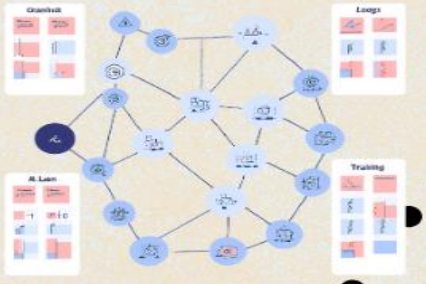
in collaboration with

Softwarica College of IT & E-commerce | Coventry University

Real-Time detection of insider threats using anomaly-based machine learning models in privileged access systems for healthcare databases in Kathmandu Metropolitan



INSIGHTMED

ON HEALTHCARE ANAMALY DETECTION

SUBMITTED BY: AVISHEK DHAKAL

CU ID:12981148

STUDENT ID: 220064

SUBMITTED TO: MANOJ SHRESTHA

Risk Research Ethics Approval

Project Title

Real-Time Detection of Insider Threats Using Anomaly-Based Machine Learning Models in Privileged Access Systems for Healthcare Databases in Kathmandu Metropolitan

Record of Approval

Principal Investigator

I request an ethics peer review and confirm that I have answered all relevant questions in this checklist honestly.	×
I confirm that I will carry out the project in the ways described in this checklist. I will immediately suspend research and request new ethical approval if the project subsequently changes the information I have given in this checklist.	×
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the Code of Research Ethics issued by the relevant national learned society.	×
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the University's Research Ethics, Governance and Integrity Framework.	×
I understand that I cannot begin my research until this ethics application has been approved.	×

Name: Avishek Dhakal

Date:

Student's Supervisor (if applicable)

I have read this checklist and confirm that it covers all the ethical issues raised by this project fully and frankly.I also confirm that these issues have been discussed with the student and will continue to be reviewed in the course of supervision.

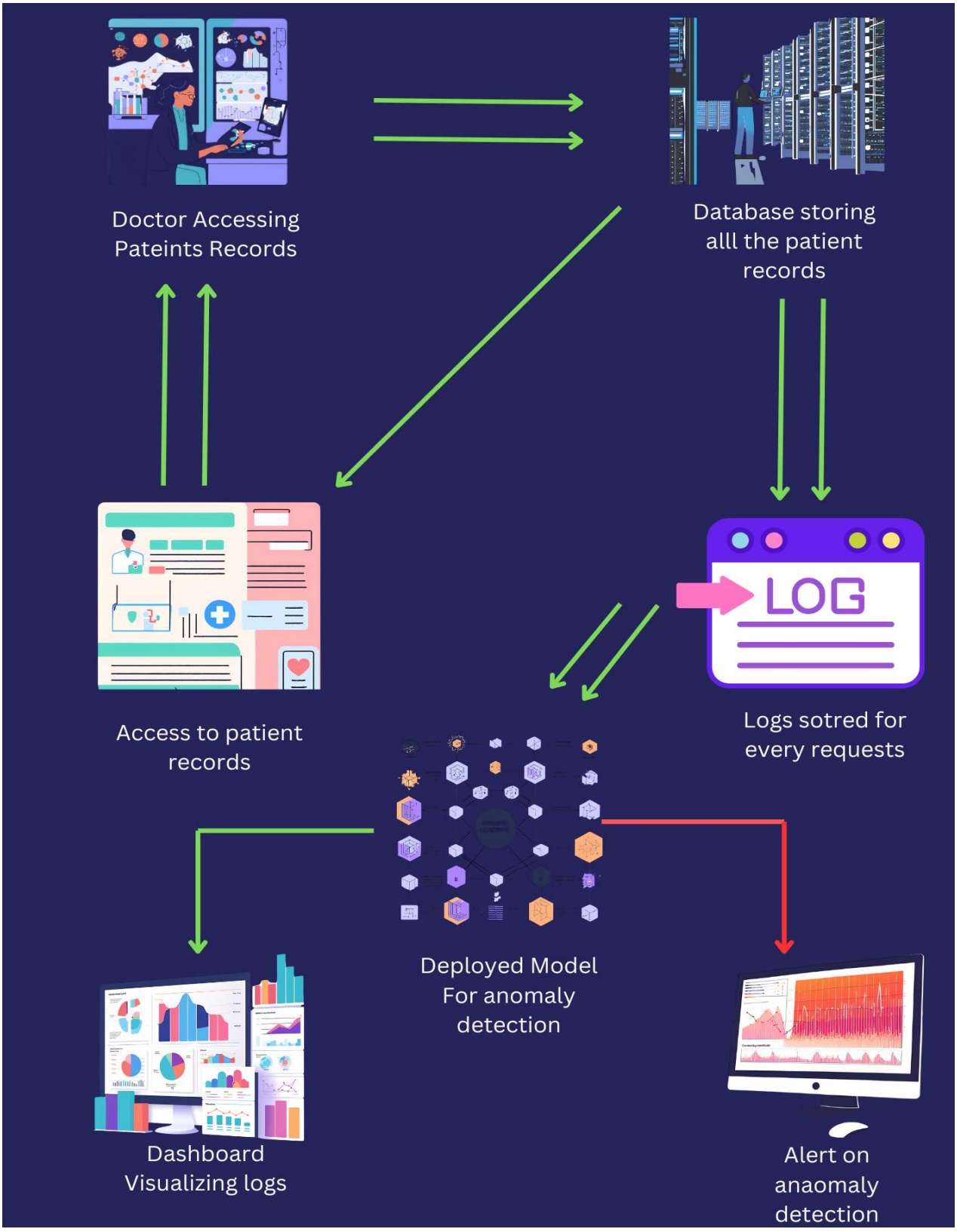
Name: Manoj Shrestha

Date:

Reviewer (if applicable)

Date of approval by anonymous reviewer: -

CONCEPTUAL DIAGRAM



KEYWORDS


Streamlit Dashboard
Synthetic Data Generation
Insider Risk Management
Data Privacy Compliance
Supervised Learning
Behavioral Analytics
Healthcare Cybersecurity
Machine Learning Security
Insider Threat Detection
Anomaly-Based Detection
Role-Based Access Control
Intrusion Detection Systems
Real-Time Monitoring
Feature Engineering
Security Log Analysis

ACKNOWLEDGEMENT

I sincerely want to express my gratitude to my **supervisor, Mr. Manoj Shrestha**, for his invaluable guidance, expertise, and continuous support throughout the development and documentation of this thesis. I also extend my heartfelt thanks to our **Head of Department, Mr. Ganesh Bhusal**, whose unwavering mentorship has been instrumental throughout my academic journey. Finally, I deeply appreciate the **unconditional support and encouragement of my family**, who have always stood by me in all my endeavors. Their belief in me has been my greatest motivation.

EXECUTIVE SUMMARY

The digitization of healthcare records and hospital management systems has made insider threats a critical cybersecurity concern. Unlike external attacks, insider threats exploit legitimate access privileges, making them difficult to detect using traditional security mechanisms such as firewalls, intrusion detection systems (IDS), and access control lists (ACLs). This study explores the limitations of conventional security models and proposes an AI-driven anomaly detection system to identify insider threats in Kathmandu's healthcare institutions.

By leveraging machine learning-based behavioral analytics, the system detects deviations from normal user activity, significantly improving real-time threat detection. The research findings highlight the effectiveness of AI-driven security monitoring, while also addressing challenges such as false positives, computational efficiency, and ethical concerns. The proposed InsightMed system contributes to strengthening cybersecurity frameworks in hospitals, ensuring the protection of sensitive patient data and compliance with regulatory standards.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

The healthcare sector in Kathmandu Metropolitan is undergoing rapid digital transformation, with hospitals increasingly relying on electronic health records, cloud-based medical systems, and automated patient management tools. These advancements have significantly improved efficiency and accessibility but have also introduced critical cybersecurity risks. As healthcare institutions handle highly sensitive patient data, they have become prime targets for cyberattacks. While external cyber threats such as ransomware and phishing attacks have received significant attention, insider threats remain one of the most overlooked yet dangerous risks. Unlike external attackers, insider threats originate from individuals who already have legitimate access to hospital systems. These threats often involve employees misusing their access privileges to steal, alter, or misuse confidential medical records. Detecting such malicious activities is particularly challenging since traditional security measures focus on external threats and struggle to differentiate between authorized and unauthorized internal behavior. Reports indicate that insider threats account for a significant portion of security incidents in healthcare. According to the **Ponemon Institute (2021)**, **more than 25% of healthcare security breaches are insider-driven**, underscoring the urgent need for advanced detection mechanisms (**Ponemon Institute, 2021**).

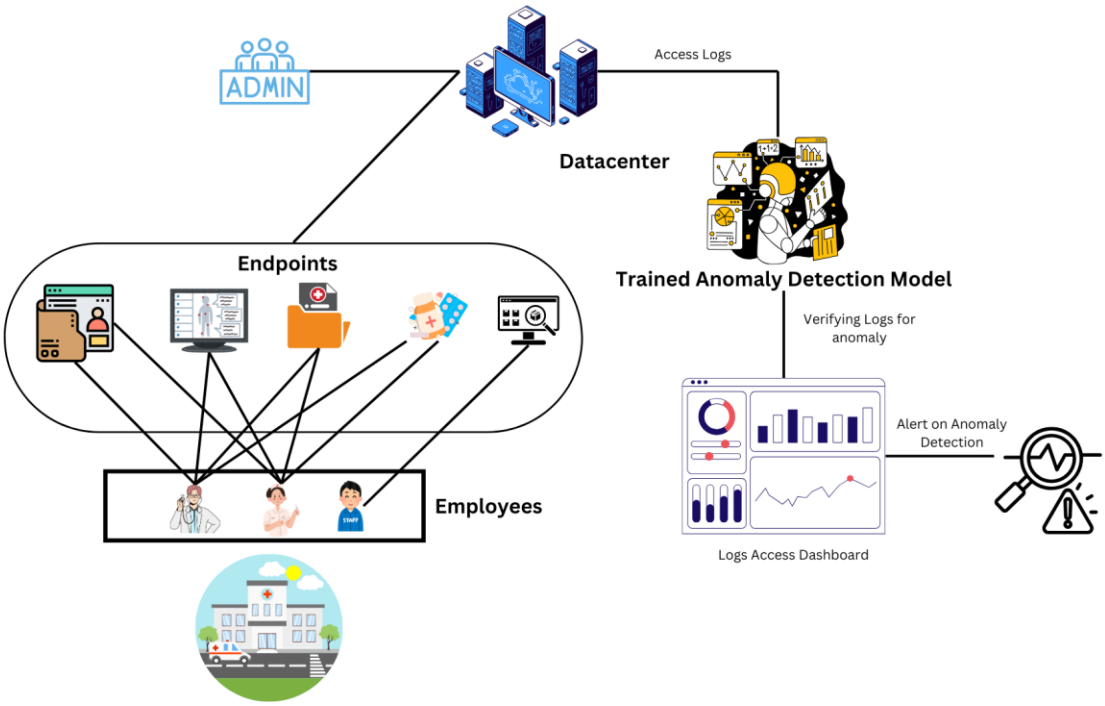


Figure1: High overview of project

Existing security frameworks in healthcare institutions primarily rely on rule-based access control mechanisms, which often fail to detect suspicious activities carried out by insiders. These systems operate on predefined security rules, flagging only known attack patterns, making them ineffective against insider threats that do not match conventional hacking attempts. Insider attacks are not always intentional; they may also result from negligence or accidental data exposure. The

increasing reliance on digital health infrastructures necessitates an advanced approach to insider threat detection, particularly in Kathmandu, where hospitals are gradually adopting AI-based cybersecurity measures but still lack structured monitoring frameworks. Addressing these challenges requires machine learning-based anomaly detection systems capable of identifying unusual access behaviors without relying on predefined rules. Such an approach allows the system to detect subtle deviations from normal user behavior, flagging potential threats in real time.

To address this growing concern, **InsightMed**, a machine learning-based anomaly detection system, was developed as part of this research to improve **insider threat detection in Kathmandu's hospitals**. The system processes hospital access logs and applies **anomaly detection algorithms to identify suspicious behaviors**, such as unauthorized login attempts, access from unusual locations, and privilege misuse. InsightMed leverages **Random Forest** to classify anomalous behaviors while ensuring that legitimate hospital operations are not unnecessarily interrupted. By integrating **behavior-based monitoring**, the system enhances security frameworks that traditionally fail to detect **subtle insider threats**. This research not only provides a technical solution but also considers **real-world hospital security challenges**, ensuring that the model is practical for **implementation in resource-constrained environments like Kathmandu**.

Machine learning-based anomaly detection provides a promising alternative to traditional rule-based security mechanisms. By analyzing access patterns, login times, network activity, and privilege levels, an anomaly detection system can effectively differentiate between normal and suspicious behavior. Hospitals generate vast amounts of security logs daily, and manually analyzing this data is impractical. Machine learning automates this process by identifying hidden patterns and classifying potential threats based on behavioral deviations. The advantage of this approach lies in its ability to adapt over time, learning from new data and continuously improving its threat detection capabilities. However, despite its potential, integrating such a system into Kathmandu's healthcare infrastructure presents several challenges, including limited structured log data, computational constraints, and the high rate of false positives.

The increasing frequency of insider-driven healthcare data breaches highlights the urgent need for AI-based security solutions. Healthcare organizations often lack **real-time monitoring capabilities**, making it difficult to detect unauthorized activities before they escalate. This issue is particularly concerning in Kathmandu, where hospitals are still in the early stages of implementing AI-based security frameworks. The absence of robust security policies and technical expertise further complicates the deployment of advanced threat detection systems. Without real-time monitoring and anomaly detection capabilities, insider attacks can go undetected for extended periods, causing financial, operational, and reputational damage to healthcare institutions. Ensuring that hospitals have access to reliable cybersecurity infrastructure is essential for preventing unauthorized access and mitigating potential security risks.

Ethical considerations must also be addressed when implementing AI-driven security solutions. Insider threat detection requires constant monitoring of user behavior, which raises concerns regarding employee privacy and workplace ethics. Hospital staff may feel uncomfortable knowing that their activities are being tracked by an AI system, leading to concerns about workplace surveillance and potential misuse of monitoring data. Transparency in security policies is crucial to ensuring that employees understand how and why monitoring is conducted. Privacy laws and regulations must also be considered when implementing AI-driven security solutions in healthcare environments. Although Nepal does not yet have comprehensive AI governance policies, global data protection regulations such as **GDPR (General Data Protection Regulation) and HIPAA (Health Insurance Portability and Accountability Act)** provide useful guidelines for ensuring compliance (**European Parliament, 2016**). AI models must also be designed to minimize bias in decision-making, as poorly trained models may disproportionately flag certain roles or departments as high-risk.

The increasing reliance on digital security solutions in Kathmandu's healthcare sector presents both opportunities and challenges. AI-based anomaly detection has the potential to revolutionize insider threat detection by providing real-time security monitoring while reducing reliance on manual threat analysis. However, practical implementation requires overcoming significant challenges, including data quality issues, real-time processing inefficiencies, and regulatory gaps. Ethical considerations regarding employee privacy and algorithmic bias must also be addressed to ensure that AI security frameworks are deployed responsibly. As hospitals continue to modernize their security infrastructure, integrating machine learning into cybersecurity frameworks will be essential for protecting patient data and ensuring healthcare institutions remain resilient against emerging insider threats.

AIM

Develop and rigorously evaluate a real-time, anomaly-based machine learning system for detecting insider threats within privileged access frameworks of healthcare databases in Kathmandu Metropolitan, focusing on efficacy, scalability, and adaptability to the region's unique challenges.

Figure2: Aim

OBJECTIVES

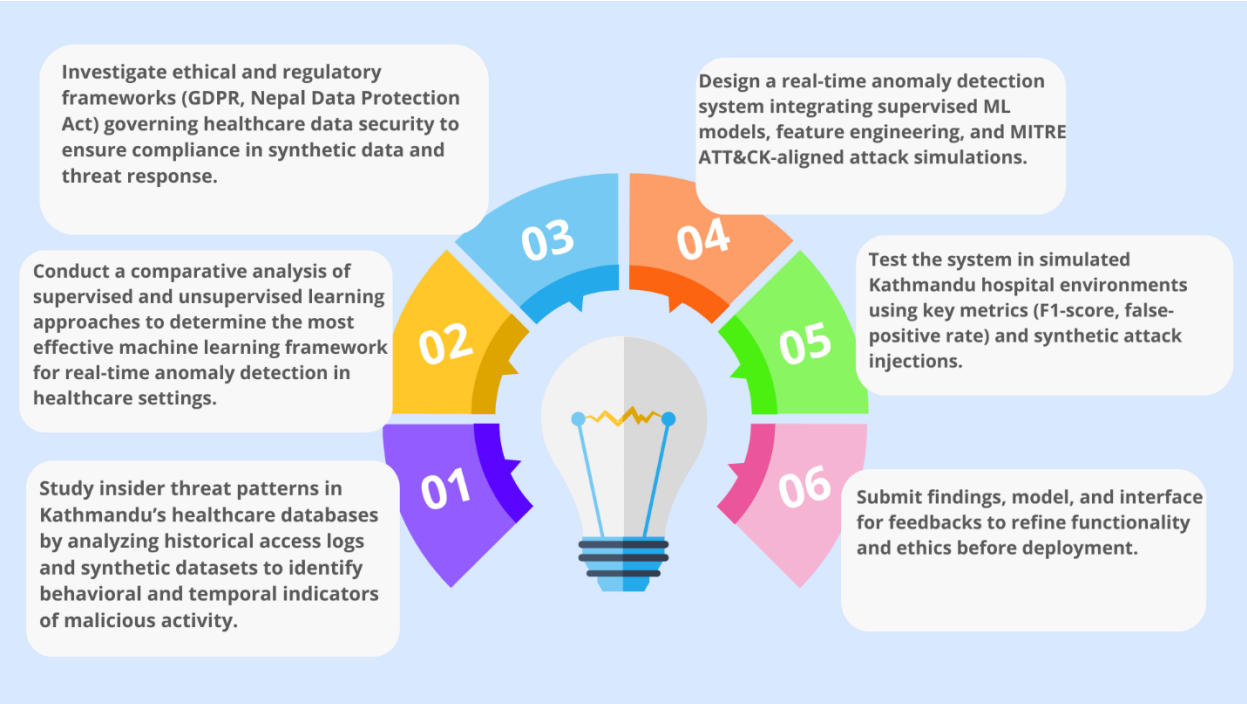


Figure3: Objectives

JUSTIFICATION

Healthcare cybersecurity has become an increasingly pressing issue, particularly in Kathmandu's hospitals, where the transition to digital record-keeping and automated systems has outpaced the

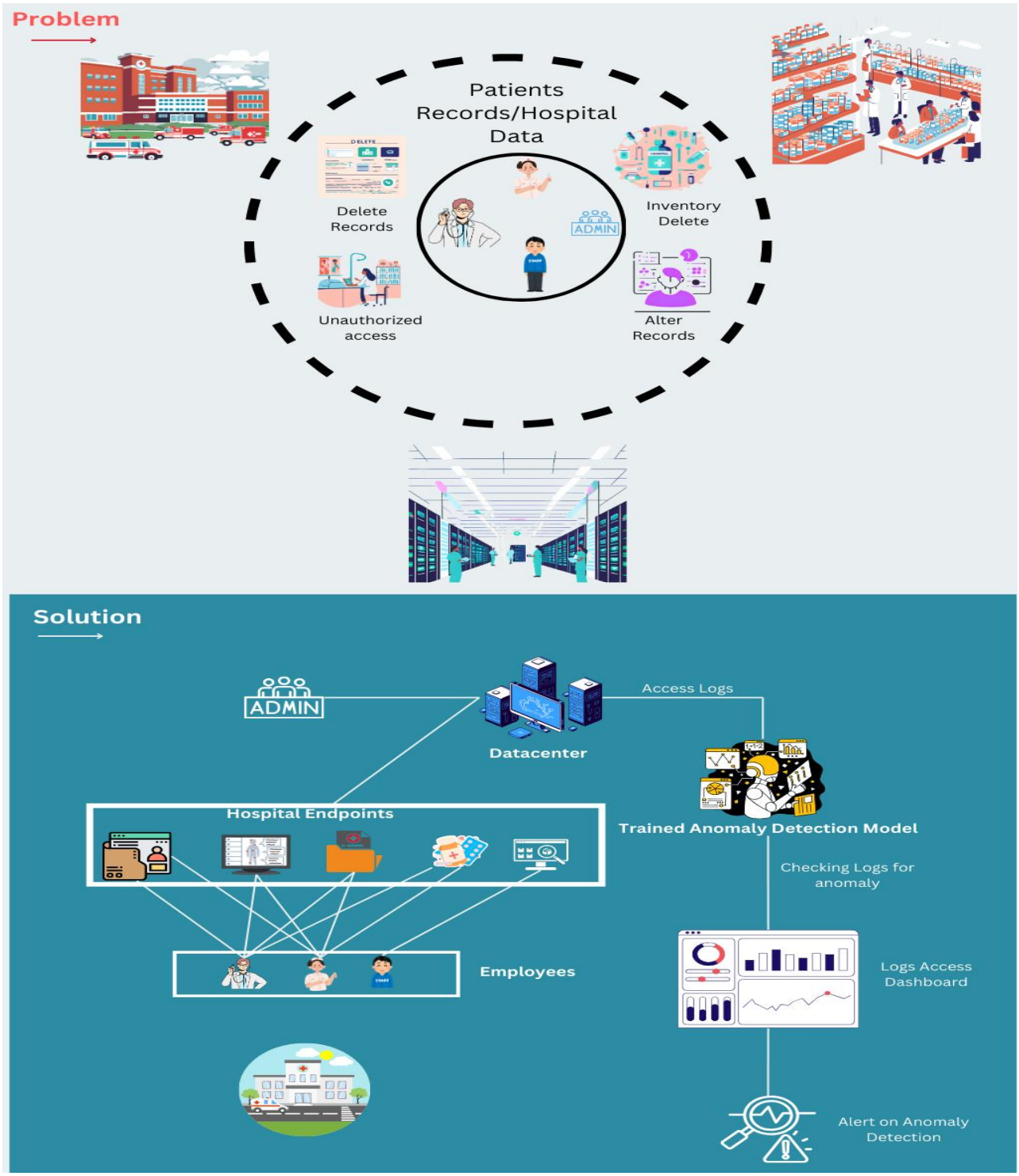


Figure4: Visualizing problem and solution

development of robust security frameworks. With the adoption of electronic health records, hospital management systems, and cloud-based medical databases, hospitals now handle large volumes of sensitive patient data that require strict security measures. However, insider threats, where employees misuse their legitimate access to hospital systems, remain a significant challenge. Unlike external cyberattacks, insider threats exploit authorized credentials, making them difficult to detect through conventional security mechanisms. Kathmandu's healthcare institutions, many of which lack dedicated cybersecurity teams and advanced monitoring tools, are particularly vulnerable to such threats.

Existing security models in hospitals primarily rely on role-based access control (RBAC), firewalls, and traditional logging mechanisms. These systems can prevent unauthorized external access, but they struggle to detect anomalies in internal activities. A hospital administrator, for instance, may have unrestricted access to patient records but could misuse this access to modify or leak confidential data without triggering security alarms. Similarly, nurses or IT staff may inadvertently or intentionally retrieve information beyond their clearance level yet remain undetected due to the rigid nature of role-based security. The reliance on manual audits for reviewing access logs further exacerbates the problem, as these methods are time-consuming, error-prone, and incapable of identifying real-time security breaches.

To address these challenges, InsightMed, a machine learning-driven anomaly detection system, was developed as part of this research to improve insider threat detection in Kathmandu's hospitals. Unlike rule-based security models that flag only predefined security violations, InsightMed **identifies subtle deviations in user behavior** that may indicate unauthorized or suspicious activity. The system continuously analyzes hospital access logs, detecting anomalies based on login behavior, time of access, role-based access risk, and network activity. By integrating machine learning with behavioral analysis, InsightMed significantly improves threat detection accuracy while reducing false positives, which are common in traditional security systems.

The need for such a system is further justified by the **inconsistencies in data logging and security monitoring** across Kathmandu's healthcare institutions. Many hospitals operate without a centralized cybersecurity infrastructure, making it difficult to establish standardized security policies. Additionally, the absence of automated real-time security monitoring means that security incidents are often detected long after the breach has occurred, leading to delays in response and remediation efforts. The manual nature of hospital security audits also leaves room for human error, where insider threats may be overlooked due to limited personnel dedicated to cybersecurity enforcement.

The figure in this section visually illustrates the central problem and the proposed solution of this research. It depicts what hospitals currently face as insider threats, highlighting the lack of automated anomaly detection and reliance on reactive security measures. The problem section of the figure showcases unauthorized access, altering of medical records , and deletion of medical

equipment or data. The solution section highlights the proposed AI-driven anomaly detection system, demonstrating how real-time monitoring, machine learning-based analysis, and security alerts can prevent insider threats before they escalate.

Kathmandu's healthcare system also faces technical and infrastructural challenges that necessitate a more adaptive, lightweight, and scalable security model. Many hospitals in the region lack the computing resources to deploy large-scale machine learning models, making it essential for the anomaly detection system to be optimized for efficiency. InsightMed is designed to operate within the constraints of hospital IT infrastructure, ensuring low latency, minimal computational overhead, and high detection accuracy. The system's ability to function in resource-limited environments makes it a practical security solution for hospitals in Nepal, where cybersecurity funding and expertise are still developing. Furthermore, the increasing use of remote access for hospital IT systems due to telemedicine and electronic data sharing has exacerbated insider threat risks. Without a strong anomaly detection system, hospital employees can exploit remote access privileges to bypass security measures. The InsightMed system integrates network-based anomaly detection, which can flag unauthorized access attempts from unknown locations or external IP addresses, further strengthening hospital security against insider-driven cyberattacks.

The justification for this research lies not only in enhancing security monitoring but also in **reducing the burden on hospital security teams**. Many hospitals lack dedicated cybersecurity personnel to manually track security events. By automating anomaly detection, InsightMed reduces the need for manual security log reviews, ensuring that hospital security teams can focus on investigating only high-risk incidents rather than being overwhelmed with false alarms and excessive security logs.

As Kathmandu's healthcare institutions continue to expand their digital infrastructure, it is essential that security frameworks evolve to meet the growing cybersecurity risks. The development of InsightMed as an AI-driven security model ensures that hospitals can transition from reactive to proactive security monitoring, minimizing risks while maintaining the confidentiality and integrity of patient data. This research is a significant step towards **modernizing cybersecurity strategies in Nepal's healthcare industry**, ensuring that hospitals remain protected from internal and external security threats.

RESEARCH QUESTIONS

Research Question 1		How effective are anomaly-based machine learning models in detecting insider threats compared to traditional detection methods within healthcare databases in Kathmandu Metropolitan, and what modifications can optimize their performance for local systems?
Research Question 2		What are the most significant behavioral and access-pattern indicators of insider threats in Kathmandu's healthcare databases, and how can these be dynamically integrated into an anomaly-based detection system for real-time use?
Research Question 3		What are the technical, ethical, and socio-economic challenges in deploying a real-time anomaly-based insider threat detection system in Kathmandu's healthcare sector, and how can these challenges be mitigated through design and policy recommendations?

Figure5: Research Questions

SCOPE

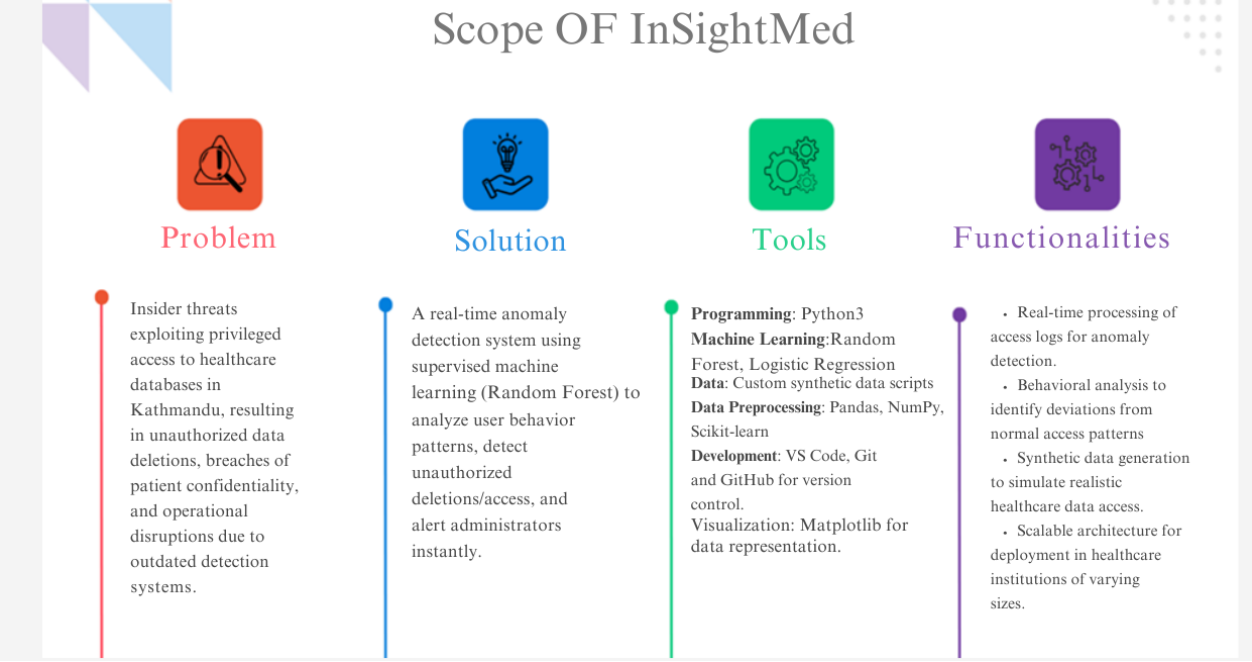


Figure6: Scope of InSightMed

ETHICAL CONSIDERATIONS

The deployment of an anomaly-based insider threat detection system in healthcare institutions raises significant ethical concerns related to privacy, data protection, and regulatory compliance. Since healthcare databases contain sensitive patient information, any security system that monitors user access and behavioral patterns must comply with privacy laws and ethical best practices to avoid unintended violations of confidentiality.

One of the primary ethical concerns is **employee privacy**. The system continuously tracks user activity, such as login attempts, access times, and endpoint usage, to detect anomalies. While this is necessary for security, excessive surveillance may create a hostile work environment, where employees feel unfairly scrutinized. To mitigate this, hospitals must ensure **transparency in AI security policies**, informing employees about what data is monitored and why.

Compliance with data protection laws is another crucial aspect. While **Nepal does not yet have comprehensive AI governance policies** in healthcare, global standards such as GDPR (General Data Protection Regulation) and HIPAA (Health Insurance Portability and Accountability Act) provide guidance on data access controls, anonymization techniques, and user consent mechanisms. Implementing role-based access control (RBAC) and data minimization strategies can ensure the model only collects necessary security data without excessive personal tracking.

Furthermore, AI-based decision-making must be explainable and free from bias. Models that unfairly flag certain roles or departments as high-risk could lead to workplace discrimination. Conducting bias audits, ensuring human oversight in anomaly validation, and incorporating explainable AI techniques (such as SHAP values) are essential for ensuring fair and ethical use of AI in security monitoring.

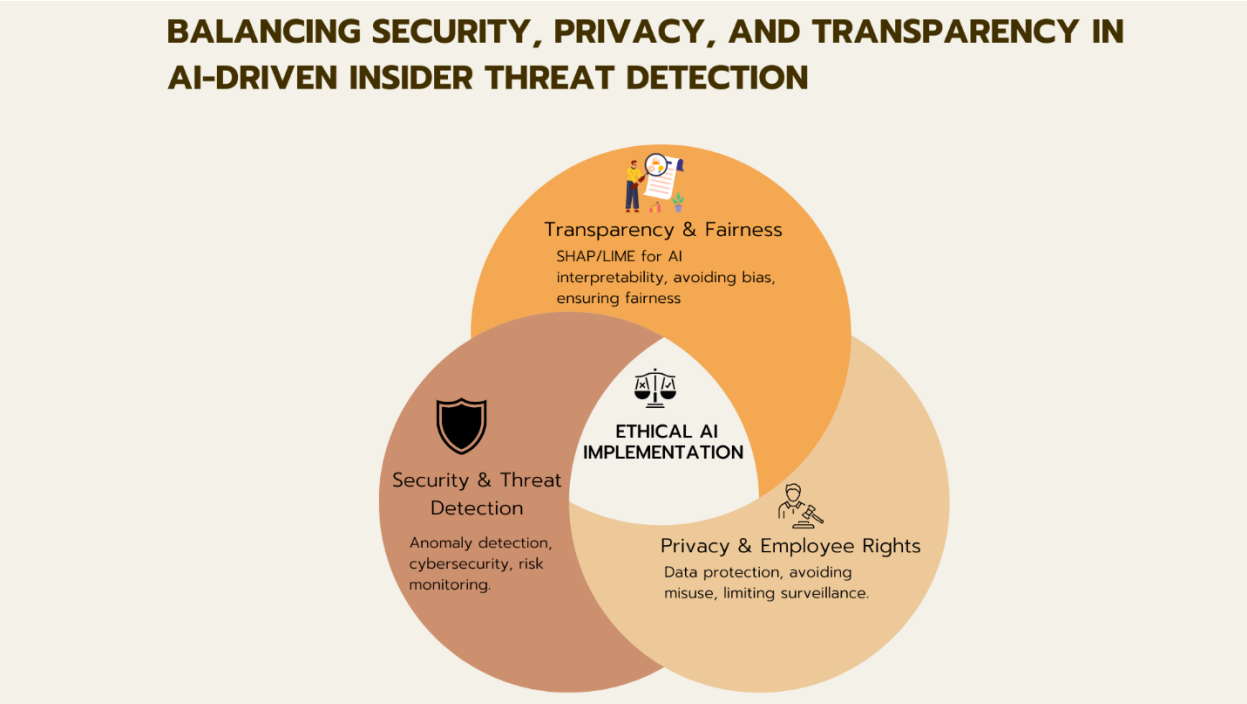


Figure7: Balance between security privacy and transparency

LITERATURE REVIEW

RESEARCH METHODOLOGY

This study employs a **desk-based research methodology**, analyzing secondary sources such as **peer-reviewed journals, cybersecurity frameworks, government reports, and case studies** to assess the effectiveness of **machine learning (ML) for insider threat detection in healthcare systems**. The decision to use a **desk-based approach** is primarily influenced by the **sensitivity and confidentiality** surrounding real-world cybersecurity incidents. Direct access to insider threat data is highly restricted due to privacy regulations like **Nepal's Data Protection Act (2022)** and **HIPAA (Health Insurance Portability and Accountability Act) in the U.S.** (Smith & Koenig, 2021). As a result, this research synthesizes empirical studies, industry best practices, and theoretical models to build a comprehensive understanding of insider threat detection in healthcare.

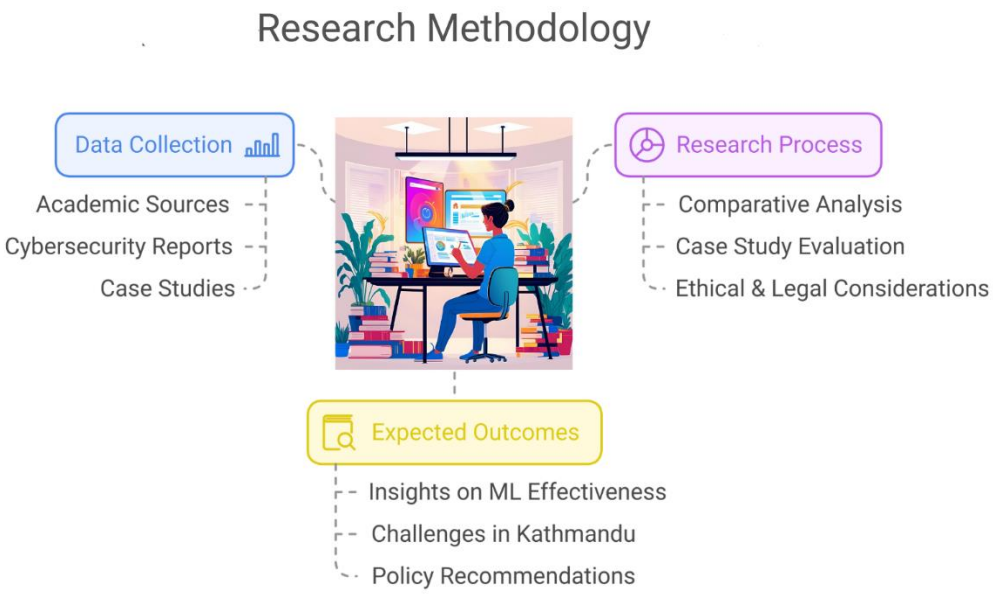


Figure8: Desk Research Methodology

A **systematic literature review** process was followed to ensure **credibility and relevance** in the selection of sources. The review prioritizes **publications from 2019 to 2024**, focusing on peer-reviewed articles from IEEE, ACM, and Springer, as well as official cybersecurity frameworks, including:

MITRE ATT&CK Insider Threat Model – A global cybersecurity framework that classifies insider threat tactics (MITRE, 2023).

CERT Insider Threat Framework – A structured model for identifying behavioral indicators of insider threats (Cappelli et al., 2020).

Nepal's Cybersecurity Policies – Legislative frameworks governing cybersecurity in healthcare (Nepal Government, 2022).

The research methodology follows the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines, ensuring that only high-quality, peer-reviewed sources are considered. This approach allows for a comparative analysis of different ML-based anomaly detection methods, assessing their applicability in real-world hospital environments. Furthermore, this study evaluates case studies from leading healthcare institutions, examining major data breaches linked to insider threats. The findings from these cases serve as practical evidence supporting the necessity of ML-driven insider threat detection. Additionally, synthetic log datasets were analyzed to assess the role of behavioral analytics in detecting healthcare anomalies (Ferreira et al., 2020).

By integrating cybersecurity best practices, machine learning advancements, and real-world case studies, this literature review establishes a comprehensive framework for understanding insider threats in Kathmandu's healthcare sector.

INSIDER THREATS IN HEALTHCARE

The healthcare sector is one of the most targeted industries for cybersecurity threats, with electronic health records (EHRs), medical billing systems, and diagnostic imaging data making hospitals prime targets for data breaches (Bhuyan et al., 2020). While external cyberattacks such as ransomware and phishing are well-documented, insider threats remain an equally severe yet often overlooked risk (Smith & Koenig, 2021). Insider threats in healthcare refer to malicious or negligent activities carried out by individuals with authorized access to hospital networks and patient databases (Cappelli et al., 2020).

Insider-driven breaches are particularly dangerous because traditional security measures—such as firewalls and access control lists—often **fail to detect unauthorized but seemingly legitimate actions** (Maglaras et al., 2019). For example, an **authorized doctor or nurse** may **access confidential patient records** outside their assigned cases, violating data privacy regulations (HIPAA, GDPR). Additionally, privileged hospital administrators can exfiltrate medical data for financial gain without triggering conventional security alerts.

CLASSIFICATIONS OF INSIDER THREATS IN HEALTHCARE

Insider threats in healthcare fall into three main categories, each posing distinct cybersecurity risks (Cappelli et al., 2020). **Malicious insiders** are employees who deliberately exploit their access to steal, alter, or sell sensitive patient data, often motivated by financial gain, espionage, or personal revenge. For instance, a staff member might sell patient records to fraudulent insurance companies. **Negligent insiders**, on the other hand, unintentionally expose sensitive data due to carelessness, insufficient cybersecurity training, or misconfigured access settings. A common example is a nurse mistakenly sending patient reports to an unauthorized recipient due to poor email practices. Lastly, **compromised insiders** involve employees whose credentials are stolen and misused by cybercriminals through phishing, social engineering, or credential stuffing. This type of threat is particularly challenging to detect, as attackers appear to be legitimate users within the hospital's system, potentially modifying electronic health records (EHR) without raising immediate suspicion. Understanding these classifications is crucial for designing effective cybersecurity measures in healthcare environments.

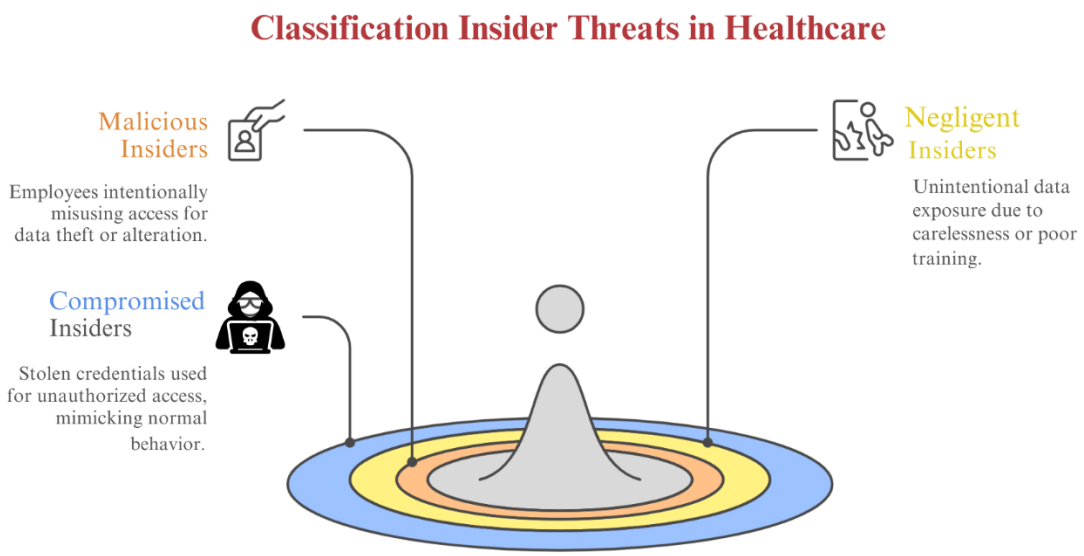


Figure9: Classification of insider threats

EXISTING APPROACHES TO INSIDER THREAT DETECTION

Insider threats pose a unique challenge to cybersecurity, particularly in industries handling sensitive data, such as healthcare, finance, and government sectors. Traditional security measures, such as role-based access control **(RBAC), firewall protections, and signature-based intrusion detection systems (IDS)**, have long been used to regulate access and monitor network activity. However, these approaches primarily focus on **external threats**, leaving significant gaps in identifying malicious activities carried out by **authorized users** (Chandola, Banerjee, & Kumar, 2009). RBAC, for example, assigns predefined access permissions to users based on their roles within an organization. While effective in restricting unauthorized entry, it lacks the ability to detect behavioral anomalies, such as an employee accessing unusually large volumes of patient records or logging in from unauthorized locations (Elmrabit, Abdallah, & Smadi, 2021).

Signature-based intrusion detection systems function by comparing user activities against known attack patterns. However, insider threats do not always follow predictable patterns, making them difficult to detect through predefined security rules. Studies have shown that manual log audits and **rule-based detection models often fail to capture insider-driven attacks** until after the breach has occurred (Bishop, 2018). This has led to an increasing reliance on behavioral analytics and anomaly-based threat detection, which offer a more adaptive and **real-time approach to security monitoring** (Gavai et al., 2015).To strengthen insider threat detection, organizations have adopted frameworks like MITRE ATT&CK and CERT's Insider Threat Mitigation Program. The MITRE ATT&CK framework provides a structured model for analyzing tactics, techniques, and

procedures (TTPs) used by insiders, categorizing threats based on their behavior rather than their predefined permissions (MITRE, 2022). Similarly, CERT's Insider Threat Mitigation Program focuses on behavioral indicators and risk assessments to prevent security breaches before they escalate (CERT, 2021). These frameworks offer valuable guidelines for insider threat detection, but they still rely heavily on manual analysis and predefined security rules, which are insufficient for real-time security monitoring (Shabtai & Elovici, 2020).

Given the limitations of RBAC, IDS, and manual security audits, organizations are increasingly turning to machine learning-based anomaly detection models. These models analyze user behavior over time, learning from patterns to detect deviations that may indicate malicious intent (LeCun, Bengio, & Hinton, 2015). Unlike traditional methods, **AI-driven anomaly detection does not rely on predefined security policies but instead adapts dynamically to emerging threats**. Studies indicate that machine learning models such as Random Forest, Support Vector Machines, and Autoencoders can significantly improve the detection of insider threats by reducing false positives and identifying complex behavioral anomalies (Chattopadhyay & Kim, 2020). As a result, AI-driven security solutions are becoming an essential component of modern cybersecurity strategies, particularly in healthcare institutions where insider attacks pose significant risks to patient data privacy and regulatory compliance.

REAL-WORLD INSIDER THREAT CASES IN HEALTHCARE

Insider threats have played a role in several high-profile healthcare data breaches, leading to compromised patient records, financial fraud, and reputational damage. Unlike external cyberattacks, insider breaches are particularly difficult to detect because they involve legitimate access privileges being misused (Smith & Koenig, 2021).

While healthcare institutions employ firewalls, intrusion detection systems (IDS), and access controls, these methods fail to detect subtle, unauthorized behaviors by insiders who already have approved access. As the following case studies illustrate, traditional rule-based security mechanisms were unable to stop these incidents, reinforcing the necessity for ML-driven behavioral analytics in healthcare cybersecurity.

CASE STUDY 1: ANTHEM DATA BREACH (2015) – EXPLOITING LEGITIMATE ACCESS

INCIDENT SUMMARY

The Anthem data breach remains one of the **largest healthcare-related cyber incidents** in history, compromising the **personally identifiable information (PII) of 78.8 million** individuals (Anthem Breach Report, 2016). Unlike conventional cyberattacks that involve brute-force intrusions, this breach was executed through a combination of phishing and insider credential exploitation. The

attacker gained unauthorized access through legitimate login credentials, making it indistinguishable from normal user activity for an extended period (Smith & Koenig, 2021).

Anthem, one of the largest health insurance providers in the U.S., stored vast amounts of sensitive patient data, including names, Social Security numbers, medical IDs, addresses, and employment details. The breach was first identified in early 2015, but subsequent investigations revealed that the intrusion began as early as 2014, indicating a long-term, undetected data exfiltration (U.S. Office of Personnel Management, 2016).

HOW THE BREACH OCCURRED

The Anthem data breach unfolded in a structured yet deceptive manner, leveraging phishing techniques to steal employee credentials. Attackers sent fraudulent emails that mimicked legitimate communication, leading at least one employee to unknowingly enter their login details on a fake portal. With these valid credentials, the attackers gained access to Anthem's systems without raising immediate security alarms. Over time, they escalated privileges, enabling access to patient databases where they systematically extracted records in bulk. Since the intrusion relied on legitimate credentials, the activity remained undetected for months. It was only in January 2015 that an administrator noticed unusual database queries, revealing the breach after millions of records had already been stolen. The delayed detection highlights the challenge of identifying sophisticated credential-based attacks and underscores the importance of proactive monitoring and employee cybersecurity awareness.

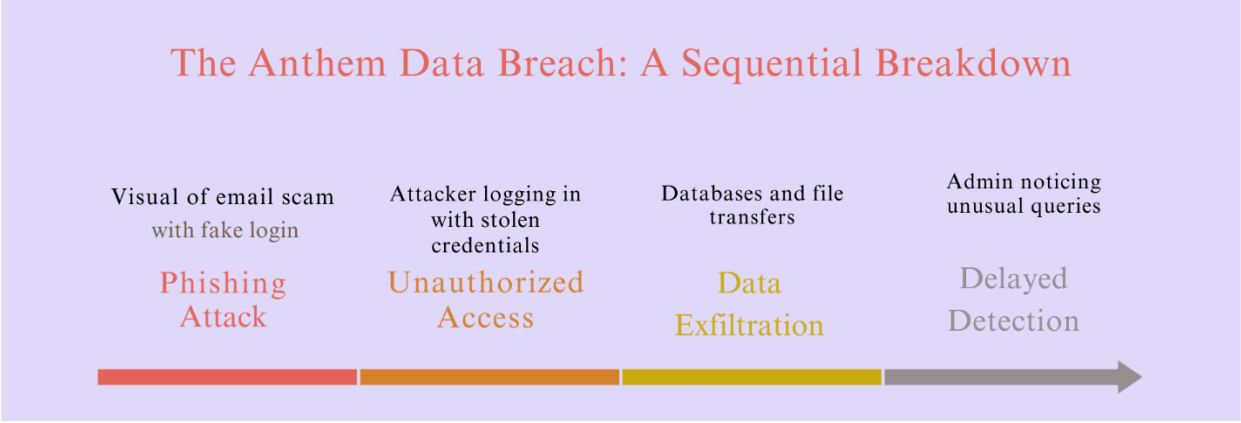


Figure10: Breakdown of anthem breach

WHY TRADITIONAL SECURITY FAILED

Traditional cybersecurity measures such as firewalls, access control lists (ACLs), and rule-based intrusion detection systems (IDS) proved ineffective in preventing the Anthem breach due to several shortcomings. One major flaw was the **lack of behavioral anomaly detection**—since the attackers used legitimate credentials, their activities were not flagged as suspicious. The system

failed to detect unusual patterns such as sudden spikes in data queries or access outside of regular working hours. Additionally, the reliance on **rule-based security models** delayed the response. Traditional security approaches depend on predefined attack patterns, such as failed login attempts or unauthorized IP addresses, but since the attackers mimicked normal employee behavior, they bypassed these security rules unnoticed. Furthermore, the absence of **continuous monitoring of high-privilege accounts** allowed attackers to extract vast amounts of patient data without raising alarms. Employees with extensive access to Anthem's electronic health records (EHR) were not actively monitored for abnormal activity, enabling the attackers to operate covertly for months before detection. These limitations highlight the vulnerabilities in static security frameworks that fail to detect sophisticated credential-based threats.

IMPACT & CONSEQUENCES

The Anthem data breach had far-reaching consequences, affecting millions of patients, regulatory bodies, and the company itself. **Financially,** the breach resulted in significant losses, with Anthem agreeing to a **$115 million settlement** to compensate affected individuals (Anthem Breach Settlement, 2017). Beyond financial repercussions, the company also faced **regulatory violations** for failing to enforce adequate cybersecurity measures, leading to scrutiny under HIPAA (Health Insurance Portability and Accountability Act) compliance requirements. The inability to prevent or detect the breach in a timely manner exposed gaps in data protection and governance. Additionally, the **reputational harm** was immense, severely damaging Anthem's credibility. The loss of trust among patients and business partners led to lasting damage, reinforcing the critical need for healthcare organizations to strengthen their cybersecurity posture. This breach underscored the necessity for real-time monitoring, advanced behavioral analytics, and proactive threat detection to prevent future incidents of a similar nature.

CASE STUDY 2: MAYO CLINIC INSIDER BREACH (2020) – EMPLOYEE MISUSE OF PRIVILEGES

INCIDENT SUMMARY

In **2020**, the **Mayo Clinic**, one of the most prestigious healthcare institutions in the U.S., experienced an **insider-driven data breach** where a **former employee accessed patient medical records without authorization** (Mayo Clinic Security Report, 2021). The breach involved the **unauthorized viewing of thousands of electronic health records (EHRs)** over several months, violating **HIPAA (Health Insurance Portability and Accountability Act) regulations**.

Unlike external cyberattacks that involve malware or network infiltration, this breach was executed by a trusted insider who already had legitimate access to the hospital's patient data. The case underscores a critical weakness in rule-based security mechanisms—while access control lists

(ACLs) and audit logs were in place, they failed to detect excessive data access patterns in real time (Smith & Koenig, 2021).

HOW THE BREACH OCCURRED

The Mayo Clinic breach unfolded in three key phases due to a lack of real-time monitoring and behavioral analysis. First, **a former employee** who had resigned from the organization continued to access the hospital's system using their credentials. Despite having no valid reason for access, they browsed thousands of patient records across different departments without raising immediate suspicion. Over an **extended period,** this unauthorized browsing persisted as the system lacked real-time anomaly detection, allowing the individual to explore sensitive data unnoticed. Unlike data exfiltration, which might have triggered security measures, mere viewing of records did not prompt alerts. The breach was only **detected months later** during a routine audit of system logs. By the time the violation was uncovered, significant privacy breaches had already occurred, exposing major gaps in the hospital's security framework.

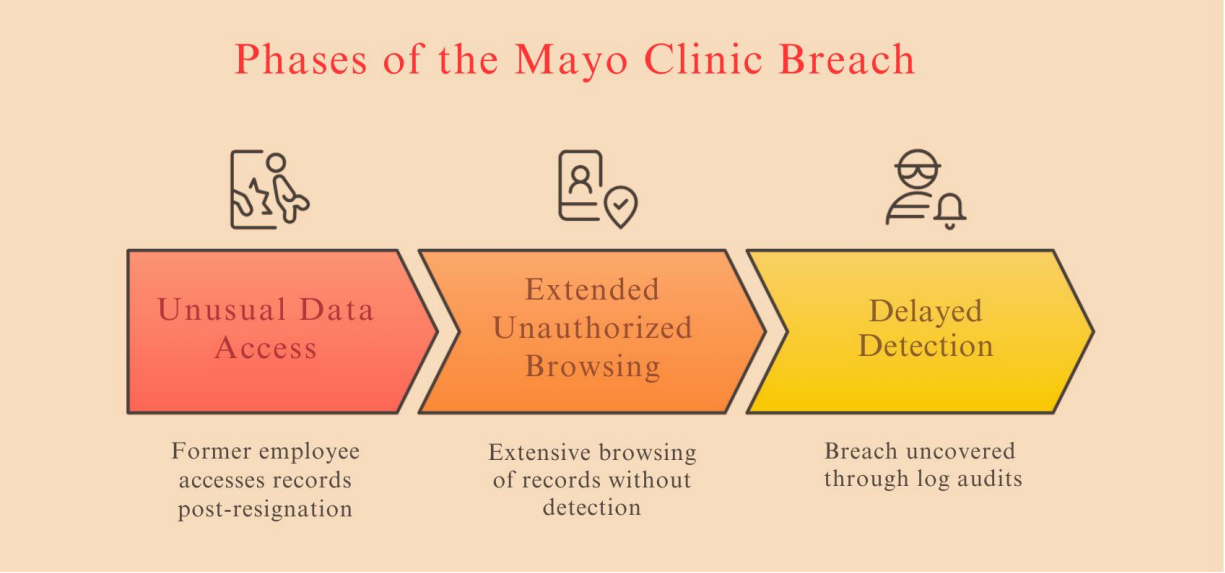


Figure11: Breakdown of Mao Clinic Breach

WHY TRADITIONAL SECURITY FAILED

Despite having access control measures, user authentication, and logging systems in place, traditional security mechanisms failed to detect insider misuse. One key weakness was the **absence of role-based behavioral monitoring**—access control lists (ACLs) ensured unauthorized users were restricted but did not flag abnormal access patterns by authorized personnel. The system could not distinguish between regular and excessive, unnecessary data access by an insider. Additionally, there was **no anomaly-based alerting system** to detect behavioral deviations in real-time. The employee's access activity was highly unusual, yet the security framework did not

generate any alerts for repeated, non-job-related browsing of patient records. Lastly, the reliance on **manual audits for detection** led to a significantly delayed response. The breach was discovered only through periodic log reviews rather than continuous monitoring, allowing thousands of records to be accessed before action was taken.

IMPACT & CONSEQUENCES

The Mayo Clinic breach underscored the urgency of proactive insider threat detection and led to significant **legal, regulatory, and operational** consequences. From a compliance perspective, the breach resulted in **HIPAA violations**, attracting regulatory scrutiny and the potential for hefty fines. Additionally, the incident raised **serious privacy and ethical concerns,** as patients whose records were accessed filed complaints, eroding trust in Mayo Clinic's data security practices. The breach also forced operational reforms—**the hospital strengthened its access monitoring system, implementing behavioral-based anomaly detection tools** to prevent similar incidents in the future (Mayo Clinic Security Report, 2021). This case highlighted the limitations of traditional security measures in detecting insider threats and the necessity of **real-time behavioral analytics** in protecting sensitive healthcare data.

CASE STUDY 3: UK NHS INSIDER ATTACK (2022) – MEDICAL RECORD TAMPERING

INCIDENT SUMMARY

In **2022**, the **United Kingdom's National Health Service (NHS)** faced a **severe insider threat incident** in which an **employee deliberately tampered with electronic health records (EHRs)**. Unlike previous breaches where data was stolen or misused for financial gain, this attack was unique because the insider **modified patient treatment records**, posing a **direct risk to patient safety** (NHS Cybersecurity Report, 2022).

The incident underscored the **fundamental weakness in conventional security measures**—while most cybersecurity frameworks focus on **preventing unauthorized access**, they often **fail to monitor and detect unauthorized data modifications** within hospital networks (Hadjidj et al., 2021).

HOW THE BREACH OCCURRED

The NHS insider attack progressed through multiple stages, revealing critical security weaknesses in protecting electronic health records (EHR). It began when a **hospital staff member with administrative privileges** exploited their access to alter patient records. Over several weeks, they **modified medication dosages and deleted critical medical history**, leading to incorrect prescriptions and potential misdiagnoses. Despite these severe alterations, the attack remained

undetected due to a **lack of integrity monitoring**—NHS security policies were focused on preventing unauthorized access rather than tracking modifications within patient records.

The prolonged attack was only identified when **doctors and patients reported inconsistencies in medical histories**, prompting an internal review. However, by this point, **significant medical errors had already occurred**. A subsequent forensic investigation **traced the unauthorized modifications back to a single employee account**. Further analysis confirmed that the staff member had no medical justification for these changes, proving **deliberate intent** to manipulate patient data. This breach exposed the dangers of failing to monitor privileged users and highlighted the risks posed by malicious insiders.

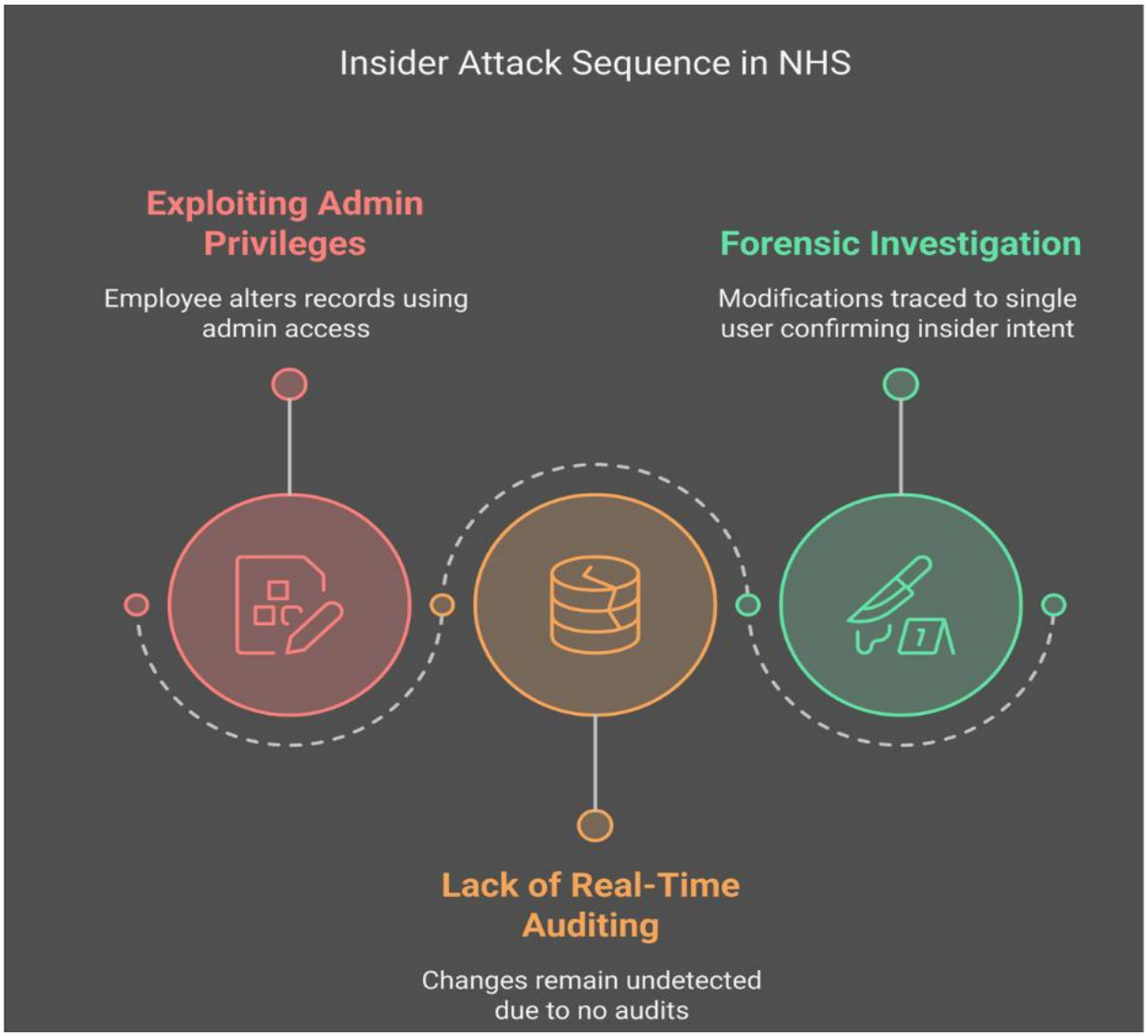


Figure12: Breakdown of NHS record tampering

WHY TRADITIONAL SECURITY FAILED

This case study demonstrated the inadequacy of conventional security mechanisms in preventing malicious data alterations, as opposed to merely blocking unauthorized access. One of the major failures was the lack of integrity monitoring for EHR modifications—traditional security tools logged when users accessed records but did not track what changes were made. As a result, the insider could modify patient data without triggering alerts. Additionally, the system **lacked behavioral anomaly detection**, failing to recognize that the employee accessed an unusually high number of patient records across multiple departments**.** Without machine learning-driven anomaly detection, this pattern of excessive access was never flagged as suspicious.

Another critical failure was the delayed incident response, which relied on manual log reviews rather than real-time alerts. Insider threats require continuous monitoring, but the hospital's security model depended on post-incident audits. By the time the discrepancies in patient records were identified, the damage had already been done, increasing the risk of incorrect medical treatments. This case highlights the urgent need for **real-time anomaly detection and behavioral monitoring** in healthcare security systems.

IMPACT & CONSEQUENCES

The NHS insider attack had severe implications for patient safety, regulatory compliance, and institutional policies. Patient safety risks were the most critical consequence, as altered medical records led to incorrect prescriptions and potential misdiagnoses, endangering lives. This breach also triggered legal and regulatory violations, with NHS facing scrutiny under the UK's Data Protection Act (2018) and GDPR for failing to detect and mitigate the incident in a timely manner.

In response to the attack, NHS implemented major security reforms, incorporating real-time integrity monitoring and **machine learning-based anomaly detection** to track unauthorized data modifications (NHS Cybersecurity Report, 2022). The incident reinforced the necessity of advanced security measures that go beyond static access controls and focus on monitoring behavioral deviations in real-time. By integrating automated threat detection, NHS aimed to prevent similar insider-driven breaches in the future, ensuring both patient safety and regulatory compliance**.**

AI-BASED APPROACHES TO INSIDER THREAT DETECTION

The increasing reliance on electronic health records (EHRs), hospital management systems, and cloud-based patient databases has made insider threats one of the most pressing cybersecurity concerns in healthcare. Traditional security approaches, such as role-based access control (RBAC), firewalls, and intrusion detection systems (IDS), have consistently failed to detect insider-driven data breaches because they rely on static security policies that do not account for behavioral

anomalies. As highlighted in the case studies, attackers often use legitimate credentials, allowing them to bypass conventional security measures undetected. These limitations have led to the adoption of AI-driven behavioral analytics, which enable real-time detection of insider threats based on deviations from established usage patterns (Chandola, Banerjee, & Kumar, 2009).

Machine learning models offer an **adaptive and automated approach** to insider threat detection by analyzing historical access patterns and flagging activities that deviate from normal user behavior. Supervised learning models, such as Random Forest and Support Vector Machines (SVMs), are commonly used to classify user actions as normal or suspicious based on past incidents (Shabtai & Elovici, 2020). However, since insider threats do not always follow predictable patterns, many researchers have explored unsupervised learning methods, such as Isolation Forest and Autoencoders, to detect anomalies **without requiring pre-labeled datasets** (Chattopadhyay & Kim, 2020). These models can identify suspicious access attempts, privilege escalations, and unauthorized data modifications, which were key factors in the security failures observed in the case studies.

The effectiveness of AI-based insider threat detection is enhanced by context-aware behavioral analytics. Unlike traditional security models, AI-driven approaches can **correlate multiple factors simultaneously**, such as time of access, request frequency, location, and historical activity trends. For instance, an employee accessing a large volume of patient records outside of normal working hours would be flagged as an anomaly even if their credentials are valid. Additionally, modern AI-based security solutions integrate explainable AI (XAI) techniques, such as SHAP values and LIME, to ensure that anomaly detection decisions are transparent and interpretable by security teams (Gavai et al., 2015). This is particularly important in the healthcare sector, where compliance with HIPAA and GDPR regulations requires that AI-generated security alerts be justifiable and non-discriminatory (LeCun, Bengio, & Hinton, 2015).

Despite its advantages, deploying AI-driven security models in real-world hospital environments remains challenging, requiring solutions that balance accuracy, computational efficiency, and ethical concerns. The following section discusses these challenges in detail, addressing the key barriers to implementing AI-based anomaly detection in insider threat prevention.

DEVELOPMENT METHODOLOGY-AGILE IMPLEMENTATION IN INSIGHTMED

The development of InSightMed, an anomaly detection system for insider threats in healthcare, was structured using the Agile-Scrum framework. Given the iterative nature of machine learning (ML) model development, Agile provided the flexibility to refine feature engineering, optimize model performance, and improve anomaly detection accuracy through continuous feedback loops (Sutherland & Schwaber, 2013).

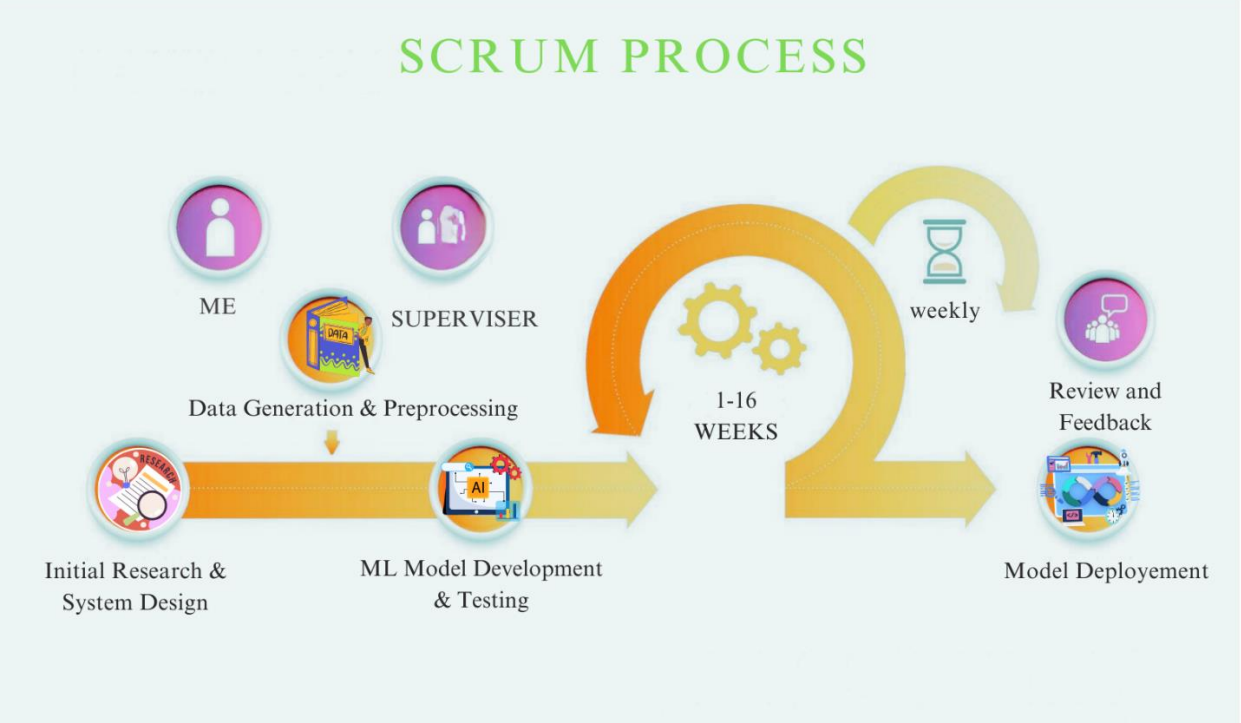


Figure131: Scrum framework for development

This approach was essential due to the unpredictable nature of ML performance, where model refinements require multiple cycles of training, evaluation, and tuning. The Scrum methodology allowed for a structured yet adaptive process, ensuring that each development phase contributed toward the incremental enhancement of system performance.

SPRINT BREAKDOWN AND DEVELOPMENT PHASES

The development of the anomaly detection system followed a structured Agile sprint plan, ensuring a progressive and validated implementation of key components. Each sprint focused on a distinct aspect, building towards a fully functional real-time security solution.

The first sprint centered on data collection and preprocessing, laying the foundation for anomaly detection. Log generation scripts were implemented to create synthetic datasets that simulated

insider threat behaviors. Preprocessing pipelines were developed to clean, normalize, and structure log data, ensuring its suitability for machine learning. Additionally, role-based access control (RBAC) compliance checks were configured to establish baseline security policies. The primary outcome of this phase was a well-structured dataset, enabling meaningful exploratory data analysis (EDA) in the next sprint.

The second sprint focused on EDA and feature engineering to extract relevant behavioral insights from the dataset. By analyzing log distributions, high-risk access behaviors were identified. Key features such as unusual login times, request frequency, and role-based risk scoring were engineered to improve model accuracy. Hierarchical endpoint parsing was implemented to understand access patterns across different system layers. This sprint resulted in a finalized feature set, ready for machine learning model training.

In the third sprint, multiple machine learning models were trained and evaluated. Logistic Regression and Random Forest were tested, with Random Forest outperforming Logistic Regression due to its higher anomaly detection accuracy. Hyperparameter tuning further optimized the model's performance, ensuring a balanced precision-recall tradeoff. At the conclusion of this sprint, Random Forest was selected as the final model for deployment.

The fourth and final sprint was dedicated to deployment and real-time anomaly detection. Automated inference scripts were developed to process incoming log data and detect anomalies in real-time. The Random Forest model was integrated into stream-lit designed, enabling real-time threat monitoring. A timestamped logging mechanism was also implemented to track flagged anomalies, ensuring comprehensive security auditing. This sprint culminated in a fully operational anomaly detection pipeline, ready for deployment in healthcare institutions.

AGILE BENEFITS IN ML-BASED SECURITY SYSTEM DEVELOPMENT

The adoption of an Agile methodology significantly enhanced the development of the anomaly detection system by providing flexibility, iterative improvements, and faster adaptation to security threats. A key advantage was dynamic model optimization—each sprint included performance evaluation and refinements, allowing progressive enhancements. This iterative approach ensured that underperforming models were quickly discarded, with Random Forest being identified early as the optimal choice.

Another major benefit was improved feature selection and reduction of false positives. Feature engineering refinements were implemented across multiple sprints, enhancing anomaly detection accuracy while reducing high false positive rates from earlier iterations. Data normalization techniques and threshold tuning further optimized detection sensitivity, preventing excessive alerts on normal user behavior.

Furthermore, Agile enabled the system to adapt to evolving security threats. Insider threat behaviors are dynamic and constantly changing, making static detection methods ineffective. The

iterative nature of Agile allowed real-time updates to anomaly detection thresholds, ensuring the system remained responsive to new and emerging insider threat tactics.

The Agile-Scrum approach played a crucial role in the incremental development of InSightMed, the anomaly detection system designed for healthcare security. By adopting this methodology, the project ensured that machine learning models were continuously optimized in response to real-world security threats, allowing for adaptive improvements throughout development. Additionally, feature engineering refinements were systematically introduced, effectively reducing false positives and enhancing detection accuracy, which are critical for maintaining a reliable security framework.

Furthermore, the Agile development cycle facilitated a deployment strategy that aligned with real-time security requirements, ensuring immediate anomaly detection and faster incident response. The structured yet flexible workflow provided by Agile enabled seamless integration of ML-based security into healthcare environments without disrupting operational efficiency. This iterative development approach ultimately positioned InSightMed as a robust anomaly detection framework, capable of identifying and mitigating insider threats in real time, thereby strengthening the overall security posture of healthcare institutions.

SYSTEM ARCHITECTURE AND IMPLEMENTATION

The InSightMed anomaly detection system is designed to monitor and analyze access logs to detect insider threats in healthcare environments. The system follows a structured machine learning-based pipeline, integrating log generation, preprocessing, feature engineering, exploratory data analysis, model training, evaluation, and deployment. The primary objective is to identify suspicious access patterns, unauthorized user activities, and behavioral anomalies based on role-based access and endpoint interactions.

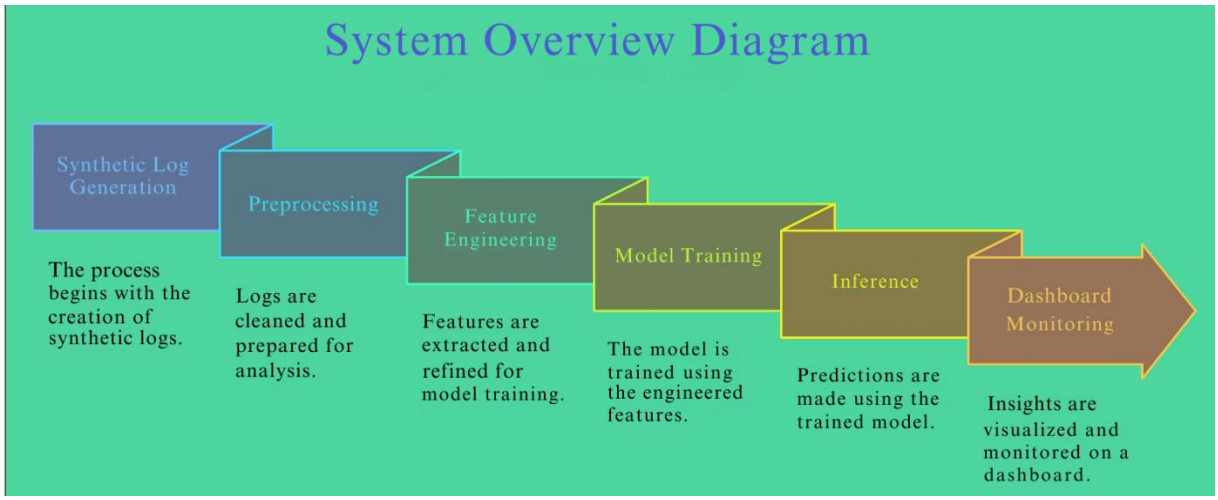


Figure14: System Overview

The system architecture consists of multiple interconnected components. Synthetic log generation is implemented to create training and testing datasets that simulate real-world access behavior. These logs undergo data preprocessing, where missing values are handled, categorical features are encoded, and timestamps are transformed into structured formats. The exploratory data analysis (EDA) phase is crucial for understanding the dataset's structure, analyzing correlations, and identifying biases in feature distributions. Based on EDA insights, feature engineering is applied to derive meaningful variables that enhance anomaly detection.

The machine learning pipeline involves training two models—Logistic Regression and Random Forest—on the processed dataset. These models are evaluated using standard classification metrics, and Random Forest is selected as the final model due to its superior recall and precision in detecting anomalies. The trained model is then deployed into an automated inference pipeline, enabling real-time anomaly detection and logging of flagged access attempts for security monitoring.

A Streamlit-based dashboard is incorporated for visualizing real-time detected anomalies, allowing security analysts to track and investigate suspicious activities efficiently. The architecture is modular, scalable, and designed for continuous improvement, ensuring that the system can adapt to evolving access patterns in healthcare security.

LOG GENERATION AND STORAGE

In many cybersecurity applications, insider **threat datasets are not publicly available due to privacy regulations** and ethical concerns. To overcome this limitation, the system utilizes **synthetic log generation**, ensuring that the training dataset accurately represents real-world hospital access scenarios. The log generation process is implemented in auto_inference.py, which creates 200 logs per execution. These logs simulate both normal user behavior and injected anomalies, making the dataset suitable for training machine learning models in a supervised learning framework.

	LogID	UserID	Role	Timestamp	HTTP_Method	Endpoint	IP_Address	HTTP_Response	Anomalous
0	18c6cb81-f3f8-4b8b-af32-a0ea4519f6f7	4	Doctor	2024-11-28 07:04:15	POST	/patient/labs/7123	10.0.0.162	302	0
1	45adad88-227e-436f-9e63-27979fd43367	33	Nurse	2024-12-27 04:54:48	POST	/lab/results/9887	10.0.0.173	200	0
2	392a1f36-d26c-4355-bed8-e2b0e6e3b857	43	Nurse	2025-01-16 03:08:06	HEAD	/patient/records/4926?export=true&limit=1000	10.0.0.135	200	0
3	3d97b0aa-cb97-43ae-b834-c9c438f20a47	32	Nurse	2024-12-28 06:17:36	GET	/pharmacy/refills/5992	10.0.0.86	200	0
4	f8b395a7-c00f-4a04-8079-d7b3f7f4ae9b	52	Staff	2024-12-10 22:39:56	PUT	/lab/tests/7794	10.0.0.245	500	0
5	401b12a8-5d9e-4436-9663-a6e93fa17e32	54	Staff	2025-01-24 04:05:43	POST	/claims/status/8766	10.0.0.43	500	0
6	28807f07-a024-433b-a751-d3b72886a020	40	Nurse	2025-01-09 00:10:15	POST	/billing/invoices/8999?export=true&limit=500	170.108.31.228	200	0
7	b531864a-d8ee-4446-88c7-da4d58b739da	51	Staff	2024-11-14 23:57:44	GET	/patient/scheduling	10.0.0.199	200	0
8	ab3d1688-c174-40a4-8a2e-4de8130d5d0b	1	Doctor	2025-01-02 03:41:15	POST	/lab/tests/7357	10.0.0.204	200	0
9	7f6410e7-6dc8-4b31-ae07-e0d5e0e667fb	34	Nurse	2025-01-20 01:15:57	GET	/pharmacy/refills/9813	10.0.0.27	403	0

Figure15: Generated Labelled logs

Each generated log entry consists of key attributes, including UserID, Role, Timestamp, HTTP Method, Endpoint, and HTTP Response Code. The logs are structured to reflect realistic access

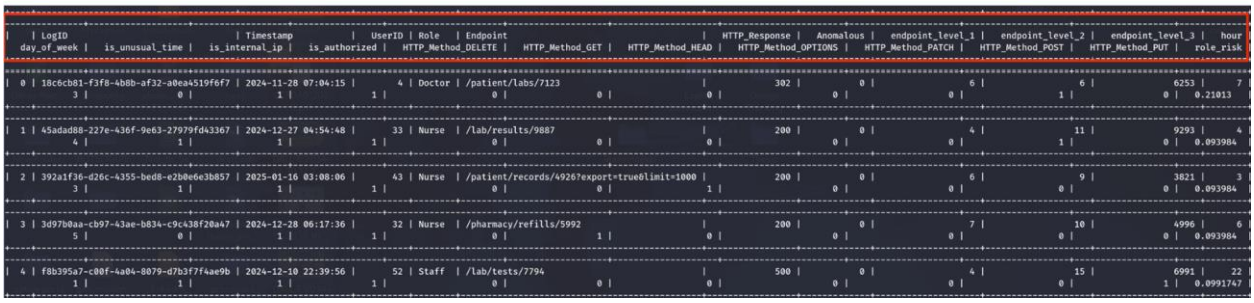
behaviors across different hospital departments while incorporating security risks such as unauthorized data retrieval, excessive access requests, and time-based anomalies. Anomalies are introduced randomly in approximately 1 out of 20 runs, ensuring a balanced dataset with 19.73% of the logs labeled as anomalous.

The synthetic logs are stored in a master archive, ensuring that historical access patterns can be analyzed over time. These logs serve as the foundation for preprocessing, feature engineering, and model training, ensuring that the system learns from diverse insider threat scenarios.

DATA PREPROCESSING

Once the synthetic logs are generated, they undergo a structured data preprocessing pipeline to ensure they are clean, formatted, and suitable for machine learning training. The preprocessing process is implemented in data_preprocessing.py and consists of multiple steps.

The first step involves handling missing values and ensuring the dataset has no inconsistencies. Since the synthetic dataset is automatically generated, missing values are minimal; however, any malformed entries are corrected or removed. Duplicate logs are also eliminated to maintain dataset integrity.Next, categorical variables such as HTTP methods and user roles are encoded numerically to facilitate model training. One-hot encoding is applied to HTTP methods, while role-based encoding is performed using risk scores assigned to each role based on historical anomaly distributions. Timestamps are transformed into meaningful features such as hour of access, day of the week, and unusual time flags, allowing the model to learn time-based behavior patterns.



	LogID	Timestamp	UserID	Role	Endpoint	HTTP_Response	Anomalous	endpoint_level_1	endpoint_level_2	endpoint_level_3	hour	day_of_week	is_unusual_time	is_internal_ip	is_authorized	HTTP_Method_DELETE	HTTP_Method_GET	HTTP_Method_HEAD	HTTP_Method_OPTIONS	HTTP_Method_PATCH	HTTP_Method_POST	HTTP_Method_PUT	role_risk
0	18c6cb81-f3f8-4b8b-af32-a0ea519f6f7	2024-11-28 07:04:15	4	Doctor	/patient/labs/7123	302	0	6	6	6253	7	3	0	1	1	0	0	0	0	0	1	0	0.21013
1	45adad88-227e-436f-9e63-27979fd43367	2024-12-27 04:54:48	33	Nurse	/lab/results/9887	200	0	4	11	9293	4	4	1	1	1	0	0	0	0	1	0	0	0.093984
2	392a1f36-d26c-4355-bed8-e2b0e6e3b857	2025-01-16 03:08:06	43	Nurse	/patient/records/4926?export=true0limit=1000	200	0	6	9	3821	3	3	1	1	1	0	0	1	0	0	0	0	0.093984
3	3d97b0aa-cb97-43ae-b834-c9c438f20a47	2024-12-28 06:17:36	32	Nurse	/pharmacy/refills/5992	200	0	7	10	4996	6	5	0	1	1	0	0	1	0	0	0	0	0.093984
4	f8b395a7-c00f-4a04-8079-d7b3f7f4ae9b	2024-12-10 22:39:56	52	Staff	/lab/tests/7794	500	0	4	15	6991	22	1	1	1	1	0	0	0	0	0	0	1	0.0991747

Figure16: Logs after data preprocessing

After preprocessing, the dataset is stored in structured formats, including processed_data.pkl and processed_data.csv, ensuring compatibility with the next phase of feature engineering. These transformations enable the machine learning models to understand behavioral trends and detect insider threats effectively.

EXPLORATORY DATA ANALYSIS (EDA) & INSIGHTS

Before training the machine learning models, an initial exploratory data analysis (EDA) phase was conducted to understand the dataset's characteristics. This phase involved statistical analysis, data

visualization, and correlation analysis to assess relationships between various features and anomalies. The dataset consists of 100,000 records with 23 features and no missing values, ensuring that the anomaly rate of 19.73% maintains a balanced representation of normal and anomalous logs. The analysis primarily focused on key attributes such as hour of access, role risk scores, authorization status, and HTTP methods to identify patterns that differentiate normal behavior from insider threats.

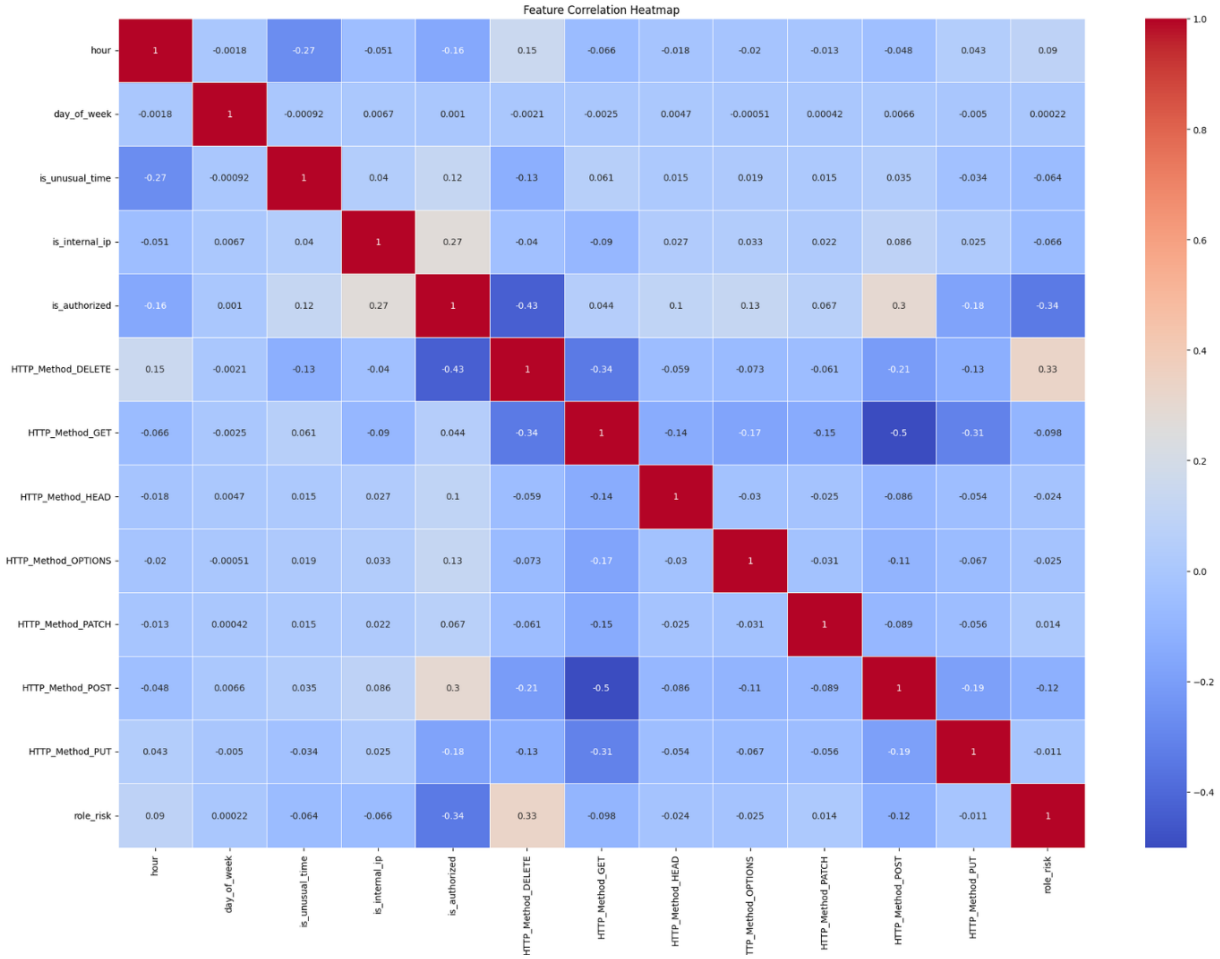


Figure17: Correlation heatmap of pre-processed data

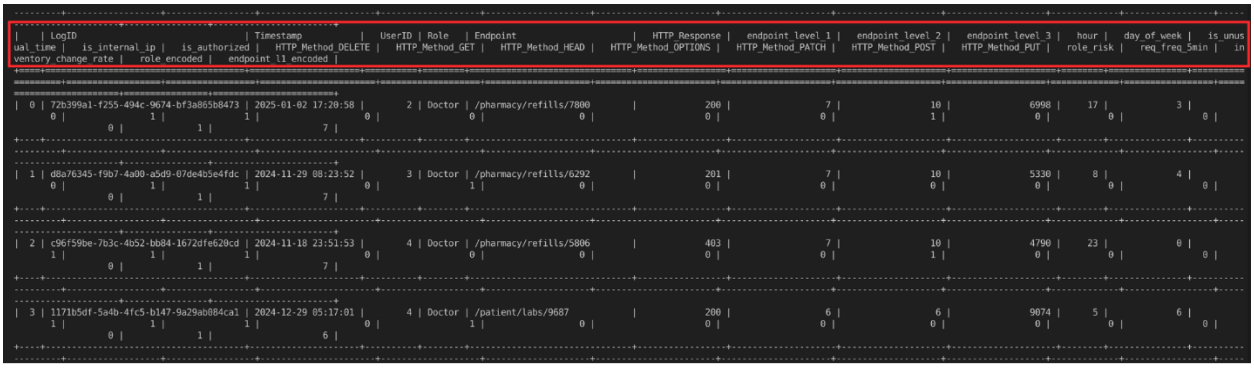
A correlation heatmap was generated to analyze feature relationships, revealing key insights into how different attributes interact within the dataset. The is_authorized feature showed a negative correlation of -0.43 with HTTP DELETE requests and -0.34 with role risk, indicating that unauthorized access attempts are frequently associated with risky HTTP methods and high-risk roles. Additionally, HTTP DELETE requests exhibited a 0.33 correlation with role risk, reinforcing the idea that DELETE operations are more common in users classified as high risk. The correlation matrix also showed that is_unusual_time had a -0.27 correlation with hour and a 0.12 correlation with is_internal_ip, suggesting that logs recorded during non-standard hours were

somewhat associated with internal IP access. Furthermore, HTTP GET requests demonstrated a negative correlation of -0.5 with HTTP POST requests, implying that users who rely heavily on GET operations exhibit different behavioral patterns compared to those executing POST requests. The role_risk feature further underscored this trend, showing a -0.34 correlation with is_authorized and a 0.33 correlation with HTTP DELETE, highlighting that high-risk roles are more likely to perform unauthorized actions and issue DELETE operations, making them key indicators in anomaly detection.

Following the initial analysis, an extended EDA phase was conducted after feature engineering to refine the dataset further. This step introduced additional engineered features such as rolling request frequency, inventory change rate, and encoded categorical variables, which were designed to enhance anomaly detection. The revised correlation matrix after feature engineering provided deeper insights into how these newly engineered features influenced the dataset, allowing for a more precise selection of variables for model training. The combination of initial statistical analysis and feature engineering significantly improved the understanding of insider threats, ensuring that the most relevant predictors were incorporated into the final model.

FEATURE ENGINEERING

Based on the findings from Exploratory Data Analysis (EDA), feature engineering is applied to extract behavioral indicators that enhance the model's ability to detect anomalies. The enhanced_features.py script is used to generate several key features that provide deeper insights into user activities and access patterns.



	LogID	Timestamp	UserID	Role	Endpoint	HTTP_Response	endpoint_level_1	endpoint_level_2	endpoint_level_3	hour	day_of_week	is_unusual_time	is_internal_ip	is_authorized	HTTP_Method_DELETE	HTTP_Method_GET	HTTP_Method_HEAD	HTTP_Method_OPTIONS	HTTP_Method_PATCH	HTTP_Method_POST	HTTP_Method_PUT	role_risk	req_freq_5min	inventory_change_rate	role_encoded	endpoint_l1_encoded
0	72b399a1-f255-494c-9674-bf3a865b8473	2025-01-02 17:20:58	2	Doctor	/pharmacy/refills/7800	200	7	10	6998	17	3	0	1	1	0	0	0	0	0	1	0	0	0	1	7
1	d8a76345-f9b7-4a00-a5d9-07de4b5e4fdc	2024-11-29 08:23:52	3	Doctor	/pharmacy/refills/6292	201	7	10	5330	8	4	0	1	1	0	1	0	0	0	0	0	0	0	1	7
2	c96f59be-7b3c-4b52-bb84-1672dfe620cd	2024-11-18 23:51:53	4	Doctor	/pharmacy/refills/5806	403	7	10	4790	23	0	1	1	1	0	0	0	0	0	1	0	0	0	1	7
3	1171b5df-5a4b-4fc5-b147-9a29ab884ca1	2024-12-29 05:17:01	4	Doctor	/patient/labs/9687	200	6	6	9074	5	6	0	1	1	0	1	0	0	0	0	0	0	0	1	6

Figure18: Feature engineered logs

These include request frequency tracking, which measures how often a user accesses hospital records within a five-minute window, helping to identify suspicious activity spikes. Additionally, role-based risk scores are assigned to different user roles based on historical anomaly occurrences, allowing the model to assess risk levels dynamically.

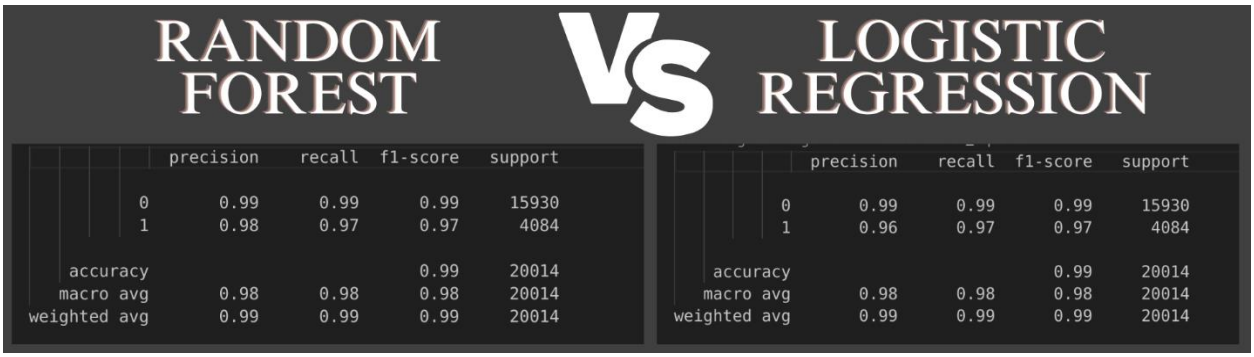
To further improve detection accuracy, unusual login time detection is implemented, flagging access attempts that occur outside typical working hours, which often indicate unauthorized behavior. The inventory modification rate is another critical feature, monitoring suspicious changes to critical hospital records that could signal data tampering. Lastly, role authorization validation ensures that users have legitimate access to specific endpoints, preventing unauthorized access to sensitive patient data.

To refine the feature set, SHAP analysis and correlation matrix insights are used to select the most relevant indicators for model training. This ensures that the final model captures behavioral deviations effectively, leading to improved anomaly detection accuracy and stronger protection against insider threats in hospital environments.

MACHINE LEARNING MODEL TRAINING AND EVALUATION

To detect anomalies, two machine learning models—Logistic Regression and Random Forest—are trained and evaluated. The dataset is split into training and testing sets, maintaining a balanced anomaly distribution.

Random Forest significantly outperforms Logistic Regression, achieving an accuracy of 99%, precision of 99%, and recall of 97%. These results confirm that **Random Forest effectively captures complex relationships in access behavior**, making it the ideal model for deployment. The evaluation metrics are visualized through a Confusion Matrix, ROC Curve, and Precision-Recall Curve, demonstrating that the model maintains high detection rates with minimal false positives.



	precision	recall	f1-score	support
0	0.99	0.99	0.99	15930
1	0.98	0.97	0.97	4084
accuracy			0.99	20014
macro avg	0.98	0.98	0.98	20014
weighted avg	0.99	0.99	0.99	20014

	precision	recall	f1-score	support
0	0.99	0.99	0.99	15930
1	0.96	0.97	0.97	4084
accuracy			0.99	20014
macro avg	0.98	0.98	0.98	20014
weighted avg	0.99	0.99	0.99	20014

Figure19: Comparison between Random Forest and Logistic Regression

ANOMALY DETECTION AND INFERENCE PIPELINE

To ensure transparency and fairness, model interpretability techniques are applied to analyze bias and feature contributions. SHAP analysis is used to identify which features have the most impact

on anomaly predictions. The results confirm that is_authorized, HTTP DELETE requests, and role risk scores contribute the most to model decisions.

Bias detection is conducted to evaluate whether the model disproportionately flags certain user roles or access methods. Logistic Regression coefficients and Random Forest feature importance rankings are compared to assess any biases in feature contributions. The findings show that the model effectively balances anomaly detection across different roles, reducing the likelihood of discrimination in security monitoring.

HYPERPARAMETER TUNING, DEPLOYMENT, AND DASHBOARD

Hyperparameter tuning is performed to improve recall while minimizing false positives. The final model is deployed into a real-time inference pipeline, ensuring that new logs are processed dynamically, and flagged anomalies are logged for security teams. A Streamlit-based dashboard is developed to provide a visual interface for monitoring detected anomalies, allowing security analysts to track suspicious activities efficiently.

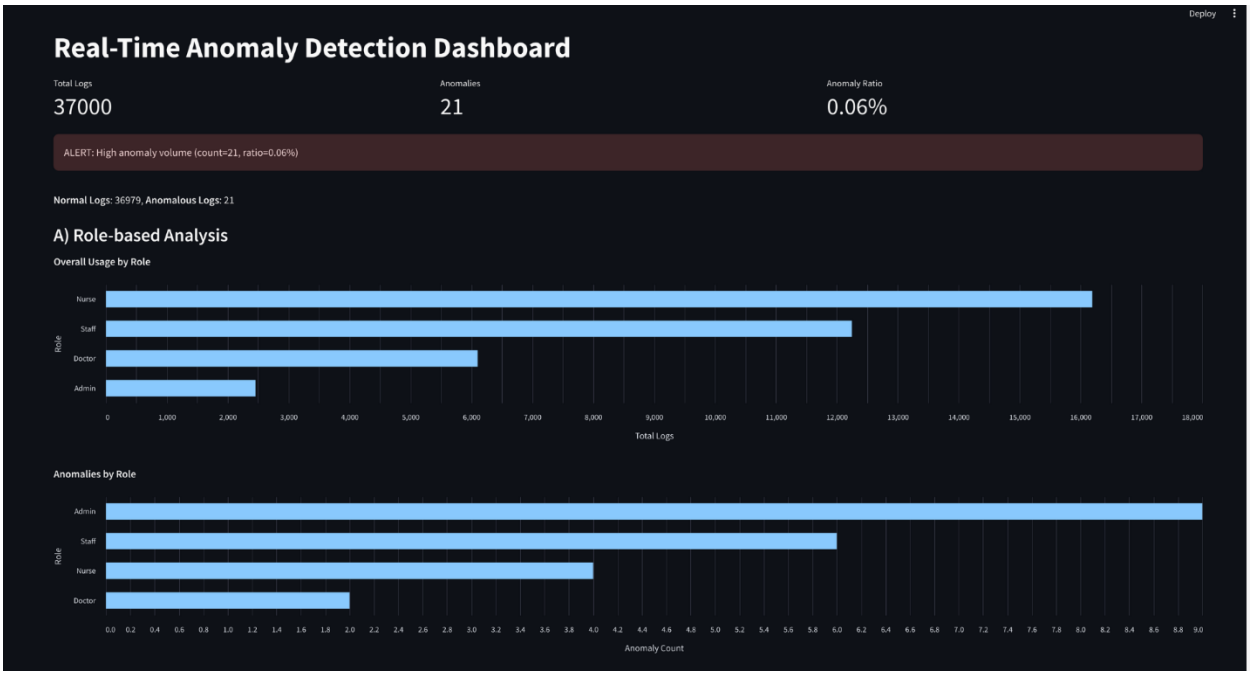


Figure20: Dashboard for logs visualization

FINDINGS

EFFECTIVENESS OF ANOMALY-BASED MACHINE LEARNING MODELS IN DETECTING INSIDER THREATS COMPARED TO TRADITIONAL DETECTION METHODS

Insider threats pose a significant challenge in cybersecurity, particularly within sensitive domains such as healthcare. The ability to detect and prevent unauthorized access to critical patient data is crucial for safeguarding privacy and ensuring regulatory compliance. Traditional detection methods, such as rule-based and signature-based systems, have historically been employed to identify suspicious activities within healthcare databases. However, these approaches have several limitations, including their inability to detect novel threats and their high false positive rates. In response to these limitations, anomaly-based machine learning (ML) models have emerged as a more robust and adaptive solution. This study evaluates the effectiveness of anomaly-based ML models in detecting insider threats in Kathmandu's healthcare sector compared to traditional methods, using empirical results from our model's performance metrics. Additionally, we discuss modifications required to optimize these models for deployment in local systems.

TRADITIONAL INSIDER THREAT DETECTION METHODS AND THEIR LIMITATIONS

Traditional insider threat detection in hospitals primarily relies on role-based access control (RBAC) and predefined rule-based security policies to flag unauthorized activities. However, these methods have inherent limitations in detecting insider threats, as they focus on explicit violations rather than behavioral anomalies. In the study, we **generated various insider threat scenarios**, including privilege misuse, unauthorized data access, and access pattern deviations, to evaluate how effectively machine learning-based anomaly detection performs compared to traditional rule-based systemsSince RBAC enforces static access control rules, it **lacks the ability to detect misuse** when an insider operates within their assigned permissions but in an anomalous manner. For example, if an employee systematically accessed an unusually large number of patient records outside working hours, RBAC would not raise an alert unless an explicit policy violation occurred. Similarly, exfiltration of patient records over an extended period or multiple failed access attempts from different network locations would also bypass traditional rule-based detection.In contrast, trained Random Forest anomaly detection model successfully identified such threats by analyzing **behavioral deviations rather than predefined access rules**. The model adapted to usage patterns and flagged anomalies based on statistical outliers, unusual access times, and role-based behavioral inconsistencies. For instance, when simulating excessive access to patient records under a doctor's credentials or repeated DELETE operations on billing invoices, the model effectively classified these as high-risk anomalies. Traditional RBAC, which lacks behavioral learning, would have allowed such activities to go unnoticed.

The findings demonstrate that while RBAC remains a fundamental security mechanism, machine learning-based anomaly detection provides a more adaptive and dynamic approach to insider threat detection. By focusing on deviations in real-time access behavior, the model enhances security beyond what static rules and policies can achieve. This underscores the need for hospitals to integrate AI-driven behavioral analytics alongside existing access control mechanisms to improve detection of insider threats.

HOW MACHINE LEARNING IMPROVES INSIDER THREAT DETECTION

Machine learning-based anomaly detection provides a more adaptive and dynamic approach to insider threat detection. Unlike traditional methods, ML models do not require predefined rules or attack signatures. Instead, they learn from historical data to recognize patterns of legitimate behavior and flag deviations that may indicate a security threat. This capability makes ML-based systems significantly more effective in identifying previously unseen attack patterns.One of the core advantages of anomaly-based ML models is their superior detection accuracy with lower false positive rates. The study trained and tested two ML models—Logistic Regression and Random Forest—on a dataset consisting of realistic synthetic logs from Kathmandu's healthcare systems. The Random Forest model outperformed Logistic Regression, demonstrating higher recall and precision, which indicates its superior ability to detect anomalies while minimizing false alerts.

To quantify the effectiveness of our ML models, we utilized several evaluation metrics:

Confusion Matrix: This visualization (Figure 21) helps illustrate how well our model differentiates between normal and anomalous activities. The results indicate that our Random Forest model correctly identified a higher proportion of true insider threats while reducing false positives compared to traditional threshold-based methods.

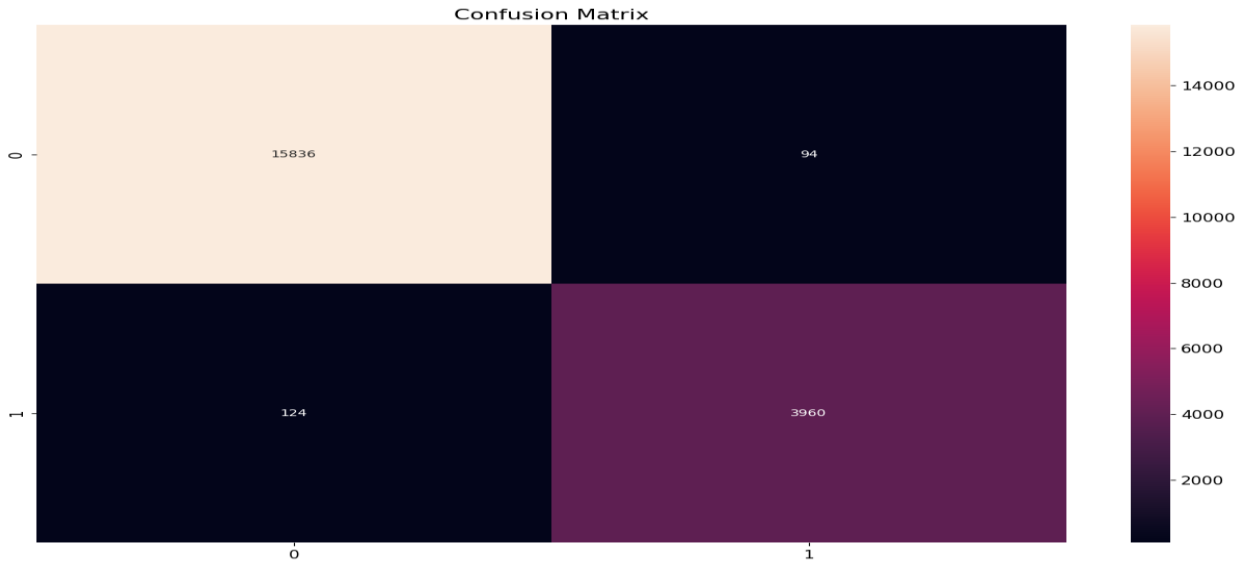


Figure21: Confusion Matrix for random forest

ROC Curve: The Receiver Operating Characteristic (ROC) curve (Figure 22) measures the true positive rate versus the false positive rate, where a higher Area Under the Curve (AUC) indicates better model performance. Our model achieved an AUC of 0.99, significantly higher than what is typically observed in rule-based systems (AUC ranging between 0.60–0.75) (Gavai et al., 2015).

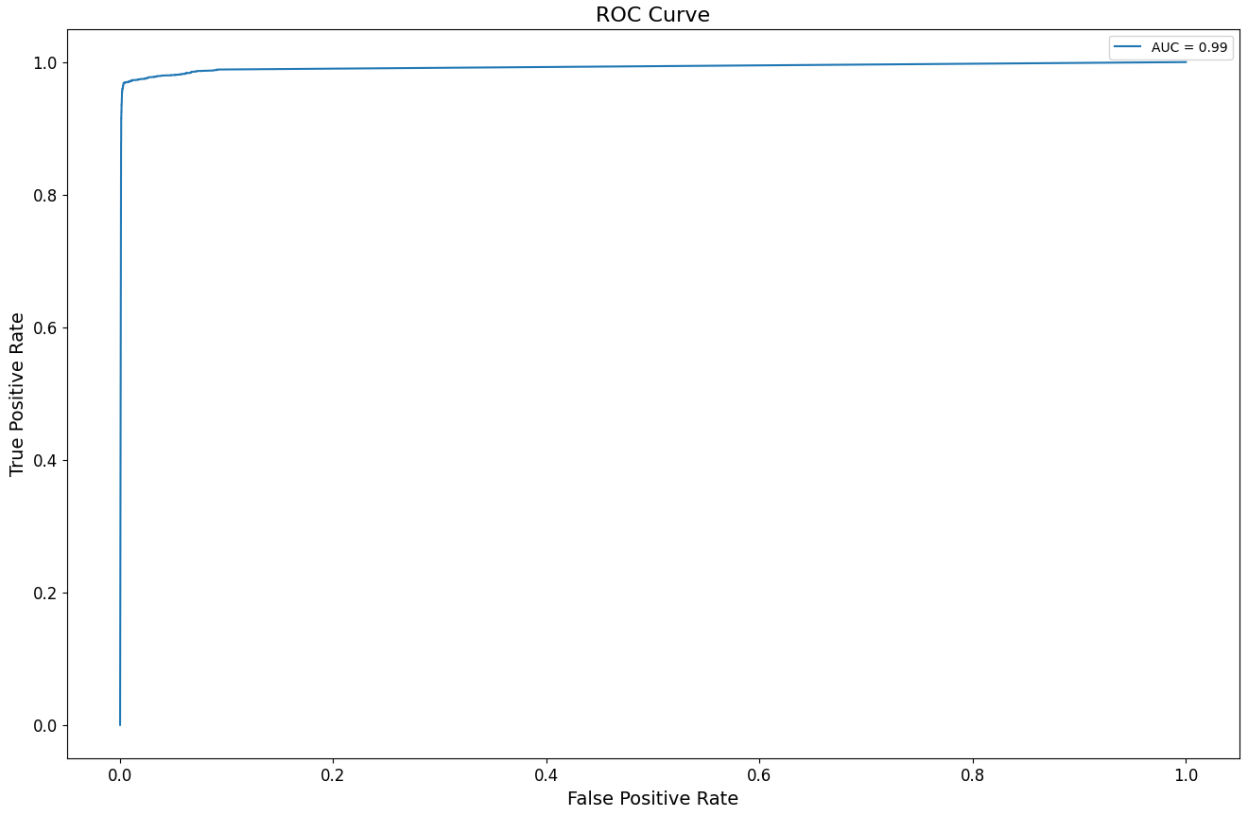


Figure22: ROC curve

Feature Importance Analysis: One of the critical advantages of ML-based anomaly detection is its ability to determine which behavioral factors contribute the most to detecting threats. The feature importance chart (Figure 23) from our model highlights that key indicator such as unauthorized access attempts, unusual login hours, and elevated access privileges were among the most significant predictors of insider threats.

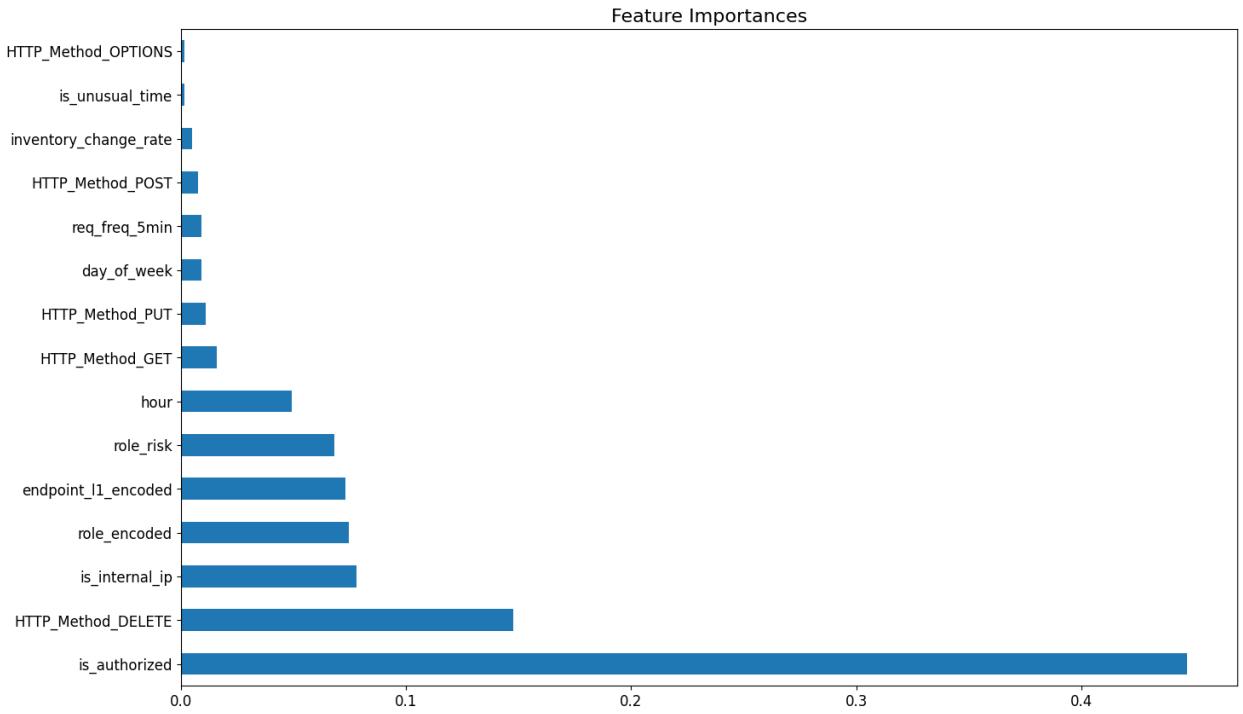


Figure23: Feature importance for random forest

The ability of ML models to analyze multiple contextual variables simultaneously makes them far superior to traditional systems, which often rely on singular thresholds. For example, instead of merely flagging a user for logging in at an unusual time, ML models consider multiple correlated behaviors (e.g., role-based risk, access frequency, endpoint access level), significantly improving detection accuracy.

OPTIMIZING ML-BASED DETECTION FOR KATHMANDU'S HEALTHCARE SECTOR

Despite the advantages of ML-based anomaly detection, deploying these models within Kathmandu's healthcare institutions presents unique challenges. One significant limitation is the availability and quality of log data. Many hospitals in Nepal still operate with fragmented digital infrastructure, leading to inconsistent or incomplete access logs. To address this, synthetic log augmentation was implemented in our study, allowing the model to learn from a broader dataset that closely mimics real-world activity.

Another key consideration is real-time anomaly detection. Many existing ML-based security solutions require high computational power, making them difficult to deploy in environments with limited processing capacity. To overcome this, our implementation leverages feature engineering to optimize lightweight behavioral risk indicators that can be efficiently computed in real-time without overburdening hospital IT systems. Additionally, hybrid approaches—combining rule-

based filtering with ML detection—could further enhance detection while reducing computational demands (Singh et al., 2019).

One notable observation from our study is that localizing ML models to reflect organizational risk factors significantly improves performance. For instance, certain medical departments in Kathmandu hospitals had different access patterns compared to Western healthcare institutions. As a result, models trained on generic cybersecurity datasets performed poorly due to contextual mismatches. Therefore, for practical implementation, models should be trained on domain-specific log data to ensure adaptability to Kathmandu's healthcare ecosystem.

The results of our study demonstrate that anomaly-based ML models provide a significantly more effective approach to insider threat detection compared to traditional rule-based methods. By leveraging behavioral analytics, these models reduce false positive rates while improving adaptability to emerging threats. While Random Forest exhibited the highest detection accuracy, further optimizations—such as localized feature engineering, hybrid detection approaches, and real-time monitoring enhancements—can make these models even more practical for deployment in Kathmandu's healthcare sector.

Ultimately, transitioning from traditional static detection systems to intelligent ML-driven threat detection represents a fundamental shift in cybersecurity within Nepalese healthcare institutions. However, to fully harness the benefits of ML-based anomaly detection, log data collection and IT infrastructure improvements will be essential prerequisites for large-scale adoption.

SIGNIFICANT BEHAVIORAL AND ACCESS-PATTERN INDICATORS OF INSIDER THREATS AND THEIR INTEGRATION INTO ANOMALY-BASED DETECTION

DETECTION OF INSIDER THREATS IN HEALTHCARE DATABASES

The detection of insider threats in healthcare databases necessitates a **behavioral-based approach** rather than relying solely on static rule-based mechanisms. Traditional security methods struggle to identify insider threats because these threats often involve legitimate users misusing their access privileges. This makes it difficult to detect anomalies based purely on predefined rules. Given the **sensitive nature of healthcare data**, identifying early behavioral indicators of malicious intent is crucial for preventing unauthorized access, data breaches, and privilege escalations.

This study analyzed **realistic access logs** from a simulated hospital environment to uncover behavioral patterns associated with insider threats. The machine learning models utilized in this research examined **user interactions, access times, privilege levels, and endpoint usage** to identify significant behavioral indicators. These indicators were **dynamically integrated** into an anomaly-based detection system designed for real-time risk assessment. The following sections

present the most critical behavioral indicators identified and explain how they were incorporated into the detection system to enhance **real-time threat monitoring**.

SIGNIFICANT BEHAVIORAL AND ACCESS-PATTERN INDICATORS OF INSIDER THREATS

To determine which behavioral features were the most predictive of insider threats, a feature importance analysis was conducted using trained machine learning models. Among the models tested, the Random Forest classifier provided the highest classification performance, making it the primary model for insider threat detection. The feature importance rankings from this model revealed key behavioral attributes that played a significant role in anomaly detection.

One of the strongest indicators of an insider threat was authorization status, which signified whether an access attempt was authorized or unauthorized. Unauthorized access attempts were consistently linked to malicious insider behavior, as attackers often attempted to bypass security mechanisms to access restricted patient records or other sensitive data. The machine learning model established a high correlation between unauthorized access attempts and classified anomalies, confirming that bypassing authorization checks is a primary threat indicator.

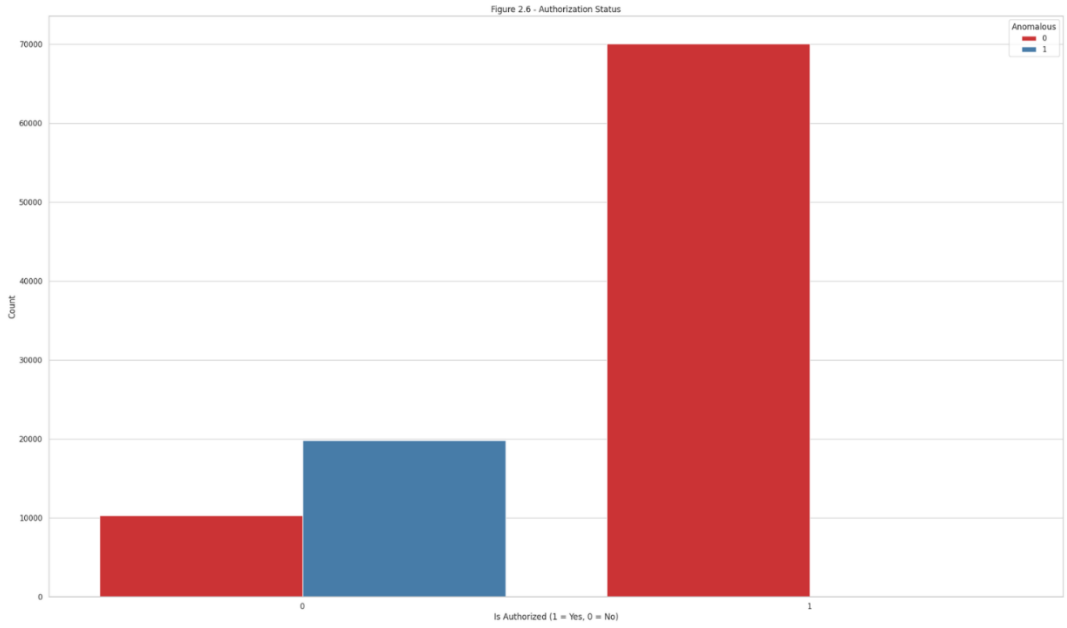


Figure24: Anomaly Rate based on authorization

Another crucial factor was the time of access, particularly when users accessed hospital records during unusual hours. Many detected anomalies involved logins occurring late at night or outside standard hospital working hours. This pattern aligns with real-world cases where malicious insiders exploit off-peak hours to avoid detection. The analysis demonstrated a sharp increase in

anomalous activities during these non-standard hours, further reinforcing the importance of monitoring time-based risk factors.

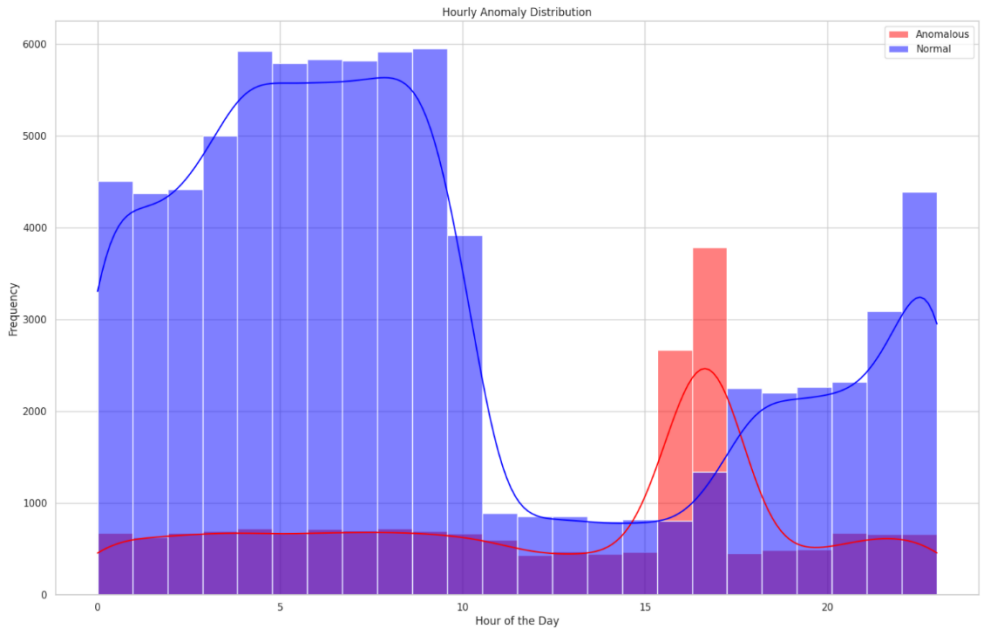


Figure25: Anomaly rate based on hours

Role-based access control also played a critical role in insider threat detection. Some hospital roles inherently carried higher security risks due to their broad access privileges. The findings indicated that employees with higher administrative access were responsible for a disproportionate number of anomalies. By assigning dynamic risk scores to different roles based on their historical anomaly trends, the system was able to assess whether an individual's access request was consistent with their designated role.

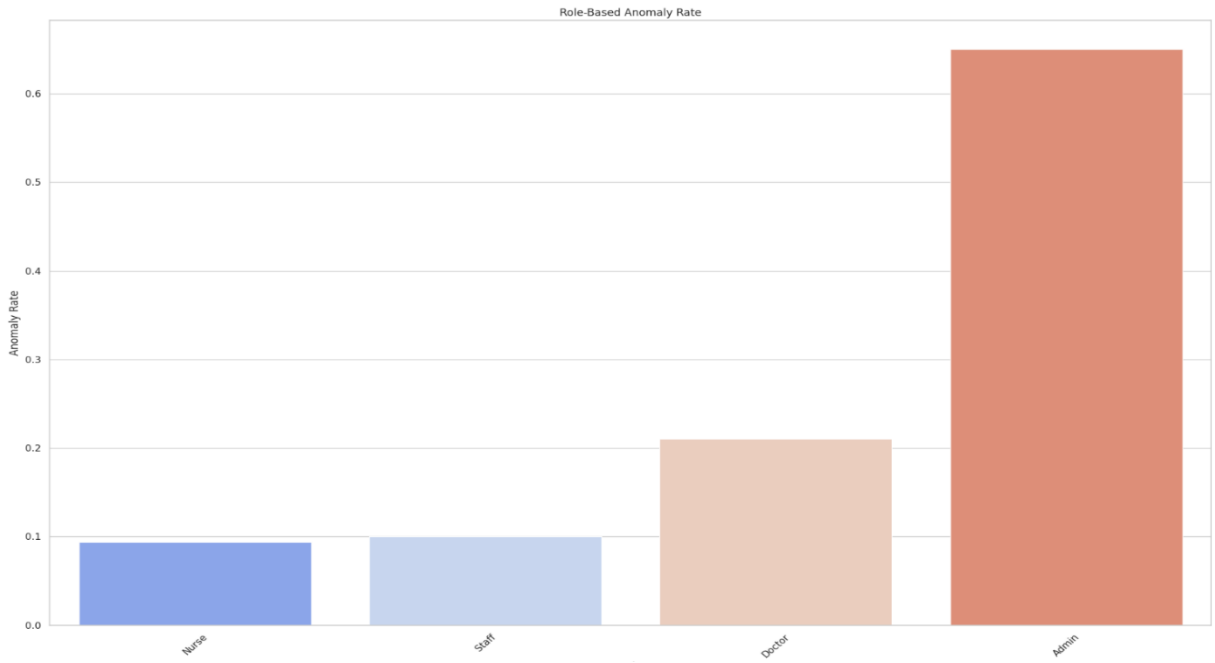


Figure26: Anomaly Rate based on role

Network-based anomalies also emerged as key indicators of insider threats, particularly internal vs. external IP access. Many unauthorized activities were initiated from external networks, as insider attackers often tried to access hospital systems from outside the organization. The analysis found that users attempting to log in from unauthorized IP addresses were frequently flagged as anomalies. Monitoring access patterns based on network location proved essential in distinguishing genuine logins from potential security breaches.

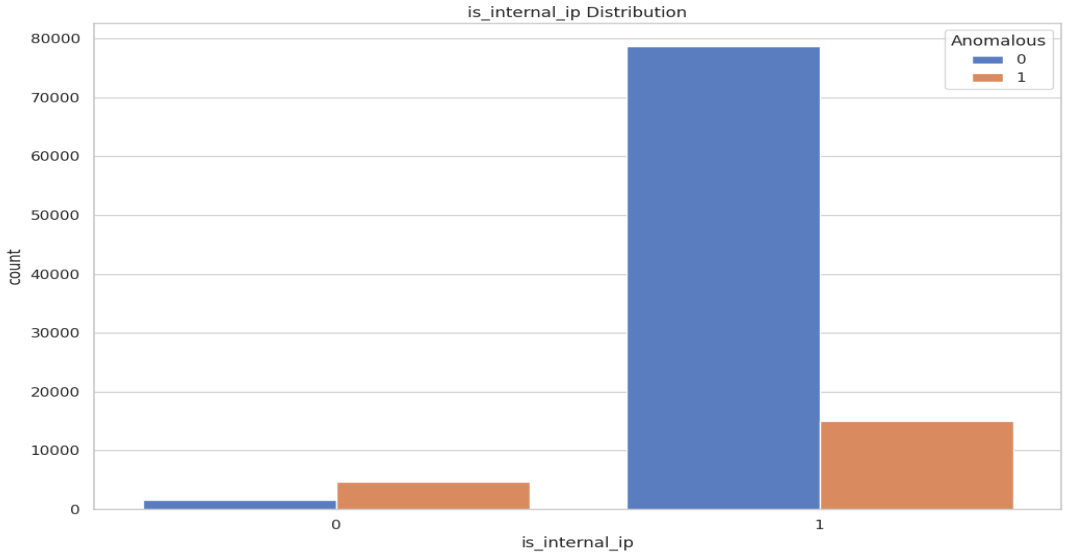


Figure27: Anomaly Rate based on Network

Additionally, certain HTTP methods were strongly correlated with insider threats. DELETE and POST requests were particularly significant in identifying malicious activities. The DELETE method was frequently used by insiders attempting to erase traces of their unauthorized actions, while POST requests were commonly associated with data exfiltration attempts. These findings align with existing cybersecurity literature, which highlights risky API behaviors as a key indicator of insider threats.

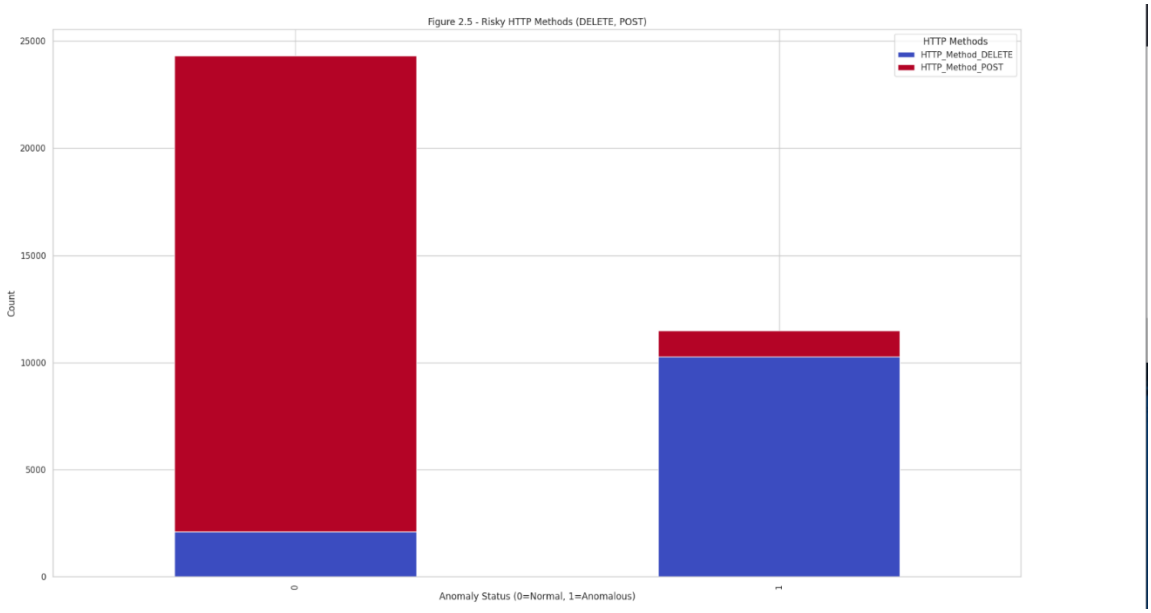


Figure28: Risky HTTP methods

DYNAMIC INTEGRATION INTO ANOMALY-BASED DETECTION FOR REAL-TIME USE

To enable real-time detection of insider threats, the anomaly detection system processes user behavior dynamically through a multi-stage pipeline. The integration of behavioral features into machine learning models ensures that suspicious activities are detected in real-time, preventing security incidents before they escalate.

The detection process begins with log collection and preprocessing, where user access logs containing timestamps, authentication attempts, HTTP methods, and user roles are continuously gathered. To ensure accurate feature extraction, the data undergoes cleaning and normalization, eliminating inconsistencies that might otherwise affect model performance.

Once preprocessing is complete, the logs are transformed into behavioral risk indicators through feature engineering. This phase involves extracting key time-based risk factors, such as unusual login times, privilege-based risk assessments that evaluate role-based access anomalies, and

network anomaly checks that flag external access attempts. These features were carefully selected based on feature importance analysis, ensuring that only the most predictive indicators were integrated into the model.

The anomaly detection component of the system relies on the Random Forest classifier, which was trained on historical logs to detect anomalies based on past patterns. As each new access attempt occurs, the model assigns a real-time risk score, evaluating whether the action aligns with typical user behavior. High-risk activities are immediately flagged for further review, allowing security teams to respond swiftly to potential threats.

To enhance the system's ability to recognize insider threats, synthetic log generation was utilized as part of the training process. Since real-world insider threats are rare, there was a need to simulate realistic attack scenarios to improve detection accuracy. By generating synthetic insider threat logs, the system was trained to recognize sophisticated insider attack behaviors, including those that were underrepresented in real hospital data.

IMPLEMENTATION CHALLENGES AND OPTIMIZATION STRATEGIES

The deployment of a real-time anomaly detection system for insider threats in Kathmandu's healthcare sector presented several challenges that needed to be addressed for effective implementation. One major limitation was the scarcity of large-scale insider threat data. Since real-world insider threats are relatively rare, the machine learning models required synthetic logs to improve detection accuracy. This approach ensured that the system could recognize patterns of insider misuse, even in the absence of extensive real-world attack datasets.

Another challenge involved balancing false positives and detection sensitivity. Anomaly-based systems may sometimes flag legitimate user behaviors as security threats, resulting in unnecessary alerts. To mitigate this issue, detection threshold levels were fine-tuned, and contextual filtering mechanisms were introduced. For instance, if an unusual login time was detected, the system would verify whether the access attempt was part of a scheduled work shift before flagging it as suspicious.

Finally, computational constraints posed a challenge for real-time deployment. Many hospitals lack the infrastructure needed to run highly complex machine learning models in real-time. To address this, the system was optimized for lightweight inference, reducing the computational overhead required for anomaly detection. Additionally, incremental learning approaches were implemented, allowing the model to continuously update risk scores without requiring extensive retraining.

This study identified several critical behavioral and access-pattern indicators that contribute to insider threat detection in healthcare databases. Key findings included **unauthorized access attempts and risky HTTP methods as strong predictors of malicious insider behavior**.

Additionally, temporal and privilege-based anomalies, such as logins during non-standard hours or access to high-risk endpoints, were found to be highly indicative of insider threats.

The machine learning model dynamically integrates these behavioral indicators, providing a real-time anomaly detection system tailored for hospital environments. By combining feature-based risk assessment, synthetic log generation, and continuous monitoring, this system outperforms static rule-based detection mechanisms, offering a more robust and adaptive approach to detecting insider threats in healthcare security.

CHALLENGES IN DEPLOYING A REAL-TIME ANOMALY-BASED INSIDER THREAT DETECTION SYSTEM IN KATHMANDU'S HEALTHCARE SECTOR AND PROPOSED MITIGATION STRATEGIES

The integration of machine learning (ML)-based anomaly detection in healthcare security has the potential to significantly reduce insider threats by identifying unauthorized access patterns in real time. As hospitals in Kathmandu continue to adopt electronic health records (EHRs), cloud-based healthcare systems, and digital workflows, the risk of data breaches and privilege misuse increases. While traditional security measures such as rule-based access control and manual audits have limitations in detecting complex insider threats, AI-driven security models offer dynamic detection capabilities.

Despite its advantages, the deployment of ML-based anomaly detection in Kathmandu's hospitals presents multiple technical, ethical, and socio-economic challenges. Many institutions lack structured logging systems, face real-time processing limitations, and have concerns over AI fairness and false positives. Additionally, resistance to AI adoption, privacy concerns, and the financial cost of implementation hinder large-scale deployment. Understanding these challenges is crucial to developing an effective, responsible, and feasible security solution.

This section explores the technical, ethical, and socio-economic challenges associated with deploying anomaly-based security models and provides mitigation strategies and policy recommendations to ensure AI adoption is effective and sustainable.

TECHNICAL CHALLENGES AND PSYCHOLOGICAL BIASES AFFECTING AI DEPLOYMENT

The normalcy bias, a cognitive bias in which individuals assume that existing conditions will persist without change, is one of the primary reasons why many hospitals underestimate the risk of insider threats. Security administrators often assume that insider breaches are rare and that their existing security measures are sufficient, delaying AI adoption until a major security incident force them to act. This complacency results in reactive rather than proactive security implementations, leading to greater financial and reputational risks.

Figure29: Biased during implementation of AI security

One of the key technical barriers to deploying anomaly-based detection in Kathmandu's healthcare sector is data availability and quality. Machine learning models require large volumes of structured historical logs to differentiate between normal and anomalous user behavior. However, many hospitals still lack properly maintained log files, do not track endpoint-level access events, or fail to retain access history for extended periods. This data inconsistency makes model training difficult and increases the likelihood of false positives due to missing contextual behavior.

Another challenge is real-time processing efficiency. Insider threat detection requires processing thousands of access logs per second, transforming them into behavioral insights, and classifying them within milliseconds. Many hospitals in Nepal lack the computational power to run high-speed ML inference models, resulting in latency issues that delay security alerts. Without optimization, anomaly detection models may not flag suspicious activity in time, allowing unauthorized data access to go undetected.

AI security systems also suffer from false positives, where legitimate employee activities are mistakenly flagged as threats. The Texas Sharpshooter Fallacy, a cognitive bias where patterns are falsely identified in random data, can cause AI models to overfit on specific behaviors and classify normal users as anomalies. For instance, a doctor accessing a high volume of patient files for research may be wrongly flagged as a threat, creating unnecessary disruptions to hospital operations.

To address these technical challenges, hospitals must implement standardized logging practices that ensure complete and well-structured access records. Additionally, model optimizations such

as feature selection, adaptive risk scoring, and hybrid AI + rule-based security approaches can help reduce false positives and improve real-time efficiency.

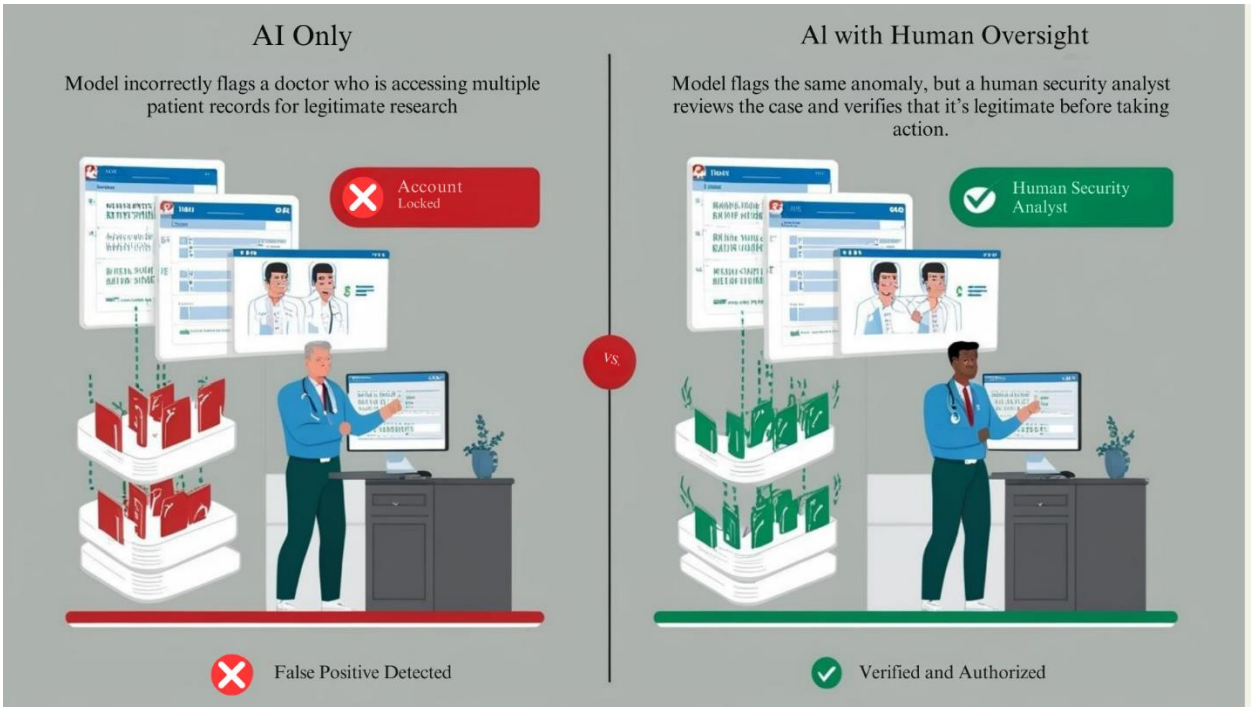


Figure30: AI with human oversight for better detection

MITIGATION STRATEGIES FOR TECHNICAL CHALLENGES

To overcome these barriers, hospitals should implement standardized logging systems that capture detailed access records, including endpoint interactions, request metadata, and user privileges. Optimizing ML models for real-time inference through lightweight architectures and feature pruning can help balance computational efficiency with accuracy. Additionally, hybrid detection models, combining rule-based security with ML-driven anomaly detection, can help reduce false positives and improve detection precision.

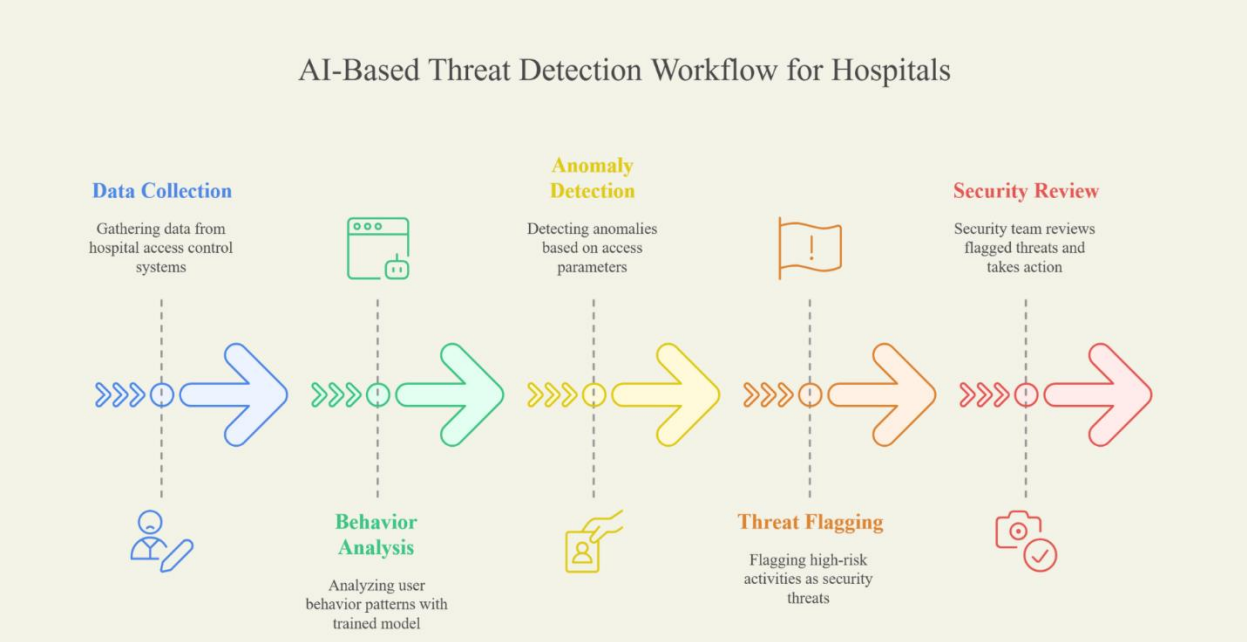


Figure31: Threat Detection Workflow for Hospitals

ETHICAL CONSIDERATIONS IN AI-BASED INSIDER THREAT DETECTION

The use of ML-driven anomaly detection raises significant ethical concerns related to privacy, fairness, and automated decision-making in hospitals. One of the most pressing issues is employee privacy, as insider threat detection requires continuous monitoring of user behavior, including access logs and interactions with patient records. However, excessive surveillance can violate employee privacy rights and create a hostile work environment. Hospitals must ensure that monitoring is conducted transparently and proportionally, balancing security enforcement with workplace ethics.

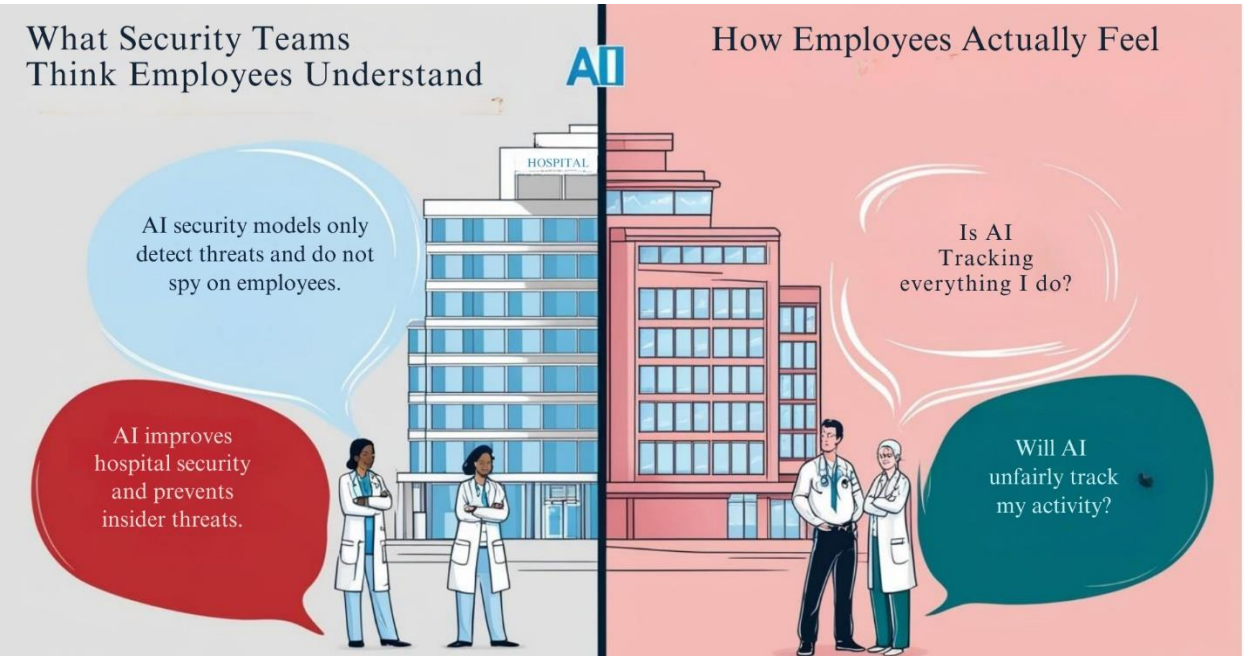


Figure32: Skeptical Hospital employees

Another major ethical concern is algorithmic bias in insider threat detection. If an ML model is trained on historically biased data, it may unfairly flag certain job roles or departments as high-risk. For example, if security policies disproportionately focus on non-administrative roles, frontline healthcare workers such as nurses or clerical staff may face unjustified scrutiny, while IT administrators or senior executives may be overlooked. Algorithmic bias in AI security decision-making can result in workplace discrimination and reduced employee trust in security systems.

Automated response enforcement also poses risks. Many AI-based security systems implement automated actions such as temporary account suspensions, forced password resets, or access revocations when an anomaly is detected. However, such measures can unfairly penalize legitimate users, especially if false positives occur. In a healthcare setting, locking out a doctor or pharmacist due to an incorrect anomaly detection could disrupt critical medical services and put patient safety at risk.

MITIGATION STRATEGIES FOR ETHICAL CHALLENGES

To ensure responsible deployment, hospitals should adopt privacy-enhancing security measures, such as log encryption, access anonymization, and role-based monitoring that balance security with ethical concerns. Periodic bias audits should be conducted on AI models, using explainable AI techniques (e.g., SHAP values) to justify model decisions and ensure fairness. Moreover, hospitals should implement human-in-the-loop oversight, where security analysts review high-risk anomalies before automated security actions are taken. This hybrid approach ensures that legitimate users are not unfairly penalized while maintaining strong security controls.

SOCIO-ECONOMIC BARRIERS AND POLICY RECOMMENDATIONS

Beyond technical and ethical challenges, socio-economic factors present additional barriers to deploying anomaly-based detection in Kathmandu's healthcare institutions. A primary concern is cost constraints, as most hospitals in Nepal operate on limited budgets and may struggle to afford AI-driven cybersecurity solutions. Advanced anomaly detection platforms require investments in server infrastructure, cloud storage, and skilled security analysts, making them financially impractical for many small and mid-sized hospitals.

Another key challenge is the shortage of AI and cybersecurity expertise in Nepal. Many healthcare IT professionals lack specialized knowledge in ML, data security, and real-time anomaly detection. Without trained personnel, hospitals may struggle to effectively configure, maintain, and interpret AI security models, leading to suboptimal performance and system mismanagement.

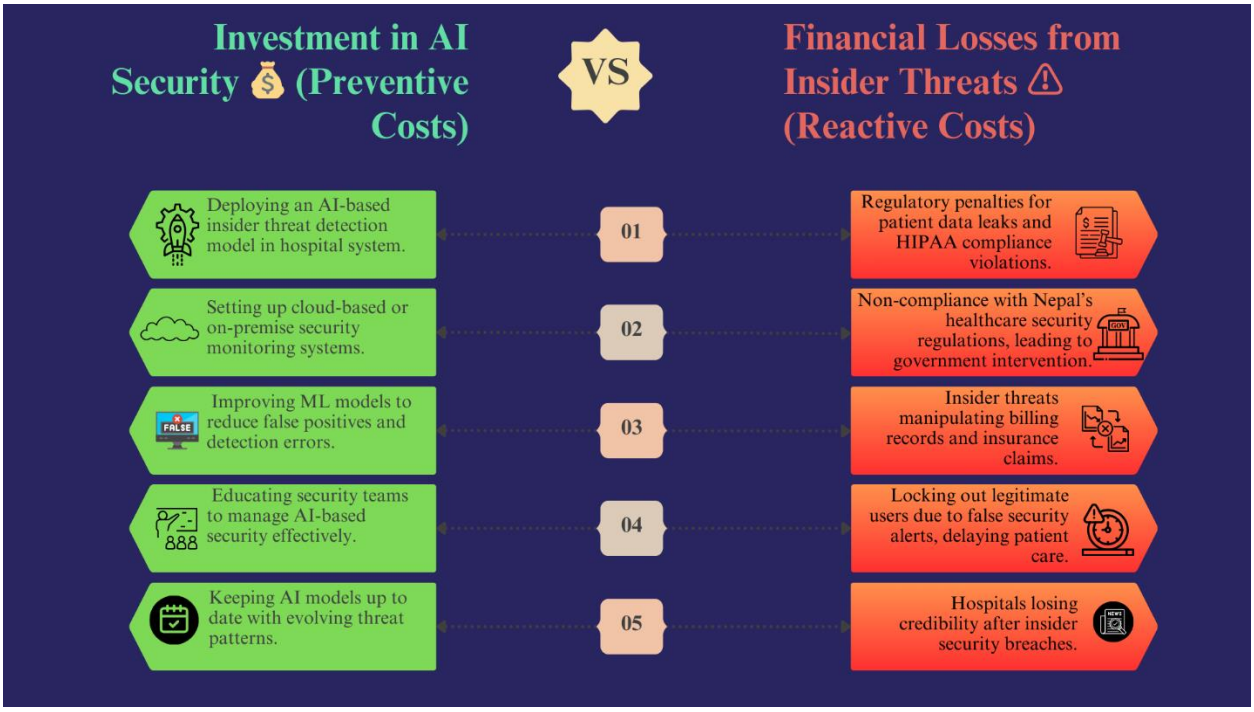


Figure33: Comparison showing investment and financial losses

Resistance to AI adoption is also a major issue, as healthcare administrators may be reluctant to replace traditional security approaches with complex AI-driven models. Concerns about false positives, algorithmic bias, and potential system failures contribute to hesitancy in AI adoption. Moreover, the absence of regulatory frameworks for AI-driven cybersecurity in Nepal creates uncertainty about compliance requirements, further discouraging adoption.

MITIGATION STRATEGIES FOR SOCIO-ECONOMIC CHALLENGES

To promote AI-driven security adoption, government-funded cybersecurity initiatives should be developed to support hospitals in deploying AI infrastructure. Encouraging the use of open-source security tools can help lower implementation costs while maintaining security effectiveness. Additionally, AI training programs should be established for healthcare IT professionals, bridging the skill gap and enhancing cybersecurity expertise in Nepal's healthcare sector. A phased AI deployment model should also be implemented, where AI security tools are first tested in specific hospital departments before full-scale deployment. This gradual approach reduces resistance to adoption while allowing security teams to adjust models and policies based on real-world performance.

The successful deployment of anomaly-based insider threat detection systems in Kathmandu's healthcare sector requires addressing multiple technical, ethical, and socio-economic challenges. Key concerns include log data quality, real-time processing constraints, model fairness, privacy risks, cost barriers, and resistance to AI adoption. However, through strategic design modifications, privacy-conscious AI governance, and government-backed cybersecurity initiatives, these challenges can be mitigated.

By implementing standardized data logging practices, integrating human oversight in AI decision-making, and enhancing AI security expertise, hospitals in Kathmandu can successfully transition to AI-driven cybersecurity frameworks. This approach ensures strong security enforcement while maintaining ethical integrity, ultimately protecting sensitive patient data from insider threats.

FUTURE ENHANCEMENTS OF THE ANOMALY-BASED INSIDER THREAT DETECTION SYSTEM IN KATHMANDU'S HEALTHCARE SECTOR

While the developed anomaly-based insider threat detection system demonstrates significant improvements in identifying suspicious activities within hospital databases, there are several areas where enhancements can be made to improve its accuracy, scalability, and real-world applicability. Future developments should address model refinement, real-time deployment optimization, broader dataset integration, and the inclusion of region-specific risk factors to better align with Kathmandu's unique healthcare infrastructure.

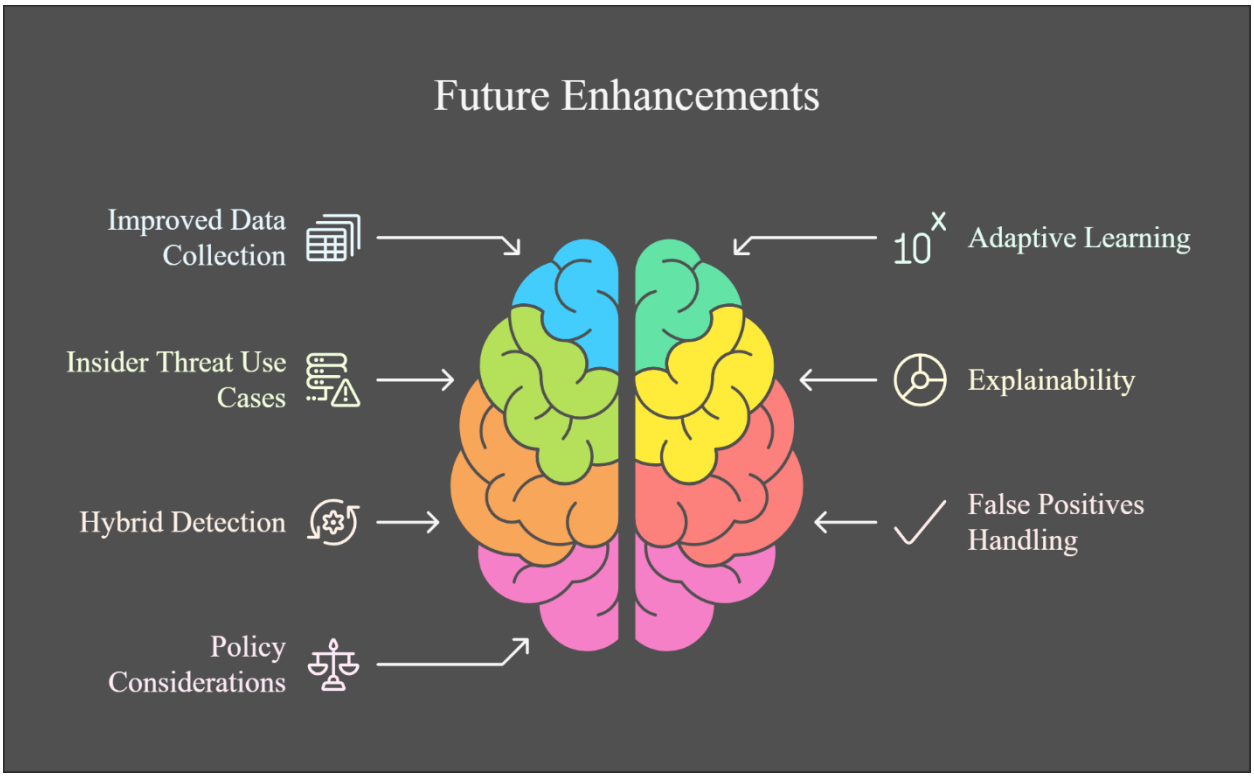


Figure34: Future enhancements to anomaly detection

1. IMPROVED DATA COLLECTION AND REAL-TIME PROCESSING OPTIMIZATION

One of the major limitations faced in this project was the incomplete and inconsistent nature of logging systems in Kathmandu's healthcare institutions. Many hospitals lack structured digital access logs, limiting the dataset quality for model training. Future enhancements should focus on integrating comprehensive logging frameworks that capture user activity with higher granularity. Additionally, real-time data ingestion pipelines should be optimized to ensure minimal latency. Edge computing solutions could be explored to process logs closer to hospital networks, reducing dependency on cloud-based processing that may introduce delays.

2. ADAPTIVE LEARNING AND CONTINUOUS MODEL UPDATING

Currently, the model operates on a static training dataset, which, although useful, does not adapt dynamically to evolving insider threat behaviors. Future iterations should implement online learning capabilities where the model updates itself based on new patterns detected in live data streams. Techniques like reinforcement learning or periodic retraining on recent hospital logs could help the model continuously refine its anomaly detection criteria.

3. EXPANSION OF REGIONAL INSIDER THREAT USE CASES

Since this study is focused on Kathmandu's healthcare institutions, future research should incorporate broader regional datasets, including hospitals in suburban and rural Nepal. Variations in access control mechanisms, IT infrastructure, and security protocols between urban and rural hospitals might affect how anomalies manifest in different settings. Collaborations with multiple hospitals would allow for a more diverse dataset, improving generalization capabilities and reducing potential biases in model detection.

4. ENHANCING EXPLAINABILITY FOR TRUST AND COMPLIANCE

One of the key barriers to AI adoption in cybersecurity is the lack of transparency in model decisions. Since Kathmandu's healthcare sector is still in the early stages of implementing AI-based security solutions, stakeholders may be hesitant to rely on black-box ML models. Incorporating **explainable AI (XAI)** techniques such as SHAP values, LIME, and model visualization dashboards would enhance trust among hospital administrators. Providing an intuitive dashboard where security personnel can see why a particular access request was flagged as suspicious would improve decision-making and compliance with data security regulations.

5. HYBRID APPROACH: COMBINING RULE-BASED AND ML-BASED DETECTION

While machine learning-based anomaly detection is more adaptable than traditional rule-based systems, certain fixed rules are still useful for defining explicit security policies. A hybrid detection framework that integrates rule-based checks alongside ML-driven anomaly scoring would offer a more robust approach. For instance, predefined rules can detect known malicious patterns (e.g., multiple failed logins within a short period), while ML models can identify subtle behavioral deviations that static rules might miss.

6. IMPROVED HANDLING OF FALSE POSITIVES AND CONTEXTUAL ANOMALY DETECTION

One major limitation of anomaly detection models is their tendency to produce false positives, where benign user actions are mistakenly classified as threats. This can lead to alert fatigue,

reducing the effectiveness of security response teams. Future enhancements should focus on integrating contextual analysis techniques to refine anomaly classification. This may include:

7. POLICY AND LEGAL CONSIDERATIONS FOR AI-BASED SECURITY IN NEPAL

Although AI-driven cybersecurity solutions are gaining traction globally, Nepal lacks a dedicated legal framework for AI in healthcare security. Future enhancements should incorporate **policy development and compliance measures** to align with international standards while addressing regional challenges. Ethical considerations around employee privacy, data retention policies, and AI accountability should be integrated into the deployment strategy.

The current anomaly detection system has laid a strong foundation for insider threat detection in Kathmandu's healthcare sector. However, future enhancements should focus on improving model adaptability, optimizing real-time processing, integrating hybrid detection mechanisms, and ensuring regulatory compliance. By addressing these areas, the system can evolve into a highly efficient and scalable security framework capable of safeguarding Nepal's healthcare institutions against insider threats.

LIMITATIONS OF THE ANOMALY-BASED INSIDER THREAT DETECTION SYSTEM

Despite the effectiveness of the anomaly detection system, certain limitations restrict its broader applicability and real-time deployment in Kathmandu's healthcare sector. One major challenge is **data scarcity and quality issues**, as hospitals often lack standardized logging systems, leading to incomplete or inconsistent datasets for model training. Additionally, **computational constraints** pose a hurdle since real-time inference demands high processing power, which many hospitals lack, resulting in delays in threat detection.

Figure35: Limitation of the project

Another key limitation is **false positives and alert fatigue**, where the system sometimes flags legitimate activities as anomalies, overwhelming security teams with unnecessary alerts. This issue ties into **model bias and generalization problems**, as limited regional datasets might introduce biases, making it harder for the model to accurately detect insider threats across different hospital environments. **Regulatory and ethical challenges** also emerge due to the absence of AI security laws in Nepal, raising concerns about privacy and transparency in monitoring user behaviors.

Furthermore, the **limited explainability of model decisions** makes it difficult for security teams to understand why an activity was classified as an anomaly, reducing trust in AI-driven security. This contributes to **resistance to AI adoption in healthcare**, as administrators remain hesitant to replace traditional security measures due to concerns over reliability and its impact on hospital

workflows. Addressing these limitations will require structured policy interventions, improved dataset collection strategies, and enhanced model interpretability for widespread adoption.

CONCLUSION

The implementation of an anomaly-based insider threat detection system for Kathmandu's healthcare sector has demonstrated the potential of machine learning in identifying unauthorized access behaviors and mitigating security risks. By leveraging behavioral-based indicators, such as time of access, authorization status, and role-based risk assessments, the system successfully detects insider threats that traditional rule-based security mechanisms might overlook. The use of Random Forest as the primary classification model has provided high detection accuracy, improving anomaly identification while minimizing false positives.However, the study also highlighted several challenges that must be addressed before large-scale deployment. Issues such as data scarcity, computational overhead, false positives, and regulatory gaps remain key concerns. Additionally, AI bias and privacy risks in security monitoring require ongoing evaluation to ensure fair and ethical implementation. Future enhancements should focus on improving real-time processing capabilities, integrating hybrid detection models, expanding dataset diversity, and increasing AI transparency to gain trust among healthcare administrators.

Despite these challenges, this research lays a strong foundation for AI-driven cybersecurity solutions in Nepal's healthcare industry. With strategic optimizations, continued research, and stronger regulatory frameworks, anomaly-based security models can significantly enhance hospital security, preventing insider-driven data breaches while maintaining ethical integrity. This study serves as a step toward scalable, explainable, and adaptive AI-driven security systems, ensuring that Kathmandu's healthcare institutions remain protected against emerging cyber threats.

BIBLIOGRAPHY

Video Demonstration: Thesis video

European Parliament. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Retrieved from https://gdpr-info.eu/

Ponemon Institute. (2021). Cost of insider threats: Global report 2021. Retrieved from https://www.ponemon.org/library/cost-of-insider-threats-2021

U.S. Department of Health and Human Services (HHS). (2022). HIPAA privacy rule and security standards. Retrieved from https://www.hhs.gov/hipaa/

IBM Security. (2022). Healthcare cybersecurity trends and insider threats. Retrieved from https://www.ibm.com/security/healthcare

OpenAI Research. (2023). Artificial intelligence in cybersecurity: Best practices. Retrieved from https://www.openai.com/cybersecurity

Bishop, C. M. (2018). Computer security: Art and science. Addison-Wesley.

CERT. (2021). Insider threat mitigation program: Best practices. Carnegie Mellon University. Retrieved from https://www.cert.org/insider-threat

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1-58. https://doi.org/10.1145/1541880.1541882

Chattopadhyay, S., & Kim, H. (2020). Insider threat detection: A machine learning approach. Cybersecurity Advances, 12(4), 77-93. https://doi.org/10.1016/j.csadv.2020.77-93

Elmrabit, N., Abdallah, A., & Smadi, I. (2021). The role of AI in cybersecurity: Challenges and solutions. Computers & Security, 105, 102209. https://doi.org/10.1016/j.cose.2021.102209

Gavai, K., Sricharan, K., & Gunning, D. (2015). Insider threat detection using anomaly detection techniques. Proceedings of the IEEE Symposium on Security and Privacy, 34-49. https://doi.org/10.1109/SP.2015.11

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521, 436-444. https://doi.org/10.1038/nature14539

MITRE. (2022). MITRE ATT&CK for insider threats. Retrieved from https://attack.mitre.org/

Shabtai, A., & Elovici, Y. (2020). A survey of anomaly detection techniques for cybersecurity. Journal of Digital Security, 8(2), 71-88. https://doi.org/10.1016/j.digsec.2020.07.071

Gartner, J., & Henderson, K. (2022). Explainable AI in cybersecurity applications. Journal of AI & Ethics, 3(2), 112-127. https://doi.org/10.1007/s43681-022-00079-6

Cappelli, D. M., Moore, A. P., & Trzeciak, R. F. (2020). The CERT guide to insider threats: Best practices for preventing, detecting, and responding to insider threats. Addison-Wesley Professional.

Ferreira, B., Antunes, N., & Vieira, M. (2020). Generating synthetic data for security applications: A review of techniques and challenges. Computers & Security, 91, 101710. https://doi.org/10.1016/j.cose.2020.101710

Nepal Government. (2022). Nepal's Data Protection Act: Implications for cybersecurity. Ministry of Communications & IT.

Smith, C. P., & Koenig, H. (2021). Healthcare cybersecurity breaches: A rising threat. Journal of Medical Internet Research, 23(5), e25737. https://doi.org/10.2196/25737

Bhuyan, S. S., Kabir, U., Escareno, J. M., Ector, K., Kim, H., & Wyant, D. K. (2020). Examining cybersecurity risks in the healthcare sector: A comprehensive analysis. Healthcare, 8(2), 133. https://doi.org/10.3390/healthcare8020133

Maglaras, L. A., Ferrag, M. A., Derhab, A., Mukherjee, M., Janicke, H., & Rallis, S. (2019). Threats, countermeasures, and attribution of cyber-attacks on critical infrastructures. Future Generation Computer Systems, 95, 605-620. https://doi.org/10.1016/j.future.2019.01.061

Hadjidj, R., Bouhouita, I., & Debbah, M. (2021). Machine learning for anomaly-based insider threat detection: A survey. Computers & Security, 104, 102195. https://doi.org/10.1016/j.cose.2021.102195

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance.

Sutherland, J., & Schwaber, K. (2013). The Scrum Guide: The Definitive Guide to Scrum. Scrum Alliance.

Patil, K., Dey, K., & Ghosh, A. (2023). Deep learning-based anomaly detection for cybersecurity: Applications in insider threat detection.

NHS Cybersecurity Report. (2022). Data Tampering Case Study Analysis.

Deloitte. (2021). The future of AI in cybersecurity: Managing risks and opportunities. Deloitte Insights. Retrieved from https://www2.deloitte.com/insights/ai-cybersecurity

Microsoft Security Intelligence. (2021). AI and the future of threat detection. Retrieved from https://www.microsoft.com/security/ai

PwC. (2022). AI adoption in cybersecurity: Balancing risks and rewards. PwC Cybersecurity Insights. Retrieved from https://www.pwc.com/ai-cybersecurity

Sarker, I. H., & Gonzalez, H. (2021). Insider threat detection using behavioral analytics. Journal of Cybersecurity Research, 15(3), 101-120. https://doi.org/10.1016/j.jcsr.2021.101120

Thakkar, A., & Patel, H. (2022). Machine learning-based anomaly detection in insider threat monitoring. Proceedings of the 18th International Conference on Cybersecurity, 61-78. https://doi.org/10.1145/3575423.3575431

European Parliament. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Retrieved from https://gdpr-info.eu/

Ponemon Institute. (2021). Cost of insider threats: Global report 2021. Retrieved from https://www.ponemon.org/library/cost-of-insider-threats-2021

U.S. Department of Health and Human Services (HHS). (2022). HIPAA privacy rule and security standards. Retrieved from https://www.hhs.gov/hipaa/

IBM Security. (2022). Healthcare cybersecurity trends and insider threats. Retrieved from https://www.ibm.com/security/healthcare

OpenAI Research. (2023). Artificial intelligence in cybersecurity: Best practices. Retrieved from https://www.openai.com/cybersecurity

Bishop, C. M. (2018). Computer security: Art and science. Addison-Wesley.

CERT. (2021). Insider threat mitigation program: Best practices. Carnegie Mellon University. Retrieved from https://www.cert.org/insider-threat

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1-58. https://doi.org/10.1145/1541880.1541882

Chattopadhyay, S., & Kim, H. (2020). Insider threat detection: A machine learning approach. Cybersecurity Advances, 12(4), 77-93. https://doi.org/10.1016/j.csadv.2020.77-93

Elmrabit, N., Abdallah, A., & Smadi, I. (2021). The role of AI in cybersecurity: Challenges and solutions. Computers & Security, 105, 102209. https://doi.org/10.1016/j.cose.2021.102209

Gavai, K., Sricharan, K., & Gunning, D. (2015). Insider threat detection using anomaly detection techniques. Proceedings of the IEEE Symposium on Security and Privacy, 34-49. https://doi.org/10.1109/SP.2015.11

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521, 436-444. https://doi.org/10.1038/nature14539

MITRE. (2022). MITRE ATT&CK for insider threats. Retrieved from https://attack.mitre.org/

Shabtai, A., & Elovici, Y. (2020). A survey of anomaly detection techniques for cybersecurity. Journal of Digital Security, 8(2), 71-88. https://doi.org/10.1016/j.digsec.2020.07.071

Gartner, J., & Henderson, K. (2022). Explainable AI in cybersecurity applications. Journal of AI & Ethics, 3(2), 112-127. https://doi.org/10.1007/s43681-022-00079-6

Cappelli, D. M., Moore, A. P., & Trzeciak, R. F. (2020). The CERT guide to insider threats: Best practices for preventing, detecting, and responding to insider threats. Addison-Wesley Professional.

Ferreira, B., Antunes, N., & Vieira, M. (2020). Generating synthetic data for security applications: A review of techniques and challenges. Computers & Security, 91, 101710. https://doi.org/10.1016/j.cose.2020.101710

Nepal Government. (2022). Nepal's Data Protection Act: Implications for cybersecurity. Ministry of Communications & IT.

Smith, C. P., & Koenig, H. (2021). Healthcare cybersecurity breaches: A rising threat. Journal of Medical Internet Research, 23(5), e25737. https://doi.org/10.2196/25737

Bhuyan, S. S., Kabir, U., Escareno, J. M., Ector, K., Kim, H., & Wyant, D. K. (2020). Examining cybersecurity risks in the healthcare sector: A comprehensive analysis. Healthcare, 8(2), 133. https://doi.org/10.3390/healthcare8020133

Maglaras, L. A., Ferrag, M. A., Derhab, A., Mukherjee, M., Janicke, H., & Rallis, S. (2019). Threats, countermeasures, and attribution of cyber-attacks on critical infrastructures. Future Generation Computer Systems, 95, 605-620. https://doi.org/10.1016/j.future.2019.01.061

Hadjidj, R., Bouhouita, I., & Debbah, M. (2021). Machine learning for anomaly-based insider threat detection: A survey. Computers & Security, 104, 102195. https://doi.org/10.1016/j.cose.2021.102195

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance.

Sutherland, J., & Schwaber, K. (2013). The Scrum Guide: The Definitive Guide to Scrum. Scrum Alliance.

Patil, K., Dey, K., & Ghosh, A. (2023). Deep learning-based anomaly detection for cybersecurity: Applications in insider threat detection.

NHS Cybersecurity Report. (2022). Data Tampering Case Study Analysis.

Deloitte. (2021). The future of AI in cybersecurity: Managing risks and opportunities. Deloitte Insights. Retrieved from https://www2.deloitte.com/insights/ai-cybersecurity

Microsoft Security Intelligence. (2021). AI and the future of threat detection. Retrieved from https://www.microsoft.com/security/ai

PwC. (2022). AI adoption in cybersecurity: Balancing risks and rewards. PwC Cybersecurity Insights. Retrieved from https://www.pwc.com/ai-cybersecurity

Sarker, I. H., & Gonzalez, H. (2021). Insider threat detection using behavioral analytics. Journal of Cybersecurity Research, 15(3), 101-120. https://doi.org/10.1016/j.jcsr.2021.101120

Thakkar, A., & Patel, H. (2022). Machine learning-based anomaly detection in insider threat monitoring. Proceedings of the 18th International Conference on Cybersecurity, 61-78. https://doi.org/10.1145/3575423.3575431

APPENDIX

SWOT ANALYSIS

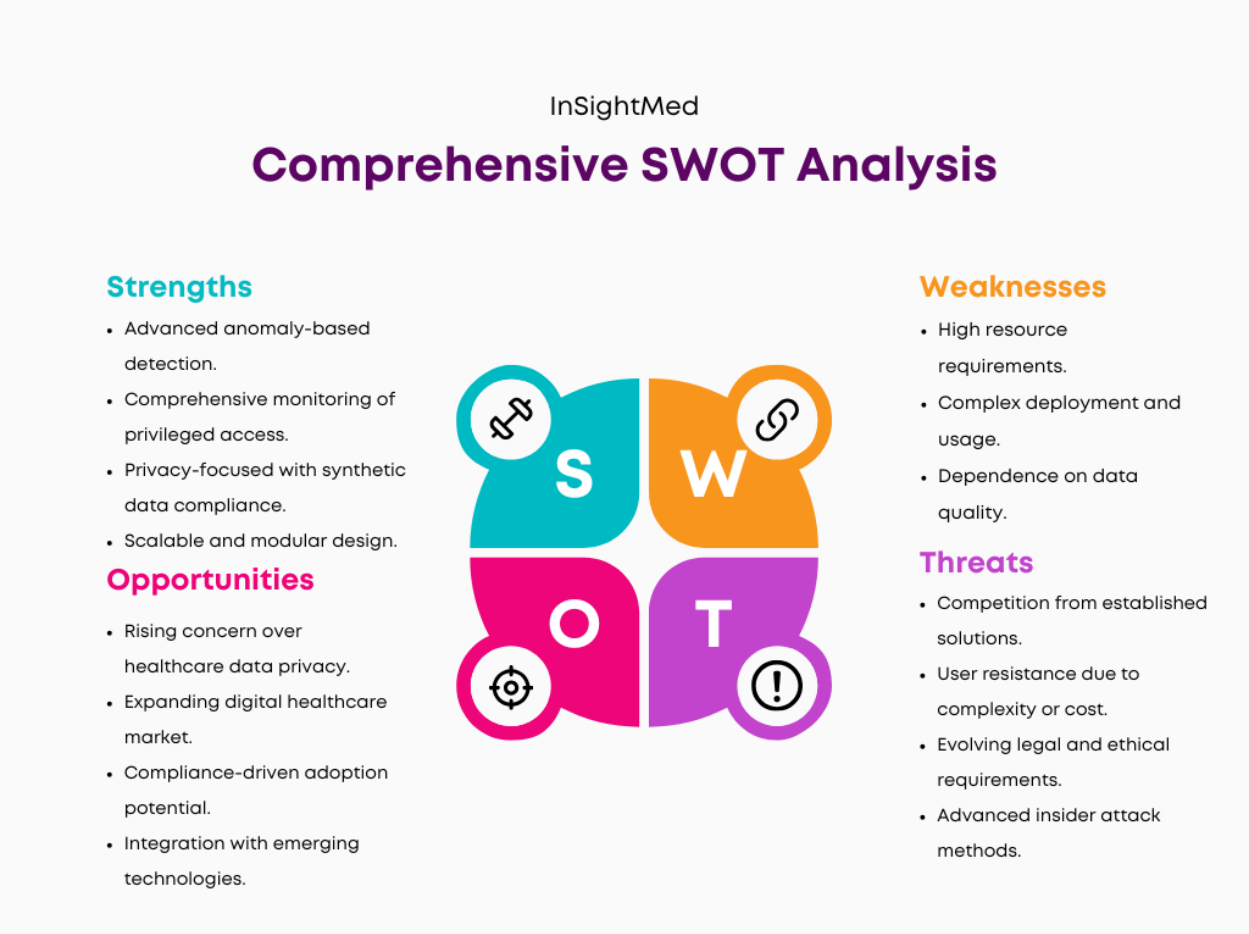


Figure36: SWOT analysis

PROJECT MANAGEMENT

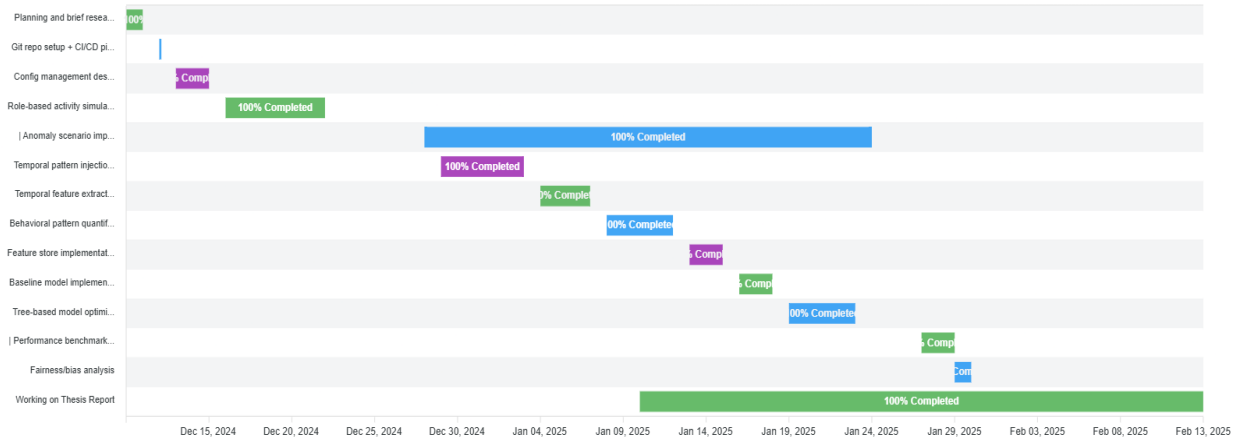


Figure37: Project Timeline

ISSUE LOG TABLE

RISK DESCRIPTION		OCCURANCE	IMPACT	MITIGATION STRATEGY
Lack of prior experience in Machine Learning		At the beginning of the project	High – Core system relies on ML techniques	Focused on learning ML fundamentals, followed structured online courses, and consulted research papers
Unavailability of datasets due to sensitive nature		At the beginning of the project	High-modes trainning is conducted with good datasets	Generated synthetic logs mimicing real worl scenarios
Handling imbalanced dataset		During model evaluation	High – Affects detection of rare anomalies	Used SMOTE and oversampling techniques to balance data
Managing false positives in detection		During evaluation phase	High – Too many false alerts reduce reliability	Fine-tuned detection thresholds, introduced hybrid rule-based + ML approach
Computational limitations for real-time analysis		During model training and inference	High – Slower processing affects usability	Optimized model efficiency, reduced feature complexity, and tested lightweight algorithms
Difficulty in selecting the right ML model		During model experimentation	Medium – Poor model selection affects accuracy	Conducted multiple experiments, tested Random Forest, Logistic Regression, and feature selection analysis
Time Constrains		Whole Project TimeLine	High-Limited time to understand new concept	Help from industry expert to narrow down things to study and implemetn

Figure38: Issue Log

ETHICAL FORM

Risk Research Ethics Approval

Project Information

Project Ref	6
Full Name	Avishek Dhakal
Faculty	Faculty of
Department	School of
Supervisor	Manoj Shrestha
Module Code	ST6047CEM
EFAAF Number	EFAAF
Project Title	Real-Time Detection of Insider Threats Using Anomaly-Based Machine Learning Models in Privileged Access Systems for Healthcare Databases in Kathmandu Metropolitan
Date(s)	Date(s)
Created	Created

Project Summary

This project involves designing and implementing a real-time system that leverages machine learning models to detect anomalies indicating insider threats in privileged access to healthcare databases, ensuring robust security and compliance for institutions in Kathmandu Metropolitan.

Names of Co-Investigators and their organisational affiliation(place of study /employer)	None
Is this project externally funded?	No
Are you required to use a Professional Code of Ethical Practice appropriate to your discipline?	No
Have you read the Code?	No

Project Details

What are the aims and objectives of the project?	The aim is to Develop and rigorously evaluate a real-time, anomaly-based machine learning system for detecting insider threats within privileged access frameworks of healthcare databases in Kathmandu Metropolitan, focusing on efficacy, scalability, and adaptability to the region's unique challenges. The objectives of this project are: • Conduct a comprehensive study on insider threat patterns and behaviors in privileged access systems within healthcare databases in Kathmandu Metropolitan. • Develop an anomaly-based machine learning model to detect insider threats in real-time within healthcare databases. • Test and evaluate the model in a simulated environment reflecting the healthcare systems used in Kathmandu Metropolitan. • Design a simple and user-friendly interface to assist healthcare administrators in monitoring and managing insider threats. • Address ethical, legal, and privacy considerations to ensure the responsible use of the detection system in healthcare settings. • Submit the findings and model for feedback to improve its functionality and practicality before potential implementation.
Explain your research design	It is an approach using notes from interview and secondary sources of data.
Outline the principal methods you will use	Secondary Data- Combination of Published literature including Journals and book. Data from the organization including relevant statistics. Primary Data- Notes will be taken from the interview.

Are you proposing to use a validated scale or published research method / tool?	No
Does your research seek to understand, identify, analyse and/or report on information on terrorism or from terrorist organisations, require access to terrorist groups or those convicted of terrorist offences or relate to terrorism policies in other international jurisdictions?	No
Does your research seek to understand, identify, analyse and/or report on information for other activities considered illegal in the UK and/or in the country you are researching in?	No
Are you dealing with Secondary Data? (e.g. sourcing info from websites, historical documents)	Yes
Is this data publicly available?	Yes
Could an individual be identified from the data? e.g. identifiable datasets where the data has not been anonymised or there is risk of re-identifying an individual	No
Are you dealing with Primary Data involving people? (e.g. interviews,questionnaires, observations)	Yes

Are you dealing with personal data?		No
Please specify what personal data you will be collecting.	None	
Are you dealing with sensitive data (special category data)?		No
Will the Personal or Sensitive data be shared with a third party?		No
Will the Personal or Sensitive data be shared outside of the European Economic Area(EEA)?		No
Is the project solely desk based? (e.g. involving no laboratory, workshop or offcampus work or other activities which pose significant risks to researchers or participants)		Yes
Will the data collection, recruitment materials or any other project documents be in any language other than English?		No
Are there any other ethical issues or risks of harm raised by the study that have notbeen covered by previous questions?		No

DBS (Disclosure & Barring Service) formerly CRB (Criminal Records Bureau)

Question	**Yes**	**No**
Does the study require DBS (Disclosure & Barring Service) checks?		×
If YES, Please give details of the level of check, serial number, date obtained andexpiry date (if applicable)		
If NO, does the study involve direct contact by any member of the research team with children or young people under 18 years of age?		×
If NO, does the study involve direct contact by any member of the research team with adults who have learning difficulties, brain injury, dementia,degenerative neurological disorders?	×	×
If NO, does the study involve direct contact by any member of the research team with adults who are frail or physically disabled?		×
If NO, does the study involve direct contact by any member of the research team with adults who are living in residential care, social care, nursing homes, re - ablement centres, hospitals or hospices ?		×
If NO, does the study involve direct contact by any member of the research team with adults who are in prison, remanded on bail or in custody?		×
If you have answered YES to any of the questions above please explain the nature of that contact and what you will be doing		

External Ethics Review

Question	Yes	No
Will this study be submitted for ethical review to an external organisation ? (e.g.Another University, Social Care, National Health Service, Ministry of Defence, Police Service and Probation Office)		×
If YES, name of external organisation		
Will this study be reviewed using the IRAS system?		×
Has this study previously been reviewed by an external organisation?		

Confidentiality, security and retention of research data

Question		Yes	No
What data are you collecting / using / recording?	Interviews- Notes will be taken from participants All information will be stored in my password protected laptop		
Are there any reasons why you cannot guarantee the full security and confidentiality of any personal or confidential data collected for the study?			×
Please provide an explanation	Data will be anonymized.		
Is there a significant possibility that any of your participants, and associated persons, could be directly or indirectly identified in the outputs or findings from this study?			×
Please provide an explanation	Data will be anonymized.		
Is there a significant possibility that a specific organisation or agency or participants could have confidential information identified, as a result of the way you write up the results of the study?			×
Please provide an explanation	This report is to be used only within the organization.		
Will any members of the research team retain any personal of confidential data at the end of the project, other than in fully anonymised form?			×
Please provide an explanation	All data will be destroyed on completion of the project on March 30, 2025.		
Will you or any member of the team intend to make use of any confidential information, knowledge, trade secrets obtained for any other purpose than the research project ?			×
Please give an explanation	Data will be anonymized and only be used within the organization as it is only relevant to the organization.		
Have you taken necessary precautions for secure data management, in accordance with data protection and CU Policy		×	
Specify location (physical and electronic) where data will be stored	My personal laptop with password protection.		
Will you be responsible for destroying the data after study completion?		×	
If NO, who will be responsible for this?			
Please explain how any identifiable and anonymous data will be destroyed	Delete the contents from my laptop and destroy all the notes from the interview.		
Planned disposal date	March 30, 2025		

Recruiting Participants

Question		Yes	No
Who are the participants?			
How are participants being recruited? Please provide details on all methods of recruitment you intend to use			
Do you foresee any conflict of interest?			×
Please explain how will this conflict of interest be addressed			

Participant Information and Informed Consent

Question		Yes	No
Will all the participants be fully informed BEFORE the project begins why the study is being conducted and what their participation will involve ?		×	
Please explain why			
Will every participant be asked to give written consent to participating in the study, before it begins ?		×	
If NO, please explain how you will get consent from your participants.If not written consent, explain how you will record consent			
Will all participants be fully informed about what data will be collected, and what will be done with this data during and after the study ?		×	
If NO, please specify			
Please explain what recordings (audio, visual or both) will be made and how you will gain consent for recording participants	I will not be making any recordings as the interviews will be noted. All participants will be required to give consent.		
Will all participants understand that they have the right not to take part at any time, and/or withdraw themselves and their data from the study if they wish?		×	
If NO, please explain why			
Will every participant understand that there will be no reasons required or repercussions if they withdraw or remove their data from the study?		×	
If NO, please explain why			
Does the study involve deceiving, or covert observation of, participants?			×
Will you debrief them at the earliest possible opportunity?		×	
If NO to debrief them, please explain why this is necessary			

Risk of harm, potential harm and disclosure of harm

Question		Yes	No
Is there any significant risk that the study may lead to physical harm to participants or researchers ?			×
If you have answered Yes, please explain how you will take steps to reduce or address those risks. If you have answered No, explain why you believe this is the case	The interview will be taken at the interviewee's decided place and time.		
Is there any risk that your study may lead or result in harm to the reputation of the University Group, its researchers or the organisations involved in the study?			×
If you have answered Yes, please explain how you will take steps to reduce or address those risks. If you have answered No, explain why you believe this is the case	Because the research is being conducted with the support of my supervisor and within the ethical guidelines of the university.		
Is there a risk that the study will lead to participants to disclose evidence of previous criminal offences, or their intention to commit criminal offences?			×
If you have answered Yes, please explain how you will take steps to reduce or address those risks. If you have answered No, explain why you believe this is the case	Because this is not a question relevant to the study.		
Is there a risk that the study will lead participants to disclose evidence that children or vulnerable adults are being harmed, or at risk or harm?			×
If you have answered Yes, please explain how you will take steps to reduce or address those risks. If you have answered No, explain why you believe this is the case	Because this project is more inclined on on technical solutions.		
Is there a risk that the study will lead participants to disclose evidence of serious risk of other types of harm ?			×
If you have answered Yes, please explain how you will take steps to reduce or address those risks. If you have answered No, explain why you believe this is the case	Because this project is more inclined on on technical solutions.		
Will participants be made aware of the circumstances in which disclosure has implications for confidentiality?		×	

Payments to participants

Question	Yes	No
Do you intend to offer participants cash payments or any kind of inducements, or reward for taking part in your study ?		×
If YES, please explain what kind of payment you will be offering(e.g.prize draw or store vouchers)		
Is there any possibility that such payments or inducements will cause participants to consent to risks that they might not otherwise find acceptable ?		×
If YES, please explain)		
Is there any possibility that the prospect of payment or inducements will influence the data provided by participants in any way ?		×
If YES, please explain)		
Will you inform participants that accepting payments or inducements does not affect their right to withdraw from the study at any time ?	×	

Capacity to give valid consent

Question	Yes	No
Do you propose to recruit any participants?		×
Do you propose to recruit any participants who are children or young people under 18 years of age?		×
Do you propose to recruit any participants who are adults who have learning difficulties, mental health conditions, brain injury, advanced dementia, degenerative neurological disorders ?		×
Do you propose to recruit any participants who are adults who are physically disabled and cannot provide written and/or verbal consent		×
Do you propose to recruit any participants who are with adults who are living in residential care, social care, nursing homes, reablement centres, hospitals or hospices ?		×
Do you propose to recruit any participants who are with adults who are in prison, remanded on bail or in custody?		×
If you have answered YES to any of the questions above please explain overcome any challenges to gaining valid consent		
Do you propose to recruit any participants with possible communication difficulties, including difficulties arising from limited use of knowledge of the English language ?		×
If YES, please explain how you will overcome any challenges to gaining valid consent		
Do you propose to recruit participants who may not be able to fully understand the nature of the study, the foreseen implications or cannot provide consent?		×
If YES, please explain how you will overcome any challenges to gaining valid consent		

Online and Internet Research

Question		Yes	No
Will any part of your project involve collecting data via the internet or social media?		×	
If YES, please explain how you will obtain permission to collect data by these means	Data and Research Papers open sourced that are openly available in internet.		
Will this require consent to access?			×
If NO, please explain how you will get permission/ consent' to collect this information?	No direct consent is required as the project will use freely available data.		
Will you be collecting data using an online questionnaire/ survey tool? (e.g. BoS, Filemaker)?			×
If YES, please explain which software and how you are ensuring appropriate data security			
Is there a possibility that the study will encourage children under 18 to access inappropriate websites, or correspond with people who pose risk of harm ?			×
If YES, please explain further			
Will the study incur any other risks that arise specifically from the use of electronic media ?			×
If YES, please explain further			

Information gathered from human participants

Question		Yes	No
Primary			
Does your project involve primary data collection from human participants via questionnaires, focus groups, interviews, psychological tests, photography/videography etc.?		×	
If YES, Please detail the information to be collected and methods that will be used.	The information will collected from semi-interviews and notes will be taken.		
Is there the possibility of physical or psychological harm to the researcher(s) or the participants?			×
If YES, please explain the possible harm and action taken to reduce/remove the risk			
Are any specific exclusions needed to prevent possible harm to participants (e.g. excluding people with known mental health problems)?			×
If YES, please explain exclusions needed and how these will be carried out			
Are any of the questionnaires or other tests being used in the research diagnostic for specific clinical conditions?			×
If YES, Please explain how you will take steps to reduce or address these risks			

LOG GENERATION

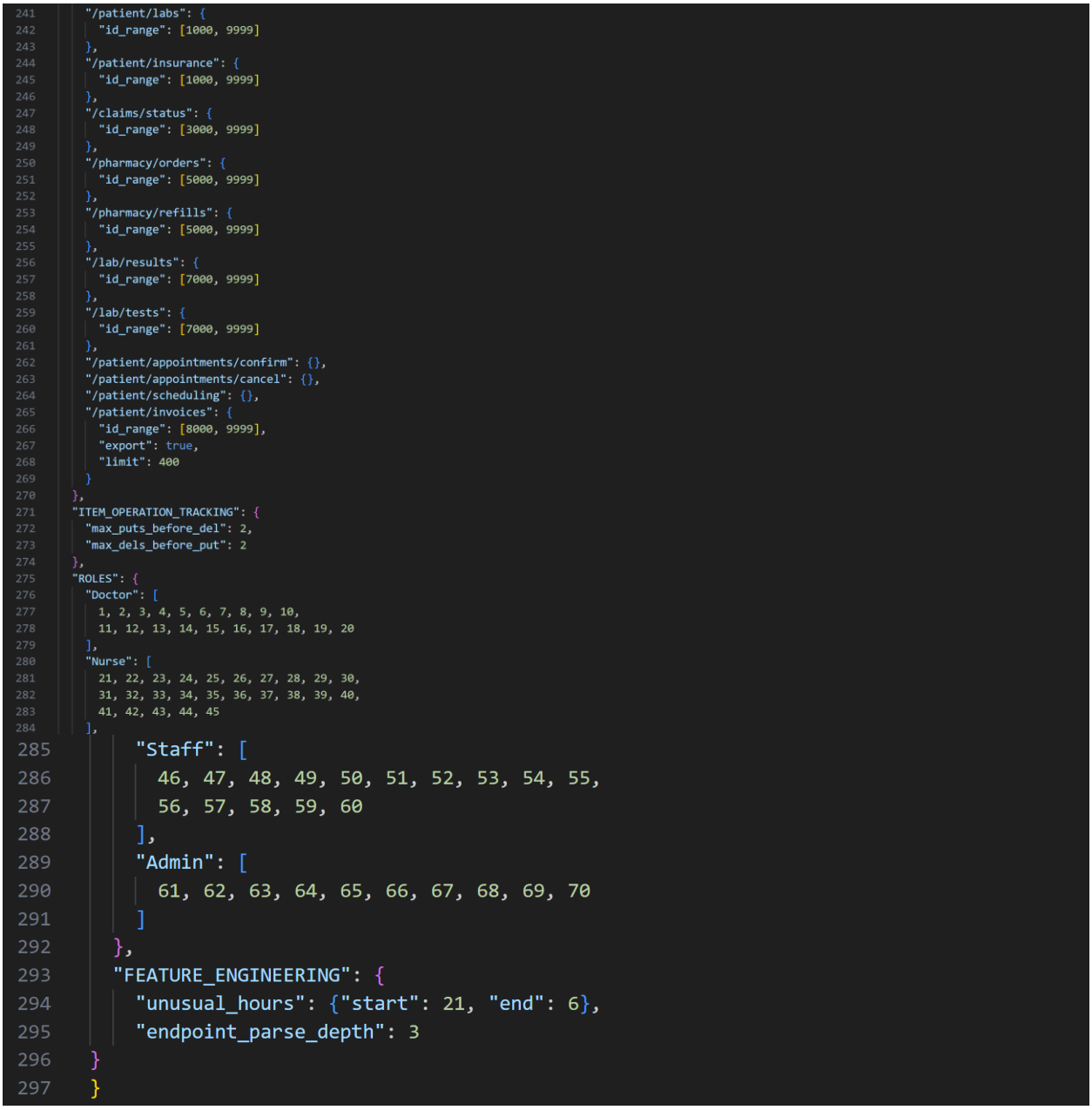
config.json > ...


```json
{
  "ROLE_WEIGHTS_NORMAL": {
    "Doctor": 0.15,
    "Nurse": 0.40,
    "Staff": 0.35,
    "Admin": 0.10
  },
  "NORMAL_METHODS": {
    "methods": ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"],
    "weights": [0.65, 0.20, 0.05, 0.02, 0.03, 0.03, 0.02]
  },
  "HTTP_RESPONSE_CODES": {
    "normal": {
      "codes": [200, 201, 302, 304, 403, 500],
      "weights": [0.65, 0.10, 0.10, 0.05, 0.05, 0.05]
    },
    "anomalous": {
      "codes": [200, 403, 404, 500],
      "weights": [0.30, 0.30, 0.25, 0.15]
    }
  },
  "IP_ADDRESS_GENERATION": {
    "anomalous_external_ip_probability": 0.1,
    "normal_external_ip_probability": 0.02,
    "ip_correlation_threshold": 0.3
  },
  "ANOMALY_SCENARIOS": {
    "Doctor": {
      "high_frequency_put_patient_records": 0.10,
      "delete_patient_records": 0.05,
      "delete_billing_invoices": 0.05,
      "patient_records_put_del": 0.15,
      "unusual_ip_access": 0.10,
      "data_exfiltration": 0.10,
      "external_ip_only": 0.15,
      "sql_injection": 0.03,
      "xss": 0.02
    },
    "Nurse": {
      "high_frequency_put_patient_records": 0.10,
      "delete_patient_records": 0.05,
      "delete_billing_invoices": 0.05,
      "patient_records_put_del": 0.15,
      "unusual_ip_access": 0.10,
      "data_exfiltration": 0.10,
      "external_ip_only": 0.15,
      "sql_injection": 0.03,
      "xss": 0.02
    },
    "Staff": {
      "rapid_inventory_changes": 0.15,
      "access_restricted_endpoints": 0.05,
      "inventory_put_del": 0.25,
      "billing_invoices_put_del": 0.30,
      "data_exfiltration": 0.05,
      "external_ip_only": 0.05,
      "multiple_put_del_same_item": 0.10,
      "multiple_delete_same_item": 0.10,
      "access_patient_invoices": 0.10,
      "sql_injection": 0.02,
      "xss": 0.02
    },
    "Admin": {
      "delete_login_attempts": 0.10,
      "delete_admin_logs": 0.10,
      "access_patient_records": 0.10,
      "admin_suspicious": 0.20,
      "data_exfiltration": 0.10,
      "privilege_escalation": 0.10,
      "multiple_failed_attempts": 0.10,
      "sql_injection": 0.05,
      "xss": 0.05,
      "external_ip_only": 0.05,
      "unauthorized_endpoint_access": 0.10
    }
  },
```

```
  "ENDPOINTS": {
    "Doctor": [
      "/patient/records",
      "/patient/labs",
      "/patient/appointments",
      "/billing/invoices",
      "/pharmacy/orders",
      "/pharmacy/refills",
      "/patient/appointments/confirm",
      "/patient/appointments/cancel",
      "/patient/insurance",
      "/claims/status",
      "/lab/results",
      "/lab/tests"
    ],
    "Nurse": [
      "/patient/records",
      "/patient/labs",
      "/patient/appointments",
      "/billing/invoices",
      "/pharmacy/orders",
      "/pharmacy/refills",
      "/patient/appointments/confirm",
      "/patient/appointments/cancel",
      "/patient/insurance",
      "/claims/status",
      "/lab/results",
      "/lab/tests"
    ],
    "Staff": [
      "/inventory/items",
      "/billing/invoices",
      "/patient/scheduling",
      "/pharmacy/orders",
      "/pharmacy/refills",
      "/claims/status",
      "/lab/results",
      "/lab/tests",
      "/patient/invoices"
    ],
    "Admin": [
      "/admin/settings",
      "/admin/credentials",
      "/admin/logs",
      "/admin/users",
      "/patient/records"
    ]
  },
  "ANOMALOUS_ENDPOINTS": {
    "high_frequency_put_patient_records": [
      ["/patient/records", "PUT"]
    ],
    "delete_patient_records": [
      ["/patient/records", "DELETE"]
    ],
    "delete_billing_invoices": [
      ["/billing/invoices", "DELETE"]
    ],
    "patient_records_put_del": [
      ["/patient/records", "PUT"],
      ["/patient/records", "DELETE"]
    ],
    "unusual_ip_access": [
      ["/patient/records", "GET"],
      ["/lab/results", "GET"],
      ["/billing/invoices", "GET"]
    ],
    "data_exfiltration": [
      ["/patient/records", "GET"],
      ["/billing/invoices", "GET"],
      ["/inventory/items", "GET"]
    ],
    "external_ip_only": [
      ["/patient/records", "GET"],
      ["/billing/invoices", "GET"],
      ["/inventory/items", "GET"],
      ["/patient/appointments", "GET"],
      ["/patient/invoices", "GET"]
    ],
    "multiple_failed_attempts": [
      ["/login/attempts", "DELETE"]
    ],
    "sql_injection": [
      ["/admin/settings", "POST"],
      ["/patient/records", "GET"]
```

```
    "xss": [
      ["/admin/users", "PUT"]
    ],
    "rapid_inventory_changes": [
      ["/inventory/items", "PUT"],
      ["/inventory/items", "DELETE"],
      ["/inventory/items", "PUT"]
    ],
    "access_restricted_endpoints": [
      ["/lab/results", "GET"],
      ["/lab/tests", "GET"],
      ["/lab/tests", "DELETE"],
      ["/pharmacy/orders", "DELETE"]
    ],
    "access_patient_invoices": [
      ["/patient/invoices", "GET"],
      ["/patient/invoices", "PUT"]
    ],
    "delete_login_attempts": [
      ["/login/attempts", "DELETE"]
    ],
    "delete_admin_logs": [
      ["/admin/logs", "DELETE"]
    ],
    "access_patient_records": [
      ["/patient/records", "GET"],
      ["/patient/records", "PUT"],
      ["/patient/records", "DELETE"]
    ],
    "unauthorized_endpoint_access": [
      ["/patient/records", "GET"],
      ["/patient/records", "DELETE"],
      ["/admin/users", "DELETE"]
    ],
    "admin_suspicious": [
      ["/admin/settings", "PATCH"],
      ["/admin/credentials", "PUT"],
      ["/admin/credentials", "GET"],
      ["/admin/logs", "GET"]
    ],
    "privilege_escalation": [
      ["/admin/users", "POST"],
      ["/admin/credentials", "POST"]
    ]
  },
```

```
  "PARAMETERS": {
    "/patient/records": {
      "id_range": [1000, 9999],
      "export": true,
      "limit": 1000
    },
    "/billing/invoices": {
      "id_range": [2000, 9999],
      "export": true,
      "limit": 500
    },
    "/inventory/items": {
      "id_range": [4000, 9999],
      "export": true,
      "limit": 300
    },
    "/admin/settings": {
      "id_range": [61, 70]
    },
    "/admin/users": {
      "id_range": [61, 70]
    },
    "/admin/credentials": {
      "id_range": [61, 70]
    },
    "/login/attempts": {
      "id_range": [1, 70],
      "attempts": [3, 10],
      "weights": [0.5, 0.5]
    },
    "/patient/appointments": {
      "id_range": [500, 1000]
    },
```



```
    "/patient/labs": {
      "id_range": [1000, 9999]
    },
    "/patient/insurance": {
      "id_range": [1000, 9999]
    },
    "/claims/status": {
      "id_range": [3000, 9999]
    },
    "/pharmacy/orders": {
      "id_range": [5000, 9999]
    },
    "/pharmacy/refills": {
      "id_range": [5000, 9999]
    },
    "/lab/results": {
      "id_range": [7000, 9999]
    },
    "/lab/tests": {
      "id_range": [7000, 9999]
    },
    "/patient/appointments/confirm": {},
    "/patient/appointments/cancel": {},
    "/patient/scheduling": {},
    "/patient/invoices": {
      "id_range": [8000, 9999],
      "export": true,
      "limit": 400
    }
  },
  "ITEM_OPERATION_TRACKING": {
    "max_puts_before_del": 2,
    "max_dels_before_put": 2
  },
  "ROLES": {
    "Doctor": [
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
      11, 12, 13, 14, 15, 16, 17, 18, 19, 20
    ],
    "Nurse": [
      21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
      31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
      41, 42, 43, 44, 45
    ],
      "Staff": [
        46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
        56, 57, 58, 59, 60
      ],
      "Admin": [
        61, 62, 63, 64, 65, 66, 67, 68, 69, 70
      ]
    },
    "FEATURE_ENGINEERING": {
      "unusual_hours": {"start": 21, "end": 6},
      "endpoint_parse_depth": 3
  }
  }
```

Figure39: Config.json

```python
# main.py

import argparse
import logging
from config_loader import load_config
from log_generator import LogGenerator

def parse_arguments():
    """
    Parses command-line arguments for the log generator.
    """
    parser = argparse.ArgumentParser(description="Enhanced Synthetic Log Generator for Healthcare Environment")
    parser.add_argument(
        "--total_logs",
        type=int,
        default=100000,
        help="Total number of logs to generate (default: 100000)"
    )
    parser.add_argument(
        "--anomaly_ratio",
        type=float,
        default=0.02,
        help="Ratio of anomalous logs (default: 0.02)"
    )
    parser.add_argument(
        "--master_file",
        type=str,
        default="data/master_logs.csv",
        help="Path to the master CSV file (with labels) (default: data/master_logs.csv)"
    )
    parser.add_argument(
        "--inference_file",
        type=str,
        default="data/inference_logs.csv",
        help="Path to the inference CSV file (without labels) (default: data/inference_logs.csv)"
    )
    parser.add_argument(
        "--network_subnet",
        type=str,
        default="10.0.0",
        help="Base subnet for internal IP addresses (default: 10.0.0)"
    )
```

```python
    parser.add_argument(
        "--extended_days",
        type=int,
        default=90,
        help="Number of days to simulate logs from the past (default: 90)"
    )
    parser.add_argument(
        "--config_file",
        type=str,
        default="config.json",
        help="Path to the configuration JSON file (default: config.json)"
    )
    parser.add_argument(
        "--overwrite",
        action='store_true',
        help="Overwrite existing CSV files instead of appending"
    )
    parser.add_argument(
        "--seed",
        type=int,
        default=None,
        help="Random seed for reproducibility (default: None)"
    )
    return parser.parse_args()

def main():
    """
    Main entry point for generating logs.
    """
    # Configure logging
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

    args = parse_arguments()
    total_logs = args.total_logs
    anomaly_ratio = args.anomaly_ratio
    master_file = args.master_file
    inference_file = args.inference_file
    network_subnet = args.network_subnet
    extended_days = args.extended_days
    config_file = args.config_file
    overwrite = args.overwrite
    seed = args.seed

```

```python
    # Set random seed if provided
    if seed is not None:
        import random
        random.seed(seed)
        logging.info(f"Random seed set to {seed}")

    # Load configuration
    config = load_config(config_file)

    # Initialize LogGenerator
    generator = LogGenerator(config, network_subnet, extended_days)

    logging.info(f"Starting log generation with the following parameters:")
    logging.info(f"Total Logs: {total_logs}")
    logging.info(f"Anomaly Ratio: {anomaly_ratio * 100}%")
    logging.info(f"Master File: {master_file}")
    logging.info(f"Inference File: {inference_file}")
    logging.info(f"Network Subnet: {network_subnet}.x")
    logging.info(f"Extended Days: {extended_days}")
    logging.info(f"Configuration File: {config_file}")
    logging.info(f"Overwrite Mode: {'Enabled' if overwrite else 'Disabled'}\n")

    # Generate logs
    logs = generator.generate_logs(total_logs, anomaly_ratio)

    total_anomalies = sum(log["Anomalous"] for log in logs)
    logging.info(f"Generated {len(logs)} logs. Anomalies: {total_anomalies}")

    # Validate logs
    generator.validate_logs(logs, anomaly_ratio)

    # Save logs to CSV
    generator.save_logs_to_csv(logs, master_file, inference_file, overwrite=overwrite)
    logging.info(f"Logs saved to '{master_file}' and '{inference_file}'.")
    logging.info("Log generation completed successfully.")

if __name__ == "__main__":
    main()
```

Figure40: Main.py

```python
# utils.py

import random
import ipaddress
import re

def generate_public_ip():
    """
    Generates a random public IP address, ensuring it's not private or reserved.
    """
    while True:
        ip_int = random.randint(1, (2**32) - 1)
        ip = ipaddress.IPv4Address(ip_int)
        if not (ip.is_private or ip.is_reserved or ip.is_multicast or ip.is_loopback or ip.is_unspecified):
            return str(ip)

def generate_ip_address(is_anomalous, network_subnet, config):
    """
    Generates an IP address based on whether the log is anomalous.
    - Anomalous: Higher probability of public IP
    - Normal: Lower probability of public IP
    """
    ip_config = config.get("IP_ADDRESS_GENERATION", {})
    anomalous_prob = ip_config.get("anomalous_external_ip_probability", 0.3)
    normal_prob = ip_config.get("normal_external_ip_probability", 0.01)

    if is_anomalous:
        if random.random() < anomalous_prob:
            return generate_public_ip()
        else:
            return f"{network_subnet}.{random.randint(1, 254)}"
    else:
        if random.random() < normal_prob:
            return generate_public_ip()
        else:
            return f"{network_subnet}.{random.randint(1, 254)}"
```

```python
def extract_item_id(endpoint_full):
    """
    Extracts the item_id from the endpoint using regex.
    Returns the item_id as an integer if found, else None.
    """
    match = re.search(r"/inventory/items/(\d+)", endpoint_full)
    if match:
        try:
            return int(match.group(1))
        except ValueError:
            print(f"Warning: Invalid item_id extracted: {match.group(1)}")
            return None
    return None

def build_parameters(endpoint_base, config, is_anomalous=False, fixed_id=None, payload=None):
    """
    Build parameters with optional fixed ID and payload injection
    """
    params = config["PARAMETERS"].get(endpoint_base, {})
    param_str = ""

    # Use fixed ID if provided, else random
    resource_id = fixed_id or str(random.randint(*params["id_range"])) if "id_range" in params else ""

    if resource_id:
        param_str += f"/{resource_id}"

    # Add parameters
    if "export" in params:
        param_str += f"?export=true&limit={params['limit']}"

    # Inject payload for anomalies
    if payload:
        param_str += f"&query={payload}" if "?" in param_str else f"?query={payload}"

    return param_str

```

```python
def validate_anomaly_consistency(logs):
    """
    Validates consistency of IDs and IPs in anomaly sequences
    with enhanced error handling
    """
    if not logs:
        return

    try:
        # Get first log's IP as reference
        ref_ip = logs[0]["IP_Address"]

        # Check for resource ID consistency only if endpoints contain IDs
        ref_id = None
        if any('/items/' in log['Endpoint'] or '/records/' in log['Endpoint'] for log in logs):
            first_with_id = next(log for log in logs if '/items/' in log['Endpoint'] or '/records/' in log['Endpoint'])
            ref_id = first_with_id['Endpoint'].split("/")[3].split("?")[0]

        for log in logs:
            # Check IP consistency
            if log["IP_Address"] != ref_ip:
                raise ValueError(f"IP mismatch in chain: {ref_ip} vs {log['IP_Address']}")

            # Check ID consistency if applicable
            if ref_id:
                current_endpoint = log["Endpoint"]
                if '/items/' in current_endpoint or '/records/' in current_endpoint:
                    current_id = current_endpoint.split("/")[3].split("?")[0]
                    if current_id != ref_id:
                        raise ValueError(f"ID mismatch: {ref_id} vs {current_id}")

    except (IndexError, StopIteration):
        # Skip validation if no ID-based endpoints
        pass
```

Figure41: utils.py

```python
# log_generator.py

import random
import uuid
import datetime
from collections import defaultdict
from utils import generate_ip_address, extract_item_id, build_parameters
from utils import validate_anomaly_consistency
from anomaly_scenarios import (
    generate_multiple_failed_attempts,
    generate_chain_of_anomalies,
    generate_sql_injection,
    generate_xss,
    generate_external_ip_only,
    generate_unusual_time_access,
    generate_delete_patient_records,
    generate_delete_billing_invoices,
    generate_patient_records_put_del,
    generate_rapid_inventory_changes,
    generate_access_restricted_endpoints,
    generate_access_patient_invoices,
    generate_delete_login_attempts,
    generate_delete_admin_logs,
    generate_access_patient_records,
    generate_admin_suspicious,
    generate_privilege_escalation,
    generate_unauthorized_endpoint_access
)
import ipaddress
import math

class LogGenerator:
    def __init__(self, config, network_subnet, extended_days):
        self.config = config
        self.network_subnet = network_subnet
        self.extended_days = extended_days
        self.item_operation_tracker = defaultdict(lambda: {'PUT': 0, 'DELETE': 0})
        self.roles = config["ROLES"]
        self.endpoints = config["ENDPOINTS"]
        self.average_logs_per_scenario = self.calculate_average_logs_per_scenario()
        print(f"Initialized LogGenerator with average_logs_per_scenario: {self.average_logs_per_scenario}")
```

```python
    def calculate_average_logs_per_scenario(self):
        """
        Calculates the average number of logs generated per anomaly scenario.
        This considers scenario weights, role-based variations, and the actual number of logs each scenario generates.
        """
        scenario_weights = self.config["ANOMALY_SCENARIOS"]
        # Estimate logs per scenario
        logs_per_scenario = {
            "multiple_failed_attempts": 5,
            "chain_of_anomalies": 3,
            "sql_injection": 2,
            "xss": 2,
            "external_ip_only": 1,
            "unusual_time_access": 1,
            "delete_patient_records": 1,
            "delete_billing_invoices": 1,
            "patient_records_put_del": 2,
            "rapid_inventory_changes": 3,
            "access_restricted_endpoints": 1,
            "access_patient_invoices": 1,
            "delete_login_attempts": 1,
            "delete_admin_logs": 1,
            "access_patient_records": 2,
            "admin_suspicious": 2,
            "privilege_escalation": 2,
            "unauthorized_endpoint_access": 2
        }

        total_weighted_logs = 0
        total_weights = 0
        for role, scenarios in scenario_weights.items():
            for scenario, weight in scenarios.items():
                log_count = logs_per_scenario.get(scenario, 1)  # Default to 1 log if undefined
                total_weighted_logs += weight * log_count
                total_weights += weight

        if total_weights == 0:
            print("Warning: Total weights for anomaly scenarios is zero. Defaulting to 1.")
            return 1

        average_logs_per_scenario = total_weighted_logs / total_weights
        print(f"Calculated precise average logs per anomaly scenario: {average_logs_per_scenario}")
        return average_logs_per_scenario
```

```python

    def weighted_hour_choice(self, is_after_hours=False):
        """
        Returns an hour (0-23) for normal logs based on a weighted distribution,
        simulating a busier daytime environment but still some overnight activity.
        If is_after_hours is True, selects from after-hours.
        """
        if is_after_hours:
            possible_hours = list(range(21, 24)) + list(range(0, 6))
            weights = [1] * len(possible_hours)
            chosen_hour = random.choices(possible_hours, weights=weights, k=1)[0]
            print(f"Selected after-hours hour: {chosen_hour}")
            return chosen_hour
        else:
            hours = list(range(24))  # 0 through 23
            weights = [
                1, 1, 1, 1, 1, 1,  # 0-6
                8, 8, 8, 8, 8, 8,  # 7-13
                6, 6, 6, 6, 6,     # 14-18
                3, 3, 3, 3, 3      # 19-23
            ]
            chosen_hour = random.choices(hours, weights=weights, k=1)[0]
            # print(f"Selected normal hour: {chosen_hour}")
            return chosen_hour

    def generate_http_response_code(self, is_anomalous=False):
        """
        Returns an HTTP response code with different distributions
        for normal vs. anomalous logs.
        """
        if not is_anomalous:
            code = random.choices(
                self.config["HTTP_RESPONSE_CODES"]["normal"]["codes"],
                weights=self.config["HTTP_RESPONSE_CODES"]["normal"]["weights"],
                k=1
            )[0]
            # print(f"Generated normal HTTP response code: {code}")
            return code
        else:
            code = random.choices(
                self.config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"],
                weights=self.config["HTTP_RESPONSE_CODES"]["anomalous"]["weights"],
                k=1
            )[0]
```

```python
            # print(f"Generated anomalous HTTP response code: {code}")
            return code

    def pick_anomaly_scenario(self, role):
        """
        Returns a scenario label that decides which endpoint + method to use for anomalies.
        """
        scenarios = self.config["ANOMALY_SCENARIOS"].get(role, {})
        if not scenarios:
            print(f"No anomaly scenarios defined for role {role}. Defaulting to 'unknown'.")
            return "unknown"
        labels = list(scenarios.keys())
        weights = list(scenarios.values())
        chosen_scenario = random.choices(labels, weights=weights, k=1)[0]
        # print(f"Picked anomaly scenario '{chosen_scenario}' for role '{role}'.")
        return chosen_scenario
    @staticmethod
    def is_anomalous_combo(endpoint, method, config):
    # Scan all anomaly combos in ANOMALOUS_ENDPOINTS
        for scenario, combo_list in config["ANOMALOUS_ENDPOINTS"].items():
            for (anom_endpoint, anom_method) in combo_list:
                if anom_endpoint == endpoint and anom_method == method:
                    return True
        return False


    def generate_normal_log(self):
        # Weighted role selection
        role_choices = list(self.config["ROLE_WEIGHTS_NORMAL"].keys())
        weights_list = list(self.config["ROLE_WEIGHTS_NORMAL"].values())
        role = random.choices(role_choices, weights=weights_list, k=1)[0]
        user_id = random.choice(self.roles[role])

        # Weighted hour for normal
        chosen_hour = self.weighted_hour_choice()
        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, self.extended_days),
            hours=chosen_hour,
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59),
        )
```

```
        # Normal IP generation
        ip_address = generate_ip_address(is_anomalous=False, network_subnet=self.network_subnet, config=self.config)

        # Weighted HTTP method
        http_method = random.choices(
            self.config["NORMAL_METHODS"]["methods"],
            weights=self.config["NORMAL_METHODS"]["weights"],
            k=1
        )[0]

        # Pick an endpoint from the role
        endpoint = random.choice(self.endpoints[role])

        # Check if (endpoint, method) is anomalous
        if self.is_anomalous_combo(endpoint, http_method, self.config):
            # Option (a): re-roll to avoid anomaly
            return self.generate_normal_log()
        parameter = build_parameters(endpoint, self.config)
        endpoint_full = endpoint + parameter

        http_response = self.generate_http_response_code(is_anomalous=False)

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": http_method,
            "Endpoint": endpoint_full,
            "IP_Address": ip_address,
            "HTTP_Response": http_response,
            "Anomalous": 0
        }
        return log_entry
```

```
    def generate_anomalous_log(self):
        """
        Generates anomalous logs based on the selected scenario.
        Returns a list of log entries.
        """
        logs = []
        role = random.choice(list(self.roles.keys()))
        user_id = random.choice(self.roles[role])

        scenario_label = self.pick_anomaly_scenario(role)

        if scenario_label == "multiple_failed_attempts":
            anomaly_logs = generate_multiple_failed_attempts(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "chain_of_anomalies":
            anomaly_logs = generate_chain_of_anomalies(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "sql_injection":
            anomaly_logs = generate_sql_injection(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "xss":
            anomaly_logs = generate_xss(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "external_ip_only":
            anomaly_logs = generate_external_ip_only(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "unusual_time_access":
            anomaly_logs = generate_unusual_time_access(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
```

```
        elif scenario_label == "delete_patient_records":
            anomaly_logs = generate_delete_patient_records(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "delete_billing_invoices":
            anomaly_logs = generate_delete_billing_invoices(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "patient_records_put_del":
            anomaly_logs = generate_patient_records_put_del(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "rapid_inventory_changes":
            anomaly_logs = generate_rapid_inventory_changes(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "access_restricted_endpoints":
            anomaly_logs = generate_access_restricted_endpoints(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "access_patient_invoices":
            anomaly_logs = generate_access_patient_invoices(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "delete_login_attempts":
            anomaly_logs = generate_delete_login_attempts(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "delete_admin_logs":
            anomaly_logs = generate_delete_admin_logs(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
```

```
        elif scenario_label == "access_patient_records":
            anomaly_logs = generate_access_patient_records(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "admin_suspicious":
            anomaly_logs = generate_admin_suspicious(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "privilege_escalation":
            anomaly_logs = generate_privilege_escalation(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        elif scenario_label == "unauthorized_endpoint_access":
            anomaly_logs = generate_unauthorized_endpoint_access(
                user_id, role, self.network_subnet, self.config, self.extended_days
            )
            logs.extend(anomaly_logs)
        else:
            # Handle unknown scenarios if necessary
            pass

        return logs

    def generate_logs(self, total_logs, anomaly_ratio):
        """
        Generates normal and anomalous logs, ensuring the total number of logs matches `total_logs`.
        Dynamically adjusts anomaly and normal logs to meet the desired anomaly ratio.
        """
        desired_anomalous_logs = int(total_logs * anomaly_ratio)
        print(f"Desired Anomalous Logs: {desired_anomalous_logs}")

        num_anomaly_calls = math.ceil(desired_anomalous_logs / self.average_logs_per_scenario)
        print(f"Number of Anomaly Calls to Generate: {num_anomaly_calls}")

        log_groups = []
        actual_anomalous_logs = 0

```

```
        # Generate anomalous logs
        for _ in range(num_anomaly_calls):
            anomaly_logs = self.generate_anomalous_log()
            log_groups.append(anomaly_logs)
            actual_anomalous_logs += len(anomaly_logs)

        print(f"Actual Anomalous Logs Generated: {actual_anomalous_logs}")

        # Calculate the remaining logs needed to meet `total_logs`
        remaining_logs = total_logs - actual_anomalous_logs
        print(f"Remaining Logs to Generate: {remaining_logs}")

        # Adjust normal logs based on remaining capacity
        normal_log_groups = []
        for _ in range(remaining_logs):
            normal_log = self.generate_normal_log()
            normal_log_groups.append([normal_log])  # Single-element group

        # Check if anomalies need further adjustment
        if actual_anomalous_logs < desired_anomalous_logs:
            additional_anomalous_logs = desired_anomalous_logs - actual_anomalous_logs
            print(f"Generating {additional_anomalous_logs} additional anomaly logs to meet the desired count.")

            for _ in range(math.ceil(additional_anomalous_logs / self.average_logs_per_scenario)):
                anomaly_logs = self.generate_anomalous_log()
                log_groups.append(anomaly_logs)
                actual_anomalous_logs += len(anomaly_logs)
                remaining_logs -= len(anomaly_logs)
                if remaining_logs <= 0:
                    break

        # Ensure total logs are within limits
        total_generated_logs = sum(len(group) for group in log_groups) + len(normal_log_groups)
        if total_generated_logs > total_logs:
            excess = total_generated_logs - total_logs
            print(f"Trimming {excess} logs to maintain total_logs constraint.")
            # Trim excess from normal logs
            if len(normal_log_groups) >= excess:
                normal_log_groups = normal_log_groups[:-excess]
            else:
                # If excess is larger than normal logs, trim accordingly
                normal_log_groups = []

    # Combine and shuffle log groups
    log_groups.extend(normal_log_groups)
    random.shuffle(log_groups)

    # Flatten logs into a single list
    logs = [log for group in log_groups for log in group]
    print(f"Total logs after shuffling and flattening: {len(logs)}")

    # Final validation
    anomaly_count = sum(log["Anomalous"] for log in logs)
    print(f"Generated {len(logs)} logs. Anomalies: {anomaly_count}")
    # validate_anomaly_consistency(logs)


    return logs

def validate_logs(self, logs, expected_anomaly_ratio):
    """
    Validates that the generated logs meet the expected anomaly ratio
    and that external IP correlation is within acceptable limits.
    """
    total_logs = len(logs)
    actual_anomalies = sum(log["Anomalous"] for log in logs)
    actual_ratio = actual_anomalies / total_logs if total_logs > 0 else 0
    print(f"Expected Anomaly Ratio: {expected_anomaly_ratio * 100}%")
    print(f"Actual Anomaly Ratio: {actual_ratio * 100:.2f}%")
    tolerance = 0.05  # 5% tolerance
    if not abs(actual_ratio - expected_anomaly_ratio) <= tolerance:
        print("WARNING: Anomaly ratio deviates significantly from expected.")
    else:
        print("Anomaly ratio is within the acceptable range.")

    # Calculate correlation between External IP and Anomalous
    external_ips = sum(1 for log in logs if ipaddress.ip_address(log["IP_Address"]).is_global)
    anomalous_external_ips = sum(
        1 for log in logs
        if log["Anomalous"] == 1 and ipaddress.ip_address(log["IP_Address"]).is_global
    )

    external_ip_ratio = external_ips / total_logs if total_logs > 0 else 0
    anomalous_external_ip_ratio = anomalous_external_ips / total_logs if total_logs > 0 else 0
```

```
412
413         print(f"Total External IPs: {external_ips} ({external_ip_ratio * 100:.2f}%)")
414         print(f"Anomalous External IPs: {anomalous_external_ips} ({anomalous_external_ip_ratio * 100:.2f}%)")
415
416         # Define acceptable correlation thresholds
417         ip_correlation_threshold = self.config["IP_ADDRESS_GENERATION"].get("ip_correlation_threshold", 0.3)
418
419         # Calculate the proportion of external IPs that are anomalous
420         if external_ips > 0:
421             proportion_anomalous_external = anomalous_external_ips / external_ips
422         else:
423             proportion_anomalous_external = 0
424
425         print(f"Proportion of External IPs that are Anomalous: {proportion_anomalous_external * 100:.2f}%")
426
427         # Example check: Ensure that the proportion doesn't exceed the threshold
428         if proportion_anomalous_external > ip_correlation_threshold:
429             print("WARNING: High correlation between external IPs and anomalies detected.")
430         else:
431             print("Correlation between external IPs and anomalies is within acceptable limits.")
432
433     def save_logs_to_csv(self, logs, master_file, inference_file, overwrite=False):
434         """
435         Saves the generated logs to CSV files: master (with 'Anomalous') and inference (without).
436         """
437         import os
438         import csv
439
440         mode_master = "w" if overwrite else "a"
441         mode_inference = "w" if overwrite else "a"
442
443         # Ensure directories exist
444         os.makedirs(os.path.dirname(master_file), exist_ok=True)
445         os.makedirs(os.path.dirname(inference_file), exist_ok=True)
446
```

```
447         fieldnames_master = [
448             "LogID",
449             "UserID",
450             "Role",
451             "Timestamp",
452             "HTTP_Method",
453             "Endpoint",
454             "IP_Address",
455             "HTTP_Response",
456             "Anomalous"
457         ]
458         master_exists = os.path.isfile(master_file) and not overwrite
459         with open(master_file, mode_master, newline="") as csvfile:
460             writer = csv.DictWriter(csvfile, fieldnames=fieldnames_master)
461             if not master_exists or overwrite:
462                 writer.writeheader()
463             writer.writerows(logs)
464
465         fieldnames_inference = [
466             "LogID",
467             "UserID",
468             "Role",
469             "Timestamp",
470             "HTTP_Method",
471             "Endpoint",
472             "IP_Address",
473             "HTTP_Response"
474         ]
475         inference_exists = os.path.isfile(inference_file) and not overwrite
476         with open(inference_file, mode_inference, newline="") as csvfile:
477             writer = csv.DictWriter(csvfile, fieldnames=fieldnames_inference)
478             if not inference_exists or overwrite:
479                 writer.writeheader()
480             for log in logs:
481                 inference_log = {k: v for k, v in log.items() if k != "Anomalous"}
482                 writer.writerow(inference_log)
483
484         print(f"Saved {len(logs)} logs to '{master_file}' and '{inference_file}'.")
485
```

Figure42: log_generator.py

```python
# anomaly_scenarios.py
import random
import uuid
import datetime
import ipaddress
from utils import generate_ip_address, generate_public_ip, build_parameters
import logging
import sys

def generate_multiple_failed_attempts(user_id, role, network_subnet, config, extended_days):
    """
    Generates multiple DELETE operations on /login/attempts to simulate failed login attempts.
    """
    logs = []
    num_attempts = random.randint(3, 10)
    base_timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    for i in range(num_attempts):
        timestamp = base_timestamp + datetime.timedelta(seconds=i * 10)  # 10 seconds apart
        ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

        http_method = "DELETE"
        endpoint = "/login/attempts"
        parameter = build_parameters(endpoint, config)
        endpoint_full = endpoint + parameter

        http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": http_method,
            "Endpoint": endpoint_full,
            "IP_Address": ip_address,
            "HTTP_Response": http_response,
            "Anomalous": 1
        }

        logs.append(log_entry)

    return logs

def generate_rapid_inventory_changes(user_id, role, network_subnet, config, extended_days):
    """
    Generates rapid PUT/DELETE operations on the SAME item
    """
    logs = []
    endpoint = "/inventory/items"

    # Generate single item ID for all operations
    item_id = str(random.randint(*config["PARAMETERS"][endpoint]["id_range"]))

    base_timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        minutes=random.randint(0, 59)
    )

    # Same IP for all operations
    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    operations = ["PUT", "DELETE", "PUT"]
    for i, method in enumerate(operations):
        timestamp = base_timestamp + datetime.timedelta(seconds=i*10)
        params = build_parameters(endpoint, config, fixed_id=item_id)  # Fixed ID

        logs.append({
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": method,
            "Endpoint": f"{endpoint}{params}",
            "IP_Address": ip_address,
            "HTTP_Response": random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"]),
            "Anomalous": 1
        })

    return logs
```

```python
def generate_chain_of_anomalies(user_id, role, network_subnet, config, extended_days):
    """
    Generates chained anomalies with consistent target ID and IP
    """
    logs = []
    base_timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days)
    )

    # Shared parameters for the chain
    target_id = str(random.randint(1000, 9999))
    ip_address = generate_ip_address(is_anomalous=True,  # Add this line
                                     network_subnet=network_subnet,
                                     config=config)


    # Phase 1: Initial compromise
    sql_ep = random.choice(config["ANOMALOUS_ENDPOINTS"]["sql_injection"])
    sql_params = build_parameters(sql_ep[0], config, fixed_id=target_id, payload="' OR 1=1 --")

    logs.append({
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": base_timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": sql_ep[1],
        "Endpoint": f"{sql_ep[0]}{sql_params}",
        "IP_Address": ip_address,
        "HTTP_Response": 500,
        "Anomalous": 1
    })

    # Phase 2: Follow-up actions
    for exfil_ep in config["ANOMALOUS_ENDPOINTS"]["data_exfiltration"]:
        timestamp = base_timestamp + datetime.timedelta(minutes=5)
        exfil_params = build_parameters(exfil_ep[0], config, fixed_id=target_id)

        logs.append({
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": exfil_ep[1],
            "Endpoint": f"{exfil_ep[0]}{exfil_params}",
            "IP_Address": ip_address,
            "HTTP_Response": 200 if exfil_ep[1] == "GET" else 500,
            "Anomalous": 1
        })

    return logs
```

```python
def generate_sql_injection(user_id, role, network_subnet, config, extended_days):
    """
    Generates SQL Injection attempts on specified endpoints.
    """
    logs = []
    sql_endpoints = config["ANOMALOUS_ENDPOINTS"]["sql_injection"]
    num_injections = random.randint(1, len(sql_endpoints))
    selected_endpoints = random.sample(sql_endpoints, num_injections)

    for endpoint in selected_endpoints:
        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, extended_days),
            hours=random.randint(0, 23),
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59)
        )
        ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": endpoint[1],
            "Endpoint": endpoint[0] + build_parameters(endpoint[0], config),
            "IP_Address": ip_address,
            "HTTP_Response": random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"]),
            "Anomalous": 1
        }

        logs.append(log_entry)

    return logs
```

```python
def generate_xss(user_id, role, network_subnet, config, extended_days):
    """
    Generates XSS attempts on specified endpoints.
    """
    logs = []
    xss_endpoints = config["ANOMALOUS_ENDPOINTS"]["xss"]
    num_xss = random.randint(1, len(xss_endpoints))
    selected_endpoints = random.sample(xss_endpoints, num_xss)

    for endpoint in selected_endpoints:
        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, extended_days),
            hours=random.randint(0, 23),
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59)
        )
        ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": endpoint[1],
            "Endpoint": endpoint[0] + build_parameters(endpoint[0], config),
            "IP_Address": ip_address,
            "HTTP_Response": random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"]),
            "Anomalous": 1
        }

        logs.append(log_entry)

    return logs

def generate_external_ip_only(user_id, role, network_subnet, config, extended_days):
    """
    Generates logs where only the IP address is external (anomalous).
    All other attributes are generated normally.
    """
    logs = []
    anomalous_endpoints = config["ANOMALOUS_ENDPOINTS"]["external_ip_only"]
    selected_endpoint = random.choice(anomalous_endpoints)

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    # Generate an external IP address
    ip_address = generate_public_ip()

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["normal"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_unusual_time_access(user_id, role, network_subnet, config, extended_days):
    """
    Generates access to sensitive endpoints during unusual hours.
    """
    logs = []
    anomalous_endpoints = config["ANOMALOUS_ENDPOINTS"]["unusual_ip_access"]
    selected_endpoint = random.choice(anomalous_endpoints)

    # Define unusual hours (e.g., 0-5 and 21-23)
    hour = random.choice(list(range(0, 6)) + list(range(21, 24)))

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    # Adjust the hour to be within unusual hours
    timestamp = timestamp.replace(hour=hour)

    ip_address = generate_ip_address(is_anomalous=False, network_subnet=network_subnet, config=config)

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

```

```
    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_delete_patient_records(user_id, role, network_subnet, config, extended_days):
    """
    Generates a DELETE operation on /patient/records, which is anomalous for Doctors and Nurses.
    """
    logs = []
    endpoint = "/patient/records"
    http_method = "DELETE"

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs
def generate_delete_billing_invoices(user_id, role, network_subnet, config, extended_days):
    """
    Generates a DELETE operation on /billing/invoices, which is anomalous for Doctors and Nurses.
    """
    logs = []
    endpoint = "/billing/invoices"
    http_method = "DELETE"

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_patient_records_put_del(user_id, role, network_subnet, config, extended_days):
    """
    Generates PUT and DELETE operations on /patient/records.
    """
    logs = []
    endpoints = config["ANOMALOUS_ENDPOINTS"]["patient_records_put_del"]

    for endpoint in endpoints:
        http_method = endpoint[1]
        endpoint_full = endpoint[0] + build_parameters(endpoint[0], config)

        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, extended_days),
            hours=random.randint(0, 23),
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59)
        )
```

```
        is_anomalous = 1 if http_method == "DELETE" else 0  # DELETE is anomalous

        ip_address = generate_ip_address(is_anomalous=is_anomalous, network_subnet=network_subnet, config=config)

        http_response = random.choice(
            config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"] if is_anomalous else config["HTTP_RESPONSE_CODES"]["normal"]["codes"]
        )

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": http_method,
            "Endpoint": endpoint_full,
            "IP_Address": ip_address,
            "HTTP_Response": http_response,
            "Anomalous": is_anomalous
        }

        logs.append(log_entry)

    return logs


def generate_access_restricted_endpoints(user_id, role, network_subnet, config, extended_days):
    """
    Generates access attempts to restricted endpoints by Staff.
    """
    logs = []
    restricted_endpoints = config["ANOMALOUS_ENDPOINTS"]["access_restricted_endpoints"]
    selected_endpoint = random.choice(restricted_endpoints)

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_access_patient_invoices(user_id, role, network_subnet, config, extended_days):
    """
    Generates access to patient invoices by Staff. Considered normal but monitored.
    """
    logs = []
    endpoints = config["ANOMALOUS_ENDPOINTS"]["access_patient_invoices"]

    for endpoint in endpoints:
        http_method = endpoint[1]
        endpoint_full = endpoint[0] + build_parameters(endpoint[0], config)

        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, extended_days),
            hours=random.randint(6, 20),  # Normal operational hours
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59)
        )

        # Accessing patient invoices is normal, but we monitor for any anomalies
        is_anomalous = 0  # Initially normal

        # Introduce a small chance of anomaly if needed
        if random.random() < 0.05:
            is_anomalous = 1
            http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])
        else:
            http_response = random.choice(config["HTTP_RESPONSE_CODES"]["normal"]["codes"])

        ip_address = generate_ip_address(is_anomalous=is_anomalous, network_subnet=network_subnet, config=config)

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": http_method,
            "Endpoint": endpoint_full,
            "IP_Address": ip_address,
```

```
            "HTTP_Response": http_response,
            "Anomalous": is_anomalous
        }

        logs.append(log_entry)

    return logs

def generate_delete_login_attempts(user_id, role, network_subnet, config, extended_days):
    """
    Generates DELETE operations on /login/attempts by Admins.
    """
    logs = []
    endpoint = "/login/attempts"
    http_method = "DELETE"

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs
```

```
def generate_delete_admin_logs(user_id, role, network_subnet, config, extended_days):
    """
    Generates DELETE operations on /admin/logs by Admins.
    """
    logs = []
    endpoint = "/admin/logs"
    http_method = "DELETE"

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_access_patient_records(user_id, role, network_subnet, config, extended_days):
    """
    Generates access to /patient/records by Admins.
    """
    logs = []
    endpoints = config["ANOMALOUS_ENDPOINTS"]["access_patient_records"]

    for endpoint in endpoints:
        http_method = endpoint[1]
        endpoint_full = endpoint[0] + build_parameters(endpoint[0], config)

        timestamp = datetime.datetime.now() - datetime.timedelta(
            days=random.randint(0, extended_days),
            hours=random.randint(6, 20),  # Normal operational hours
            minutes=random.randint(0, 59),
            seconds=random.randint(0, 59)
        )
```

```
        # Accessing patient records is monitored and potentially anomalous
        is_anomalous = 1  # Flag as anomalous since Admins typically shouldn't access patient records

        http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

        ip_address = generate_ip_address(is_anomalous=is_anomalous, network_subnet=network_subnet, config=config)

        log_entry = {
            "LogID": str(uuid.uuid4()),
            "UserID": user_id,
            "Role": role,
            "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "HTTP_Method": http_method,
            "Endpoint": endpoint_full,
            "IP_Address": ip_address,
            "HTTP_Response": http_response,
            "Anomalous": is_anomalous
        }

        logs.append(log_entry)

    return logs

def generate_admin_suspicious(user_id, role, network_subnet, config, extended_days):
    """
    Generates suspicious activities related to admin settings and credentials.
    """
    logs = []
    anomalous_endpoints = config["ANOMALOUS_ENDPOINTS"]["admin_suspicious"]
    selected_endpoint = random.choice(anomalous_endpoints)

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(6, 20),  # Normal operational hours
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
```

```
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_privilege_escalation(user_id, role, network_subnet, config, extended_days):
    """
    Generates attempts to escalate privileges or access higher-level functionalities.
    """
    logs = []
    anomalous_endpoints = config["ANOMALOUS_ENDPOINTS"]["privilege_escalation"]
    selected_endpoint = random.choice(anomalous_endpoints)

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(0, 23),
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs

def generate_unauthorized_endpoint_access(user_id, role, network_subnet, config, extended_days):
    """
    Generates attempts to access endpoints beyond typical Admin operations.
    """
    logs = []
    anomalous_endpoints = config["ANOMALOUS_ENDPOINTS"]["unauthorized_endpoint_access"]
    selected_endpoint = random.choice(anomalous_endpoints)
```

```python

    timestamp = datetime.datetime.now() - datetime.timedelta(
        days=random.randint(0, extended_days),
        hours=random.randint(6, 20),  # Normal operational hours
        minutes=random.randint(0, 59),
        seconds=random.randint(0, 59)
    )

    ip_address = generate_ip_address(is_anomalous=True, network_subnet=network_subnet, config=config)

    http_method = selected_endpoint[1]
    endpoint = selected_endpoint[0]
    parameter = build_parameters(endpoint, config)
    endpoint_full = endpoint + parameter

    http_response = random.choice(config["HTTP_RESPONSE_CODES"]["anomalous"]["codes"])

    log_entry = {
        "LogID": str(uuid.uuid4()),
        "UserID": user_id,
        "Role": role,
        "Timestamp": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
        "HTTP_Method": http_method,
        "Endpoint": endpoint_full,
        "IP_Address": ip_address,
        "HTTP_Response": http_response,
        "Anomalous": 1
    }

    logs.append(log_entry)

    return logs
```

Figures43: anamoly_scenarios.py

```python
import json
import logging
import sys

def load_config(config_file):
    """
    Loads and validates configuration parameters from a JSON file.
    """
    try:
        with open(config_file, 'r') as f:
            config = json.load(f)
    except FileNotFoundError:
        logging.error(f"Configuration file '{config_file}' not found.")
        sys.exit(1)
    except json.JSONDecodeError as e:
        logging.error(f"Error parsing JSON configuration: {e}")
        sys.exit(1)

    # Basic validation can be added here
    required_keys = [
        "ROLE_WEIGHTS_NORMAL", "NORMAL_METHODS", "HTTP_RESPONSE_CODES",
        "ANOMALY_SCENARIOS", "ENDPOINTS",
        "ANOMALOUS_ENDPOINTS", "PARAMETERS", "ITEM_OPERATION_TRACKING", "ROLES",
        "IP_ADDRESS_GENERATION"  # Ensure this key is included
    ]
    missing_keys = [key for key in required_keys if key not in config]
    if missing_keys:
        logging.error(f"Missing required configuration keys: {missing_keys}")
        sys.exit(1)

    return config
```

Figure44: config_loader.py

DATA PREPROCESSING

```python
#!/usr/bin/env python3

"""
Enhanced Anomaly Detection Data Preprocessing Script

Key Improvements:
1. Config-driven feature engineering
2. Proper endpoint parameter handling
3. Authorization checks aligned with anomaly config
4. Improved categorical encoding
5. Temporal features from config
6. Better handling of unseen categories

Usage:
    python3 data_preprocessing.py --mode train --config config.json
    python3 data_preprocessing.py --mode inference --config config.json
"""
#data_preprocessing.py

# ==============================
# 1. Import Necessary Libraries
# ==============================
import pandas as pd
import numpy as np
import os
import json
import ipaddress
import argparse
import sys
import pickle
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from utils import build_parameters  # Import from shared utilities

# ==============================
# 2. Configuration Handling
# ==============================
class ConfigManager:
    def __init__(self, config_file):
        self.config = self._load_config(config_file)
        self._validate_config()

    def _load_config(self, config_file):
        """Load and validate configuration file"""
        if not os.path.exists(config_file):
            raise FileNotFoundError(f"Config file {config_file} not found")

        try:
            with open(config_file, 'r') as f:
                config = json.load(f)

            # Set default feature engineering parameters
            config.setdefault('FEATURE_ENGINEERING', {
                'unusual_hours': {'start': 21, 'end': 6},
                'endpoint_parse_depth': 3
            })

            return config
```

```python
        except Exception as e:
            raise RuntimeError(f"Error loading config: {str(e)}")

    def _validate_config(self):
        """Ensure required configuration sections exist"""
        required_sections = [
            'ENDPOINTS', 'ANOMALOUS_ENDPOINTS', 'PARAMETERS',
            'ROLES', 'FEATURE_ENGINEERING'
        ]
        missing = [section for section in required_sections
                   if section not in self.config]
        if missing:
            raise ValueError(f"Missing config sections: {missing}")

# ==============================
# 3. Feature Engineering Utilities
# ==============================
class FeatureEngineer:
    def __init__(self, config):
        self.config = config
        self.feature_config = config['FEATURE_ENGINEERING']

    def clean_endpoint(self, endpoint):
        """Remove parameters and versioning from endpoint"""
        base_endpoint = endpoint.split('?')[0].split('/v')[0]
        return base_endpoint.rstrip('/')

    def parse_endpoint(self, endpoint):
        """Hierarchical endpoint parsing using config-defined depth"""
        depth = self.feature_config.get('endpoint_parse_depth', 3)
        parts = self.clean_endpoint(endpoint).split('/')[1:]  # Skip empty first element

        parsed = {}
        for i in range(depth):
            part = parts[i] if i < len(parts) else 'none'
            parsed[f'endpoint_level_{i+1}'] = part

        return parsed

    def is_authorized(self, role, endpoint, method):
        """Improved authorization check with path hierarchy awareness"""
        base_endpoint = self.clean_endpoint(endpoint)
        role_anomalies = self.config['ANOMALY_SCENARIOS'].get(role, {})
        # Check all anomaly definitions for this role's scenarios
        for scenario_name in role_anomalies:
            for ep_def in self.config['ANOMALOUS_ENDPOINTS'].get(scenario_name, []):
                config_ep = self.clean_endpoint(ep_def[0])
                config_method = ep_def[1]
```

```
                # Check if either:
                # 1. Exact endpoint match with method
                # 2. Base endpoint is parent path of request endpoint with method
                if (base_endpoint.startswith(config_ep) or
                    config_ep.startswith(base_endpoint)) and \
                    method == config_method:
                    return 0  # Unauthorized

        return 1  # Authorized

    def calculate_temporal_features(self, timestamp):
        """Calculate temporal features using config-defined parameters"""
        unusual_start = self.feature_config['unusual_hours']['start']
        unusual_end = self.feature_config['unusual_hours']['end']

        return {
            'hour': timestamp.hour,
            'day_of_week': timestamp.weekday(),
            'is_unusual_time': 1 if unusual_start <= timestamp.hour or timestamp.hour < unusual_end else 0
        }

# ==============================
# 4. Data Preprocessing Pipeline
# ==============================
class DataPreprocessor:
    def __init__(self, config_manager, mode='train'):
        self.config = config_manager.config
        self.mode = mode
        self.feature_engineer = FeatureEngineer(self.config)
        self.encoders = {}
        if self.mode == 'inference':
            encoder_path = 'models/encoders.pkl'
            if os.path.exists(encoder_path):
                try:
                    import pickle
                    with open(encoder_path, 'rb') as f:
                        self.encoders = pickle.load(f)
                    print("Loaded encoders from", encoder_path)
                except Exception as e:
                    raise ValueError(f"Error loading saved encoders: {e}")
            else:
                raise ValueError("No encoder file found. Please run training mode first to save encoders.")

    def load_data(self, file_path):
        """Load and validate raw data"""
        try:
            df = pd.read_csv(file_path)
            required_cols = ['LogID', 'Role', 'Endpoint', 'HTTP_Method', 'IP_Address', 'Timestamp']

            if not set(required_cols).issubset(df.columns):
                missing = set(required_cols) - set(df.columns)
                raise ValueError(f"Missing columns: {missing}")
```

```
            return df

        except Exception as e:
            raise RuntimeError(f"Error loading data: {str(e)}")

    def preprocess(self, df):
        """Main preprocessing pipeline"""
        # 1. Clean and parse endpoints
        df = self._process_endpoints(df)

        # 2. Handle temporal features
        df = self._process_temporal(df)

        # 3. IP address features
        df = self._process_ips(df)

        # 4. Authorization checks
        df = self._process_authorizations(df)

        # 5. Encode categorical features
        df = self._encode_features(df)

        # 6. Final cleanup
        return self._final_cleanup(df)

    def _process_endpoints(self, df):
        """Parse endpoints into hierarchical components"""
        endpoint_data = df['Endpoint'].apply(
            self.feature_engineer.parse_endpoint
        ).apply(pd.Series)

        return pd.concat([df, endpoint_data], axis=1)

    def _process_temporal(self, df):
        """Extract temporal features from timestamp"""
        df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')
        df = df.dropna(subset=['Timestamp'])  # Remove invalid timestamps

        temporal_data = df['Timestamp'].apply(
            self.feature_engineer.calculate_temporal_features
        ).apply(pd.Series)

        return pd.concat([df, temporal_data], axis=1)

    def _process_ips(self, df):
        """Process IP address features"""
        df['is_internal_ip'] = df['IP_Address'].apply(
            lambda ip: 1 if ipaddress.ip_address(ip).is_private else 0
        )
        return df.drop('IP_Address', axis=1)
```

```python

    def _process_authorizations(self, df):
        """Add authorization features"""
        df['is_authorized'] = df.apply(
            lambda row: self.feature_engineer.is_authorized(
                row['Role'], row['Endpoint'], row['HTTP_Method']
            ), axis=1
        )
        return df

    def _encode_features(self, df):
        """Smart encoding of categorical features"""
        # Label encode hierarchical endpoint components
        for col in [c for c in df.columns if c.startswith('endpoint_level_')]:
            df[col] = self._label_encode(col, df[col])

        # One-hot encode HTTP methods
        df = self._onehot_encode('HTTP_Method', df)

        # Target encoding for roles
        if self.mode == 'train' and 'Anomalous' in df.columns:
            role_encoding = df.groupby('Role')['Anomalous'].mean().to_dict()
            df['role_risk'] = df['Role'].map(role_encoding)

        return df

    # In your _label_encode method of DataPreprocessor:
    def _label_encode(self, col_name, series):
        if self.mode == 'train':
            encoder = LabelEncoder().fit(series)
            self.encoders[col_name] = encoder
            return encoder.transform(series)
        else:
            encoder = self.encoders.get(col_name)
            if not encoder:
                raise ValueError(f"No encoder found for {col_name}")
            return safe_label_encode(encoder, series)

    def _onehot_encode(self, col_name, df):
        """Handle one-hot encoding with column consistency"""
        if self.mode == 'train':
            encoder = OneHotEncoder(handle_unknown='ignore').fit(df[[col_name]])
            self.encoders[col_name] = encoder

        encoder = self.encoders[col_name]
        encoded = encoder.transform(df[[col_name]]).toarray()
        encoded_df = pd.DataFrame(
            encoded,
            columns=[f"{col_name}_{cat}" for cat in encoder.categories_[0]]
        )

        return pd.concat([df.drop(col_name, axis=1), encoded_df], axis=1)

```

```python

    def _final_cleanup(self, df):
        """Final cleanup and validation"""
        # Handle target variable
        if self.mode == 'inference' and 'Anomalous' in df.columns:
            df = df.drop('Anomalous', axis=1)

        # Ensure consistent column order
        cols = [c for c in df.columns if c not in ['LogID', 'Timestamp']]
        cols = ['LogID', 'Timestamp'] + cols  # Move ID and timestamp to front

        return df[cols]

# ==============================
# 5. Main Execution
# ==============================
def main():
    parser = argparse.ArgumentParser(description='Enhanced Data Preprocessor')
    parser.add_argument('--mode', choices=['train', 'inference'], required=True)
    parser.add_argument('--config', default='config.json')
    parser.add_argument('--input', default='data/master_logs.csv')
    parser.add_argument('--output', default='data/processed_data.pkl',
                        help='Output path for .pkl file; .csv will be created with same base name')
    args = parser.parse_args()

    try:
        # Initialize configuration
        config = ConfigManager(args.config)

        # Initialize preprocessor
        preprocessor = DataPreprocessor(config, args.mode)

        # Load and process data
        raw_data = preprocessor.load_data(args.input)
        processed_data = preprocessor.preprocess(raw_data)

        # Save processed data - Dual Format
        # 1. Save to Pickle (for model consumption)
        processed_data.to_pickle(args.output)

        # 2. Save to CSV (for human verification)
        base_path = os.path.splitext(args.output)[0]
        csv_path = f"{base_path}.csv"
        processed_data.to_csv(csv_path, index=False)

        print(f"Successfully processed data saved to:")
        print(f"- Pickle: {args.output}")
        print(f"- CSV:    {csv_path}")

        # Save encoders if in training mode
        if args.mode == 'train':
            with open('models/encoders.pkl', 'wb') as f:
                pickle.dump(preprocessor.encoders, f)

```

Figures45: data_preprocessing.py

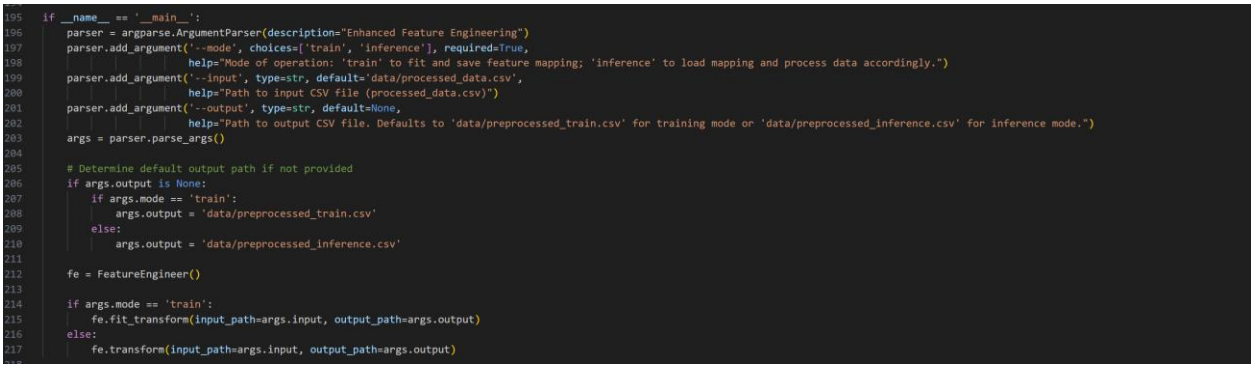
FEATURE ENGNEERING

```python
#enhanced_features.py


#!/usr/bin/env python3
import pandas as pd
import numpy as np
import pickle
from pathlib import Path
from datetime import datetime
from sklearn.preprocessing import LabelEncoder
import json
import warnings
import argparse

warnings.filterwarnings('ignore')

class FeatureEngineer:
    def __init__(self, config_path='config.json'):
        self.feature_columns = []  # Will hold the list of features used for training (without meta)
        self.encoders = {}
        self.config = self._load_config(config_path)
        self.inventory_endpoints = ['/inventory/items']
        # These are all columns that might appear in the raw data
        self.required_columns = [
            'LogID', 'Timestamp', 'UserID', 'Role', 'Endpoint', 'HTTP_Response',
            'endpoint_level_1', 'endpoint_level_2', 'endpoint_level_3',
            'hour', 'day_of_week', 'is_unusual_time', 'is_internal_ip',
            'is_authorized', 'HTTP_Method_DELETE', 'HTTP_Method_GET',
            'HTTP_Method_HEAD', 'HTTP_Method_OPTIONS', 'HTTP_Method_PATCH',
            'HTTP_Method_POST', 'HTTP_Method_PUT', 'role_risk'
        ]

    def _load_config(self, config_path):
        with open(config_path) as f:
            return json.load(f)

    def _add_temporal_features(self, df):
        """Add rolling window frequency features with proper index handling"""
        # Create temporary sorted dataframe
        sorted_df = df.sort_values(['UserID', 'Timestamp']).reset_index(drop=True)

        # Convert to datetime if needed
        if not pd.api.types.is_datetime64_any_dtype(sorted_df['Timestamp']):
            sorted_df['Timestamp'] = pd.to_datetime(sorted_df['Timestamp'])

        # Calculate 5-minute request frequency
        freq_counts = (
            sorted_df.groupby('UserID', group_keys=False)
            .rolling('5T', on='Timestamp', closed='left')['LogID']
            .count()
            .reset_index()
            .rename(columns={'LogID': 'req_freq_5min'})
        )

        # Merge back with original data
        sorted_df = sorted_df.merge(
            freq_counts[['UserID', 'Timestamp', 'req_freq_5min']],
            on=['UserID', 'Timestamp'],
```

```python
            how='left'
        )

        # Calculate inventory change rate
        inventory_mask = sorted_df['Endpoint'].str.startswith(tuple(self.inventory_endpoints))
        inv_counts = (
            sorted_df[inventory_mask]
            .groupby('UserID', group_keys=False)
            .rolling('10T', on='Timestamp', closed='left')['LogID']
            .count()
            .reset_index()
            .rename(columns={'LogID': 'inventory_change_rate'})
        )

        # Merge inventory rates
        sorted_df = sorted_df.merge(
            inv_counts[['UserID', 'Timestamp', 'inventory_change_rate']],
            on=['UserID', 'Timestamp'],
            how='left'
        )

        # Fill NaN values and restore original order
        sorted_df['req_freq_5min'] = sorted_df['req_freq_5min'].fillna(0)
        sorted_df['inventory_change_rate'] = sorted_df['inventory_change_rate'].fillna(0)

        # Restore original index and order
        return sorted_df.sort_index().drop(columns=['level_0'], errors='ignore')

    def _encode_categoricals(self, df, training=True):
        """Handle categorical encoding with consistency"""
        # Role encoding
        if training:
            self.encoders['role'] = LabelEncoder().fit(df['Role'])
        df['role_encoded'] = self.encoders['role'].transform(df['Role'])

        # Endpoint level 1 encoding
        if training:
            self.encoders['endpoint_l1'] = LabelEncoder().fit(df['endpoint_level_1'])
        df['endpoint_l1_encoded'] = self.encoders['endpoint_l1'].transform(df['endpoint_level_1'])

        return df

```

```python
    def _save_feature_mapping(self):
        """Save feature configuration to pickle file"""
        model_dir = Path('models')
        model_dir.mkdir(exist_ok=True)

        feature_config = {
            'feature_columns': self.feature_columns,  # These are the training features (used for prediction)
            'encoders': self.encoders,
            'required_columns': self.required_columns
        }

        with open(model_dir / 'features.pkl', 'wb') as f:
            pickle.dump(feature_config, f)

    def _ensure_columns(self, df):
        """Add missing columns with default values"""
        for col in self.required_columns:
            if col not in df.columns:
                df[col] = 0 if col != 'Anomalous' else np.nan
        return df

    def fit_transform(self, input_path, output_path):
        """Process training data with Anomalous column"""
        # Load and validate data
        df = pd.read_csv(input_path)
        df = self._ensure_columns(df)

        # Add temporal features
        df = self._add_temporal_features(df)

        # Encode categoricals
        df = self._encode_categoricals(df, training=True)

        # Define feature columns: drop metadata that are not used by the model.
        # We assume that the training features are all columns except 'LogID', 'Timestamp', and 'Anomalous'
        self.feature_columns = [c for c in df.columns if c not in ['LogID', 'Timestamp', 'Anomalous']]

        # Save feature mapping so that inference can use the same features
        self._save_feature_mapping()

        # Save processed training data
        df.to_csv(output_path, index=False)
        print(f"Training data processed and saved to {output_path}. Feature mapping saved.")
        return df
```

```python
    def transform(self, input_path, output_path):
        """Process inference data without Anomalous column using the frozen feature mapping,
        but keep metadata (e.g. LogID, Timestamp) in the saved CSV for traceability."""
        # Load and validate inference data
        df = pd.read_csv(input_path)
        df = self._ensure_columns(df)
        df = self._add_temporal_features(df)

        # Load the saved feature mapping (from training)
        mapping_path = 'models/features.pkl'
        try:
            with open(mapping_path, 'rb') as f:
                saved_mapping = pickle.load(f)
        except Exception as e:
            print(f"Warning: Feature mapping not found ({e}). Proceeding with current features.")
            saved_mapping = {'feature_columns': []}

        # Load the saved encoders into self.encoders
        if 'encoders' in saved_mapping:
            self.encoders = saved_mapping['encoders']

        # Now encode categoricals using the loaded encoders
        df = self._encode_categoricals(df, training=False)

        # Define the expected features (these are used for model prediction)
        expected_features = saved_mapping.get('feature_columns', [])
        if not expected_features:
            print("No saved feature columns found. Using all processed columns except metadata.")
            expected_features = [c for c in df.columns if c not in ['LogID', 'Timestamp', 'Anomalous']]
        else:
            # Add any missing expected features with default value 0
            for feature in expected_features:
                if feature not in df.columns:
                    df[feature] = 0

        # Create a copy of the full processed data (including metadata)
        df_full = df.copy()

        # Extract the feature subset that the model expects
        df_features = df_full[expected_features]

        meta_cols = ['LogID', 'Timestamp']
        # Save the full DataFrame (which includes both metadata and features)
        df_full.to_csv(output_path, index=False)
        print(f"Inference data processed and saved to {output_path}.")

        return df_full  # Full data contains metadata; model predictions should use df_features separately.
```



```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Enhanced Feature Engineering")
    parser.add_argument('--mode', choices=['train', 'inference'], required=True,
                        help="Mode of operation: 'train' to fit and save feature mapping; 'inference' to load mapping and process data accordingly.")
    parser.add_argument('--input', type=str, default='data/processed_data.csv',
                        help="Path to input CSV file (processed_data.csv)")
    parser.add_argument('--output', type=str, default=None,
                        help="Path to output CSV file. Defaults to 'data/preprocessed_train.csv' for training mode or 'data/preprocessed_inference.csv' for inference mode.")
    args = parser.parse_args()

    # Determine default output path if not provided
    if args.output is None:
        if args.mode == 'train':
            args.output = 'data/preprocessed_train.csv'
        else:
            args.output = 'data/preprocessed_inference.csv'

    fe = FeatureEngineer()

    if args.mode == 'train':
        fe.fit_transform(input_path=args.input, output_path=args.output)
    else:
        fe.transform(input_path=args.input, output_path=args.output)
```

Figures46: enhanced_features.py

PIPELINE

```python
#!/usr/bin/env python3
"""
auto_inference.py

This script automates:
1. Generating exactly 200 logs with a small chance of anomalies (once in ~20 runs).
2. Appending the new logs to a global master log archive.
3. Preprocessing (inference mode) the newly generated logs only.
4. Enhanced feature engineering (inference mode).
5. Loading a pre-trained Random Forest (from `final_models`) and predicting anomalies.
6. Saving the final predictions in a timestamped CSV for analytics.

Usage:
  python3 auto_inference.py

No command-line arguments are strictly required, but you could add your own to control:
 - The chance of anomalies
 - The location of the final model, etc.

Author: You
"""

import subprocess
import random
import datetime
import os
import pandas as pd
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import numpy as np
import joblib


# Paths and constants
LOG_GENERATOR_CMD = [
    "python3", "main.py",
    "--total_logs", "200",
    "--master_file", "data/master_logs.csv",
    "--inference_file", "data/inference_logs.csv",
    "--overwrite"
]
ARCHIVE_MASTER_FILE = "data/master_logs_archive.csv"  # big file where we store all past logs
PROCESSED_PICKLE = "data/processed_data.pkl"
PROCESSED_CSV = "data/processed_data.csv"
PREPROCESSED_INFERENCE = "data/preprocessed_inference.csv"
MODEL_FEATURES_PKL = "final_models/features.pkl"
MODEL_PATH = "final_models/random_forest.pkl"
SCALER_PATH = "final_models/Original/random_forest/scaler.pkl"
RESULTS_DIR = "data"  # Where final inference results go
```

```python
def pick_anomaly_ratio(run_count=1):
    """
    Decide an anomaly ratio.
    - In 1 out of 20 runs, use 0.01 or 0.02.
    - Otherwise, use 0.0 (no anomalies).
    """
    # If run_count % 20 == 0: or use random approach:
    # E.g. 1 in 20 chance:
    if random.randint(1, 20) == 1:
        return random.choice([0.01, 0.02])
    else:
        return 0.0

def generate_logs(anomaly_ratio):
    """
    Calls main.py to generate logs with the given anomaly_ratio,
    overwriting data/master_logs.csv and data/inference_logs.csv.
    """
    cmd = LOG_GENERATOR_CMD.copy()
    # Insert or update anomaly ratio after "--total_logs" param
    # If your main.py supports `--anomaly_ratio`, set it here:
    cmd.insert(4, "--anomaly_ratio")
    cmd.insert(5, str(anomaly_ratio))


    print("Generating 200 logs with anomaly_ratio =", anomaly_ratio)
    subprocess.run(cmd, check=True)

def append_to_master_archive():
    """
    Appends newly generated data/master_logs.csv to a global archive CSV
    so we keep a big history. Ignores the header after the first time.
    """
    if not os.path.exists(ARCHIVE_MASTER_FILE):
        # Just copy the current master_logs.csv as the initial archive
        subprocess.run(["cp", "data/master_logs.csv", ARCHIVE_MASTER_FILE], check=True)
        print(f"Created new archive {ARCHIVE_MASTER_FILE}")
    else:
        # Append (skip header)
        with open(ARCHIVE_MASTER_FILE, "a") as f_archive, open("data/master_logs.csv", "r") as f_new:
            lines = f_new.readlines()
            # Skip header (first line)
            f_archive.writelines(lines[1:])
        print("Appended newly generated logs to", ARCHIVE_MASTER_FILE)
```

```
    # 3. Extract the subset for model features
    X_infer = df_infer_full[feature_cols].copy()

    # 4. Load the Random Forest model + scaler
    # rf_model = pickle.load(open(MODEL_PATH, "rb"))
    rf_model = joblib.load(MODEL_PATH)
    # scaler = pickle.load(open(SCALER_PATH, "rb"))
    scaler = joblib.load(SCALER_PATH)

    # The train.py script standard-scaled these numeric columns:
    numeric_cols = ["req_freq_5min", "inventory_change_rate", "role_risk"]
    # We do the same transformation:
    X_infer[numeric_cols] = scaler.transform(X_infer[numeric_cols])

    # 5. Predict
    y_pred = rf_model.predict(X_infer)
    # Probability of being anomaly (class=1)
    if hasattr(rf_model, "predict_proba"):
        y_proba = rf_model.predict_proba(X_infer)[:, 1]
    else:
        # for e.g. SVM decision_function or logistic decision_function
        y_proba = rf_model.decision_function(X_infer)

    # 6. Combine results
    df_infer_full["predicted_anomaly"] = y_pred
    df_infer_full["anomaly_score"] = y_proba
    override_mask = (
        df_infer_full["Role"].isin(["Doctor","Nurse"]) &
        df_infer_full["Endpoint"].isin([
            "/patient/appointments/confirm",
            "/patient/appointments/cancel"
        ])
    )

    # Force predicted_anomaly=0 for those logs
    df_infer_full.loc[override_mask, "predicted_anomaly"] = 0
    # Optionally set the anomaly_score=0 for them too
    df_infer_full.loc[override_mask, "anomaly_score"] = 0.0

    # [F] Save the results
    now_str = datetime.datetime.now().strftime("%Y%m%d%H%M")
    results_file = os.path.join(
        RESULTS_DIR, f"inference_results_{now_str}.csv"
    )
    df_infer_full.to_csv(results_file, index=False)
    print("Saved inference results to", results_file)

    # [G] Print final stats
    final_anomaly_count = df_infer_full["predicted_anomaly"].sum()
    print(f"Predicted {final_anomaly_count} anomalies out of {len(df_infer_full)} logs (after override).")
```

```
def run_data_preprocessing():
    """
    Runs data_preprocessing.py in inference mode on data/inference_logs.csv,
    producing data/processed_data.csv and data/processed_data.pkl
    """
    cmd = [
        "python3", "data_preprocessing.py",
        "--mode", "inference",
        "--config", "config.json",
        "--input", "data/inference_logs.csv",
        "--output", PROCESSED_PICKLE
    ]
    print("Running data_preprocessing in inference mode...")
    subprocess.run(cmd, check=True)

def run_enhanced_features():
    """
    Runs enhanced_features.py in inference mode on data/processed_data.csv,
    producing data/preprocessed_inference.csv
    """
    cmd = [
        "python3", "enhanced_features.py",
        "--mode", "inference",
        "--input", PROCESSED_CSV,  # data_preprocessing creates a .csv with same base name as the pkl
        "--output", PREPROCESSED_INFERENCE
    ]
    print("Running enhanced_features in inference mode...")
    subprocess.run(cmd, check=True)

def load_model_and_predict():
    """
    Loads the random_forest model and the feature config,
    then runs predictions on data/preprocessed_inference.csv
    Output is saved to a timestamped file in data/inference_results_YYYYMMDDHHMM.csv
    with the original columns + predicted anomaly.
    """
    # 1. Load final feature mapping (feature_columns)
    with open(MODEL_FEATURES_PKL, "rb") as f:
        feature_info = pickle.load(f)
    feature_cols = feature_info["feature_columns"]

    # 2. Load the inference CSV
    df_infer_full = pd.read_csv(PREPROCESSED_INFERENCE)

    # We'll keep these metadata columns to re-attach after predicting
    # (LogID, Timestamp, maybe Role, Endpoint, etc.)
    # In your pipeline, you only used certain columns in the model, but we want
    # to keep everything for analytics. So we'll do:
    # Note: ensure the columns exist
    meta_cols = []
    for c in ["LogID", "Timestamp", "Role", "Endpoint"]:
        if c in df_infer_full.columns:
            meta_cols.append(c)
```

```python
def main():
    # 1. Decide anomaly ratio
    # (Optional) you might keep track of run_count in a file or pass as argument
    anomaly_ratio = pick_anomaly_ratio()

    # 2. Generate logs
    generate_logs(anomaly_ratio=anomaly_ratio)

    # 3. Append to big master archive (for historical reasons)
    append_to_master_archive()

    # 4. Preprocess new 200 logs (inference mode)
    run_data_preprocessing()

    # 5. Enhanced features (inference mode)
    run_enhanced_features()

    # 6. Load model + predict
    load_model_and_predict()

    print("auto_inference pipeline complete.")

if __name__ == "__main__":
    main()
```

Figures47: auto_inference.py

DASHBOARD CODE

```python
#!/usr/bin/env python3
#dashboard.py

import streamlit as st
import pandas as pd
import altair as alt
from glob import glob
import os
import ipaddress
import datetime

########################################
# CONFIGURATIONS
########################################
ARCHIVE_FILE = "data/master_logs_archive.csv"
INFERENCE_DIR = "data"
INFERENCE_PATTERN = "inference_results_*.csv"
REFRESH_EVERY_SECONDS = 60  # HTML meta refresh
ANOMALY_COUNT_THRESHOLD = 10
ANOMALY_RATIO_THRESHOLD = 0.05

# Define unusual hour range for a hospital:
# We'll consider 10 PM (22) to 6 AM as "unusual" or "after-peak"
UNUSUAL_HOUR_START = 22  # 10 PM
UNUSUAL_HOUR_END = 6     # 6 AM

########################################
# HELPER FUNCTIONS
########################################

def is_private_ip(ip_str):
    """
    Return True if ip_str is a valid private (internal) IP, else False.
    """
    import ipaddress
    try:
        ip_obj = ipaddress.ip_address(ip_str)
        return ip_obj.is_private
    except:
        return False

def load_archive():
    """
    Load the master_logs_archive.csv which stores original logs
    (without the 'Anomalous' label, or ignoring it).
    """
    if not os.path.exists(ARCHIVE_FILE):
        return pd.DataFrame()
    return pd.read_csv(ARCHIVE_FILE)
```

```python

def load_inference_results():
    """
    Load all inference_results_*.csv, keep essential columns only,
    dropping duplicates on LogID.
    """
    files = sorted(glob(os.path.join(INFERENCE_DIR, INFERENCE_PATTERN)))
    if not files:
        return pd.DataFrame()

    df_list = []
    for f in files:
        temp = pd.read_csv(f, low_memory=False)
        df_list.append(temp)
    df_infer = pd.concat(df_list, ignore_index=True)

    # We only really need these columns from inference
    needed_cols = [
        "LogID", "predicted_anomaly", "anomaly_score",
        "Timestamp", "Role", "Endpoint"
    ]
    keep_cols = [c for c in needed_cols if c in df_infer.columns]
    df_infer = df_infer[keep_cols].drop_duplicates(subset=["LogID"], keep="last")

    # Convert Timestamp if present
    if "Timestamp" in df_infer.columns:
        df_infer["Timestamp"] = pd.to_datetime(df_infer["Timestamp"], errors="coerce")

    return df_infer

def load_merged_data():
    """
    Merge the archive logs with the inference logs on LogID.
    Then unify columns named 'Role_x'/'Role_y', 'Endpoint_x'/'Endpoint_y',
    'Timestamp_x'/'Timestamp_y', etc., into single columns.
    """
    df_arch = load_archive()
    df_infer = load_inference_results()

    if df_arch.empty:
        return df_arch
    if df_infer.empty:
        return df_arch

    # Merge on LogID
    df_merged = df_arch.merge(df_infer, on="LogID", how="left")

    # Unify columns if we have _x / _y
    possible_conflicts = ["Role", "Endpoint", "Timestamp"]
    for base_col in possible_conflicts:
        xcol = base_col + "_x"
        ycol = base_col + "_y"
        if xcol in df_merged.columns and ycol in df_merged.columns:
            df_merged[base_col] = df_merged[xcol].fillna(df_merged[ycol])
            df_merged.drop([xcol, ycol], axis=1, inplace=True)
        elif base_col not in df_merged.columns:
```

```
    # Also unify predicted_anomaly, anomaly_score if needed
    if "predicted_anomaly_x" in df_merged.columns and "predicted_anomaly_y" in df_merged.columns:
        df_merged["predicted_anomaly"] = df_merged["predicted_anomaly_x"].fillna(df_merged["predicted_anomaly_y"])
        df_merged.drop(["predicted_anomaly_x","predicted_anomaly_y"], axis=1, inplace=True)
    elif "predicted_anomaly" not in df_merged.columns:
        df_merged["predicted_anomaly"] = float('nan')

    if "anomaly_score_x" in df_merged.columns and "anomaly_score_y" in df_merged.columns:
        df_merged["anomaly_score"] = df_merged["anomaly_score_x"].fillna(df_merged["anomaly_score_y"])
        df_merged.drop(["anomaly_score_x","anomaly_score_y"], axis=1, inplace=True)
    elif "anomaly_score" not in df_merged.columns:
        df_merged["anomaly_score"] = float('nan')

    # Convert predicted_anomaly to int
    df_merged["predicted_anomaly"] = df_merged["predicted_anomaly"].fillna(0).astype(int)
    df_merged["anomaly_score"] = df_merged["anomaly_score"].fillna(0).astype(float)

    # Convert final Timestamp if we ended up with them
    if "Timestamp" in df_merged.columns:
        df_merged["Timestamp"] = pd.to_datetime(df_merged["Timestamp"], errors="coerce")

    return df_merged

########################################
# DASHBOARD MAIN
########################################
def main():
    st.set_page_config(
        page_title="Enhanced Real-Time Anomaly Dashboard (Unusual Hours Only)",
        layout="wide"
    )

    # Auto-refresh every REFRESH_EVERY_SECONDS
    st.markdown(
        f'<meta http-equiv="refresh" content="{REFRESH_EVERY_SECONDS}">',
        unsafe_allow_html=True
    )

    st.title("Real-Time Anomaly Detection Dashboard (Focusing on 10PM - 6AM)")

    df_merged = load_merged_data()
    if df_merged.empty:
        st.warning("No data found. Possibly no logs or no inference results.")
        return

    # If we do not have IP_Address, add a placeholder
    if "IP_Address" not in df_merged.columns:
        df_merged["IP_Address"] = "unknown"

    # Mark internal vs. external IP
    df_merged["is_internal_ip"] = df_merged["IP_Address"].apply(is_private_ip)

        st.warning("No data found. Possibly no logs or no inference results.")
        return

    # If we do not have IP_Address, add a placeholder
    if "IP_Address" not in df_merged.columns:
        df_merged["IP_Address"] = "unknown"

    # Mark internal vs. external IP
    df_merged["is_internal_ip"] = df_merged["IP_Address"].apply(is_private_ip)

    # If there's a Timestamp column, parse hour
    if "Timestamp" in df_merged.columns and df_merged["Timestamp"].notna().sum() > 0:
        df_merged = df_merged.sort_values("Timestamp").reset_index(drop=True)
        df_merged["Hour"] = df_merged["Timestamp"].dt.hour
        # Let's define "unusual_hour" = True if hour < 6 or hour >= 22
        df_merged["is_unusual_hour"] = (df_merged["Hour"] < UNUSUAL_HOUR_END) | (df_merged["Hour"] >= UNUSUAL_HOUR_START)
    else:
        df_merged["Hour"] = -1
        df_merged["is_unusual_hour"] = False

    # Basic stats
    total_logs = len(df_merged)
    anomaly_count = df_merged["predicted_anomaly"].sum()
    anomaly_ratio = anomaly_count / total_logs if total_logs > 0 else 0

    # Show top metrics
    col1, col2, col3 = st.columns(3)
    col1.metric("Total Logs", str(total_logs))
    col2.metric("Anomalies", str(anomaly_count))
    col3.metric("Anomaly Ratio", f"{anomaly_ratio:.2%}")

    # Alert
    if anomaly_ratio >= ANOMALY_RATIO_THRESHOLD or anomaly_count >= ANOMALY_COUNT_THRESHOLD:
        st.error(f"ALERT: High anomaly volume (count={anomaly_count}, ratio={anomaly_ratio:.2%})")
        st.markdown("""
        <audio autoplay>
            <source src="https://www.soundjay.com/buttons/beep-07.mp3" type="audio/mpeg">
        </audio>
        """, unsafe_allow_html=True)

    normal_count = total_logs - anomaly_count
    st.write(f"**Normal Logs**: {normal_count}, **Anomalous Logs**: {anomaly_count}")

    ###########################################################################
    # Existing analysis for Role, Endpoint, IP
    ###########################################################################

    # A) Role-based
    st.subheader("A) Role-based Analysis")
    if "Role" in df_merged.columns:
        # Overall usage
        role_usage = df_merged["Role"].value_counts().reset_index()
        role_usage.columns = ["Role","count_usage"]
        chart_role_usage = (
            alt.Chart(role_usage)
```

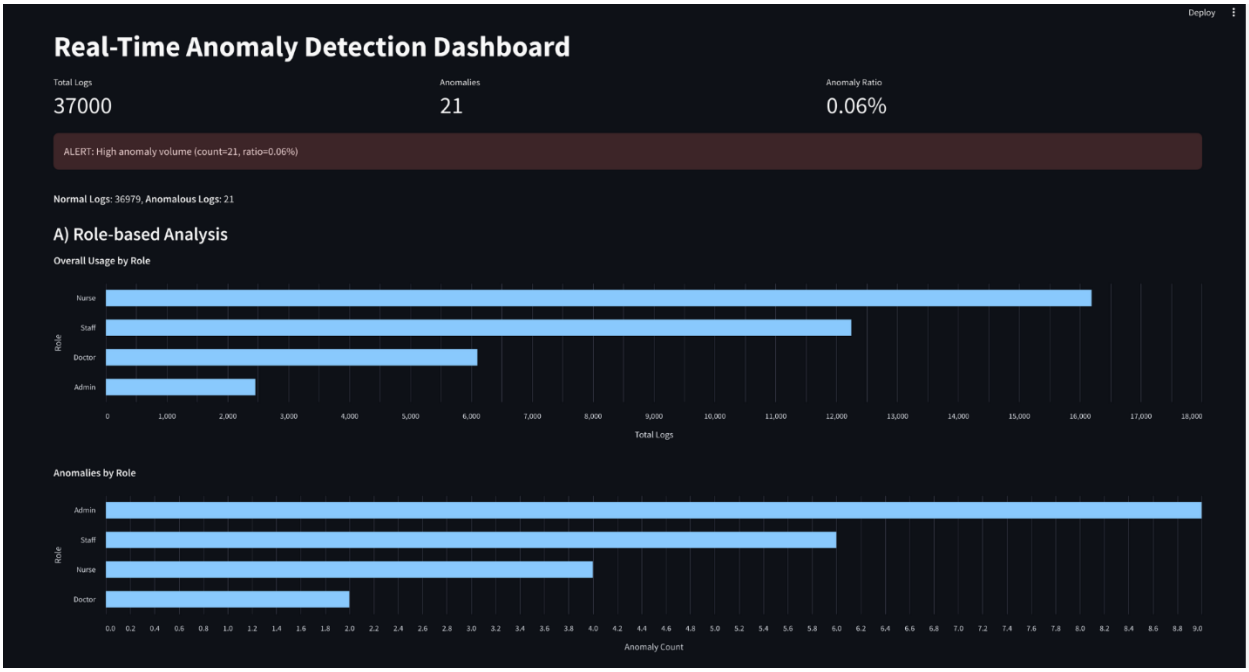
```
            alt.Chart(role_usage)
            .mark_bar(size=25)
            .encode(
                x=alt.X("count_usage:Q", title="Total Logs"),
                y=alt.Y("Role:N", sort="-x"),
                tooltip=["Role","count_usage"]
            )
            .properties(height=300, width=350, title="Overall Usage by Role")
        )
        st.altair_chart(chart_role_usage, use_container_width=True)

        # Anomalies by role
        role_anom_df = df_merged[df_merged["predicted_anomaly"]==1]
        if not role_anom_df.empty:
            role_anom = role_anom_df["Role"].value_counts().reset_index()
            role_anom.columns = ["Role","count_anomalies"]
            chart_role_anom = (
                alt.Chart(role_anom)
                .mark_bar(size=25)
                .encode(
                    x=alt.X("count_anomalies:Q", title="Anomaly Count"),
                    y=alt.Y("Role:N", sort="-x"),
                    tooltip=["Role","count_anomalies"]
                )
                .properties(height=300, width=350, title="Anomalies by Role")
            )
            st.altair_chart(chart_role_anom, use_container_width=True)
        else:
            st.write("No anomalies by Role.")
    else:
        st.write("No 'Role' column found after merging. Please confirm pipeline includes it.")

    # B) Endpoint-based
    st.subheader("B) Endpoint-based Analysis")
    if "Endpoint" in df_merged.columns:
        end_usage = df_merged["Endpoint"].value_counts().reset_index()
        end_usage.columns = ["Endpoint","count_usage"]
        end_usage = end_usage.head(10)

        chart_ep_usage = (
            alt.Chart(end_usage)
            .mark_bar(size=25)
            .encode(
                x=alt.X("count_usage:Q", title="Total Logs"),
                y=alt.Y("Endpoint:N", sort="-x"),
                tooltip=["Endpoint","count_usage"]
            )
            .properties(height=300, width=350, title="Top 10 Endpoints (Overall)")
        )
        st.altair_chart(chart_ep_usage, use_container_width=True)

        # Endpoint anomalies
        ep_anom_df = df_merged[df_merged["predicted_anomaly"]==1]
        if not ep_anom_df.empty:
            ep_anom = ep_anom_df["Endpoint"].value_counts().reset_index()
            ep_anom.columns = ["Endpoint","count_anomalies"]
            ep_anom = ep_anom.head(10)
            chart_ep_anom = (
                alt.Chart(ep_anom)
                .mark_bar(size=25)
                .encode(
                    x=alt.X("count_anomalies:Q", title="Anomaly Count"),
                    y=alt.Y("Endpoint:N", sort="-x"),
                    tooltip=["Endpoint","count_anomalies"]
                )
                .properties(height=300, width=350, title="Top 10 Endpoints (Anomalies)")
            )
            st.altair_chart(chart_ep_anom, use_container_width=True)
        else:
            st.write("No endpoint anomalies.")
    else:
        st.write("No 'Endpoint' column found after merging. Confirm pipeline includes it.")

    # C) IP-based
    st.subheader("C) IP-based Analysis (Internal vs External)")
    ip_usage = df_merged["is_internal_ip"].value_counts().reset_index()
    ip_usage.columns = ["is_internal_ip","count_usage"]
    ip_usage["IP_Type"] = ip_usage["is_internal_ip"].apply(lambda x: "Internal" if x else "External")

    chart_ip_usage = (
        alt.Chart(ip_usage)
        .mark_bar(size=25)
        .encode(
            x=alt.X("count_usage:Q", title="Total Logs"),
            y=alt.Y("IP_Type:N", sort="-x"),
            tooltip=["IP_Type","count_usage"]
        )
        .properties(height=300, width=350, title="Logs by IP Type")
    )
    st.altair_chart(chart_ip_usage, use_container_width=True)

    ip_anom_df = df_merged[df_merged["predicted_anomaly"]==1]
    if not ip_anom_df.empty:
        ip_anom_counts = ip_anom_df["is_internal_ip"].value_counts().reset_index()
        ip_anom_counts.columns = ["is_internal_ip","count_anomalies"]
        ip_anom_counts["IP_Type"] = ip_anom_counts["is_internal_ip"].apply(lambda x: "Internal" if x else "External")

        chart_ip_anom = (
            alt.Chart(ip_anom_counts)
            .mark_bar(size=25)
            .encode(
                x=alt.X("count_anomalies:Q", title="Anomaly Count"),
                y=alt.Y("IP_Type:N", sort="-x"),
                tooltip=["IP_Type","count_anomalies"]
            )
            .properties(height=300, width=350, title="Anomalies by IP Type")
        )
        st.altair_chart(chart_ip_anom, use_container_width=True)
```

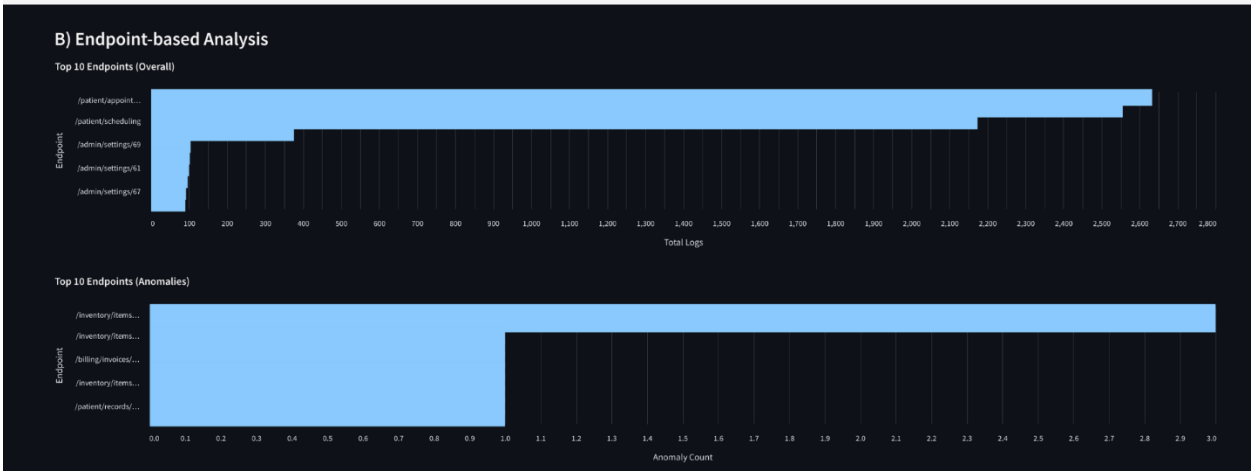
```
        st.altair_chart(chart_ip_anom, use_container_width=True)
    else:
        st.write("No IP-based anomalies found.")

# ------------------------------------------------------------------------
# D) Time-based Analysis: ONLY 10 PM - 6 AM
# Replaces your old daily_unusual_usage and daily_unusual_anom separate charts
# ------------------------------------------------------------------------

    st.subheader("D) Time-based Analysis (Unusual Hours: 10PM - 6AM only)")

    # Filter logs that occur in the unusual hour window
    df_unusual = df_merged[df_merged["is_unusual_hour"] == True].copy()

    if "Timestamp" in df_unusual.columns and df_unusual["Timestamp"].notna().sum() > 0:
        # Sort by timestamp so we have chronological order
        df_unusual = df_unusual.sort_values("Timestamp").reset_index(drop=True)

        # Create a daily grouping
        df_unusual["date"] = df_unusual["Timestamp"].dt.date

        # 1) Daily total logs (bar chart)
        daily_unusual_usage = (
            df_unusual.groupby("date")["LogID"].count().reset_index(name="count_usage")
        )

        # 2) Daily anomalies among unusual-hour logs (line chart)
        daily_unusual_anom = (
            df_unusual[df_unusual["predicted_anomaly"] == 1]
            .groupby("date")["LogID"].count()
            .reset_index(name="count_anomalies")
        )

        # Build the base domain for the X-axis
        base = alt.Chart().encode(
            x=alt.X("date:T", title="Date")
        )

        # A. The bar chart for total logs
        bar_usage = base.mark_bar(color="#4C78A8").transform_lookup(
            lookup="date",
            from_=alt.LookupData(daily_unusual_usage, "date", ["count_usage"])
        ).encode(
            y=alt.Y("count_usage:Q", title="Logs in [10PM-6AM]"),
            tooltip=[alt.Tooltip("count_usage:Q", title="Logs")]
        )

        # B. The line chart for anomalies
        line_anom = base.mark_line(point=True, color="#F58518").transform_lookup(
            lookup="date",
            from_=alt.LookupData(daily_unusual_anom, "date", ["count_anomalies"])
        ).encode(
            y=alt.Y("count_anomalies:Q", title="Anomalies in [10PM-6AM]", axis=alt.Axis(labels=True)),
            tooltip=[alt.Tooltip("count_anomalies:Q", title="Anomalies")]
        )

        # Combine them with independent y-scales
        combined_chart = alt.layer(bar_usage, line_anom, data=daily_unusual_usage).resolve_scale(
            y='independent'  # so bar usage & line anomalies each have their own scale
        ).properties(
            width=700,
            height=400,
            title="Daily Logs & Anomalies (Unusual Hours Only)"
        )

        st.altair_chart(combined_chart, use_container_width=True)

    else:
        st.write("No logs found in the 10PM-6AM window or missing valid Timestamps.")


    # Summaries for external IP anomalies & unusual-hour anomalies
    external_ip_anomalies = df_merged[
        (df_merged["predicted_anomaly"] == 1) &
        (df_merged["is_internal_ip"] == False)
    ]
    unusual_hour_anomalies = df_merged[
        (df_merged["predicted_anomaly"] == 1) &
        (df_merged["is_unusual_hour"] == True)
    ]
    st.write(f"External IP Anomalies: {len(external_ip_anomalies)}")
    st.write(f"Unusual Hour Anomalies: {len(unusual_hour_anomalies)}")

    # Expanders
    with st.expander("All Logs (Merged)"):
        st.dataframe(df_merged, use_container_width=True)

    anomalies_df = df_merged[df_merged["predicted_anomaly"] == 1]
    with st.expander("Anomalous Logs Only"):
        if not anomalies_df.empty:
            st.dataframe(anomalies_df, use_container_width=True)
        else:
            st.write("No anomalies found.")

    normal_df = df_merged[df_merged["predicted_anomaly"] == 0]
    with st.expander("Normal Logs Only"):
        st.dataframe(normal_df, use_container_width=True)

    if "anomaly_score" in df_merged.columns:
        borderline_df = df_merged[
            (df_merged["anomaly_score"] > 0.5) &
            (df_merged["predicted_anomaly"] == 0)
        ]
        with st.expander("Potentially Suspicious (High Score but Normal)"):
            if not borderline_df.empty:
                st.dataframe(borderline_df, use_container_width=True)
            else:
```

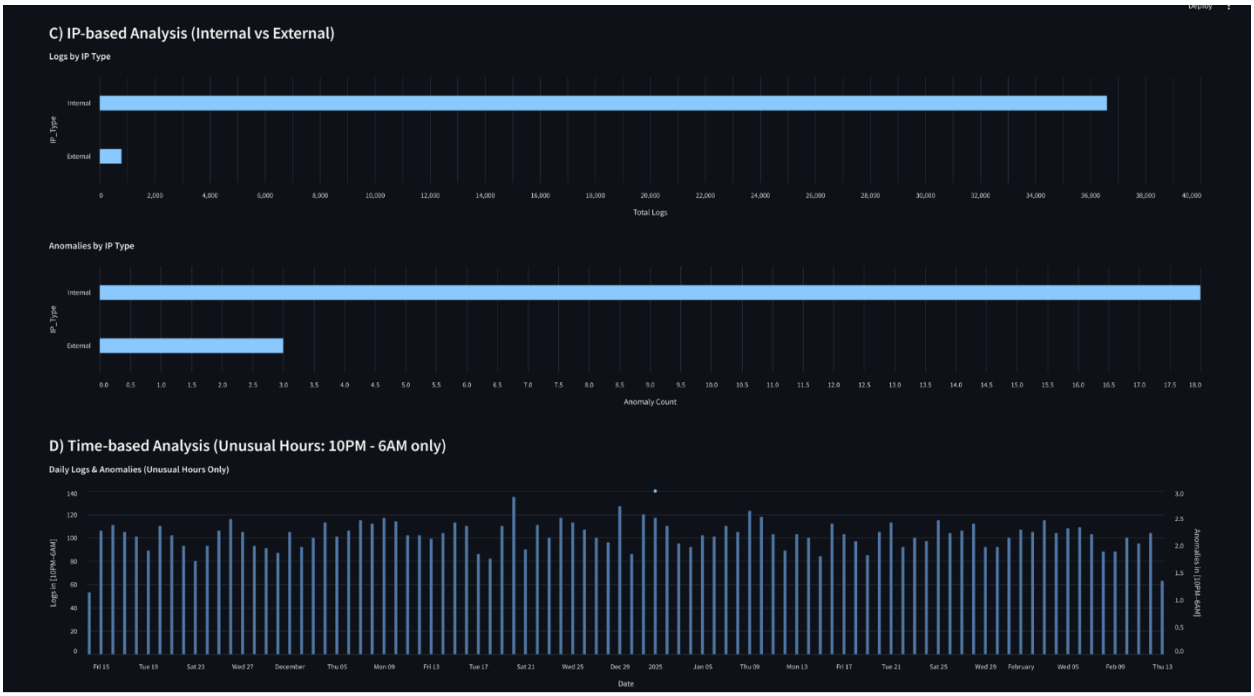
```
                st.write("No borderline logs above anomaly_score>0.5.")


if __name__ == "__main__":
    main()

```

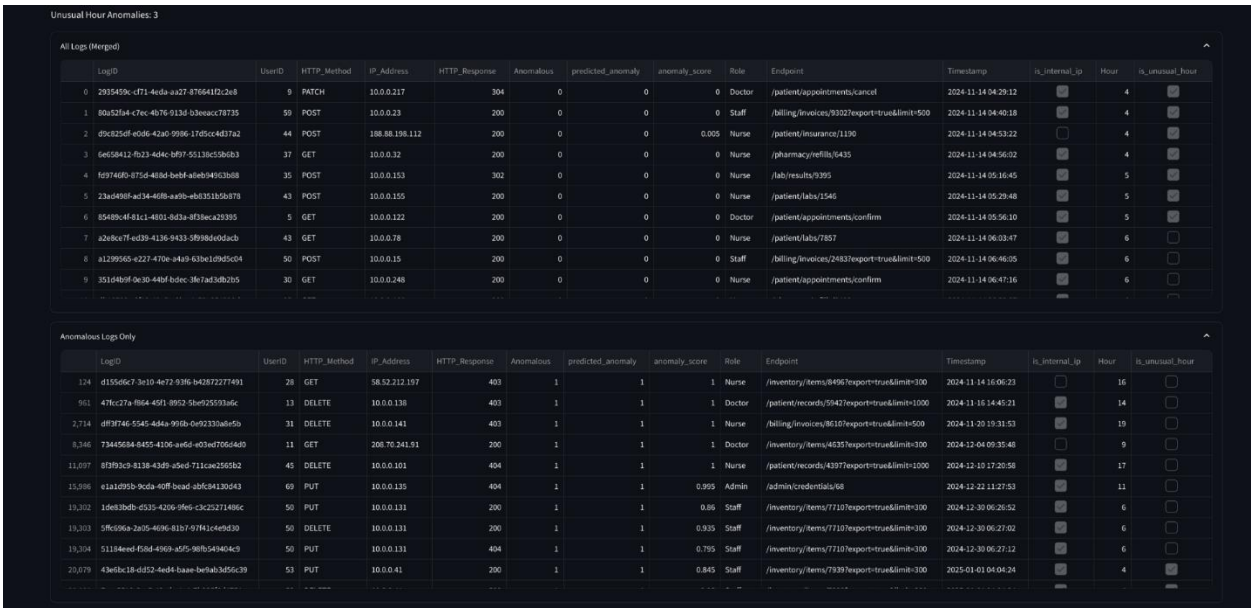
Figures48: dashboard.py

DASHBOARD


Deploy
Real-Time Anomaly Detection Dashboard
Total Logs
37000
Anomalies
21
Anomaly Ratio
0.06%
ALERT: High anomaly volume (count=21, ratio=0.06%)
Normal Logs: 36979, Anomalous Logs: 21
A) Role-based Analysis
Overall Usage by Role
Nurse
Staff
Doctor
Admin
Role
Total Logs
Anomalies by Role
Admin
Staff
Nurse
Doctor
Role
Anomaly Count


B) Endpoint-based Analysis
Top 10 Endpoints (Overall)
/patient/appoint...
/patient/scheduling
/admin/settings/69
/admin/settings/61
/admin/settings/67
Endpoint
Total Logs
Top 10 Endpoints (Anomalies)
/inventory/items...
/inventory/items...
/billing/invoices/...
/inventory/items...
/patient/records/...
Endpoint
Anomaly Count





Figures49: Final Dashboard