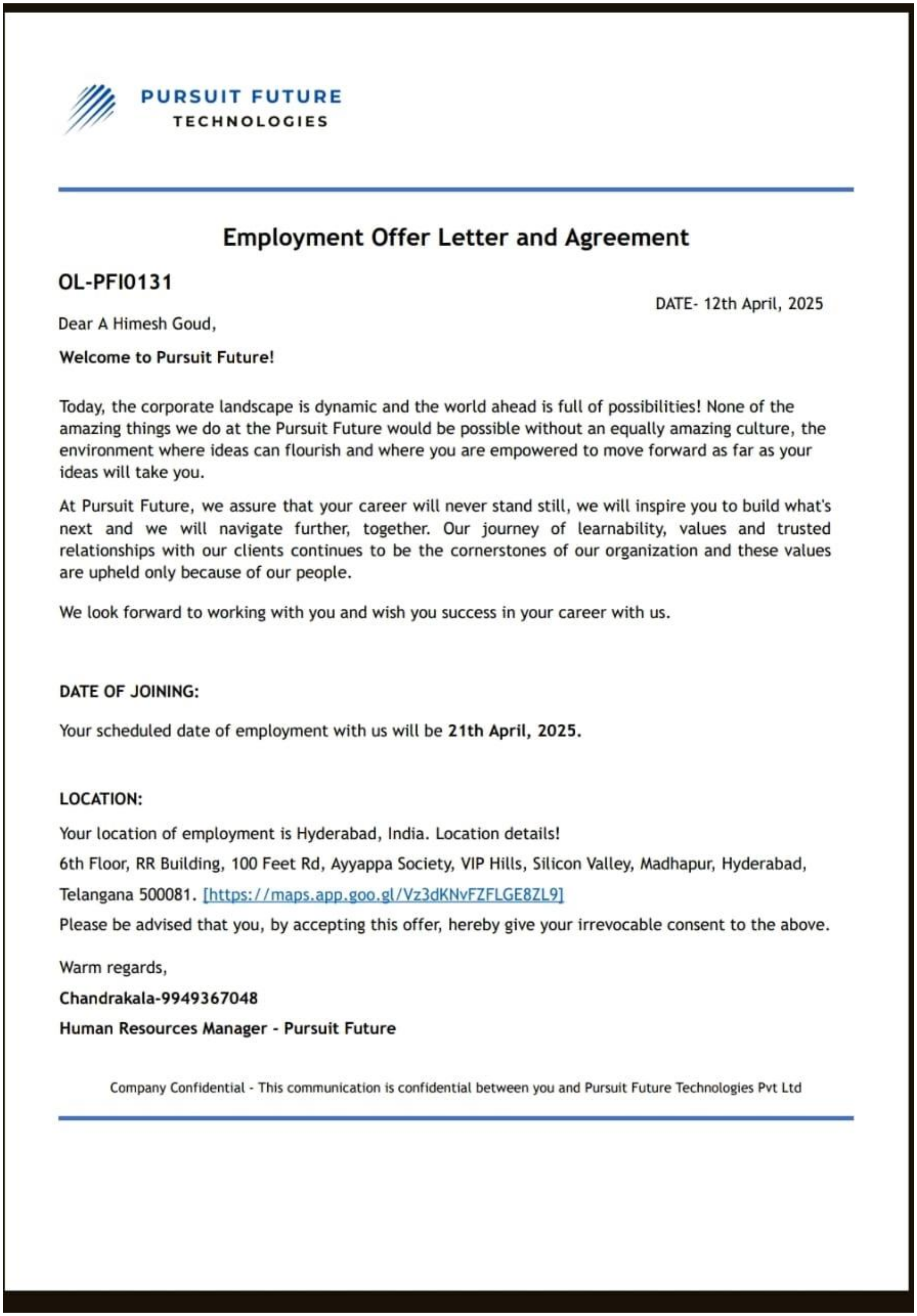
SUB: INTERNSHIP PROGRAM AT QSPIDERS DILSUKHNAGAR

As mentioned in the above subject, Bhukya Sandeep Kumar is attending JAVA FULL STACK internship program at Qspiders Dilsukhnagar (HYDERABAD).The internship duration is for SIX MONTHS (25-02-2025) This internship will be helpful for the student for his academic projects as well as placements so kindly allow the student to attend the sessions at Qspiders Dilsukhnagar branch.

QSPIDERS SOFTWARE TESTING INSTITUTE
(Uniit Of Test Yantra Software Solutions India Pvt. Ltd)
Qspiders / Jspiders Above [illegible] Shopping Mall,
Plot No 23/106/1 To 23/107/3 Dilsukhnagar,
Tyagarayanagar, Near Chaitanyapuri Metro Station
Kothapet, Hyderabad, Telangana

[illegible]

BRANCH HEAD SIGNATURE



PURSUIT FUTURE
TECHNOLOGIES

Employment Offer Letter and Agreement

OL-PFI0131

DATE- 12th April, 2025

Dear A Himesh Goud,

Welcome to Pursuit Future!

Today, the corporate landscape is dynamic and the world ahead is full of possibilities! None of the amazing things we do at the Pursuit Future would be possible without an equally amazing culture, the environment where ideas can flourish and where you are empowered to move forward as far as your ideas will take you.

At Pursuit Future, we assure that your career will never stand still, we will inspire you to build what's next and we will navigate further, together. Our journey of learnability, values and trusted relationships with our clients continues to be the cornerstones of our organization and these values are upheld only because of our people.

We look forward to working with you and wish you success in your career with us.

DATE OF JOINING:

Your scheduled date of employment with us will be **21th April, 2025.**

LOCATION:

Your location of employment is Hyderabad, India. Location details!

6th Floor, RR Building, 100 Feet Rd, Ayyappa Society, VIP Hills, Silicon Valley, Madhapur, Hyderabad, Telangana 500081. [https://maps.app.goo.gl/Vz3dKNvFZFLGE8ZL9]

Please be advised that you, by accepting this offer, hereby give your irrevocable consent to the above.

Warm regards,

Chandrakala-9949367048

Human Resources Manager - Pursuit Future

Company Confidential - This communication is confidential between you and Pursuit Future Technologies Pvt Ltd