DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

LOGNOMALY: THE AUTOMATIC DETECTION OF LOG ANOMALIES IN SYSTEMS WITH MACHINE LEARNING

by
Fatmagül FIRTINA
Doğa Ece KOCA

Advisor
Prof. Dr. Derya BİRANT

December, 2025
İZMİR

LOGNOMALY: THE AUTOMATIC DETECTION OF LOG ANOMALIES IN SYSTEMS WITH MACHINE LEARNING

A Thesis Submitted to the
Dokuz Eylül University, Department of Computer Engineering
In Partial Fulfillment of the Requirements for the Degree of B.Sc.

by
Fatmagül FIRTINA
Doğa Ece KOCA

Advisor
Prof. Dr. Derya BİRANT

December, 2025
İZMİR

CONTENTS

Page

CHAPTER ONE

INTRODUCTION

1.1 Background Information

As the software systems have gotten more modernized, which has brought a bigger and more complex structure, the amount of log data produced by those systems has also shown a great increase that can't go unnoticed. Tasks such as tracking system health, debugging and detecting security threats rely heavily on these logs. However, the manual tracing of them has become impossible due to huge volumes they have grown into. Hence, the detection of anomalies through automated log analysis has become a critical research field for both industrial applications and academia. Effective automated log analysis systems protect system security and prevent system failures by detecting issues beforehand. Important opportunities have been presented by the developments in this field for Machine Learning and Natural Language Processing techniques to be applied to practical problems.

1.2 Problem Definition

The analysis of log data collected from current software systems carries significant importance for ensuring system health and security; however, the available solutions in this field have presented important limitations and challenges. The basic shortcomings this thesis has aimed to address are:

- The Insufficiency of Singular Approaches:

The present log anomaly detection systems in literature have generally focused on a singular approach such as only supervised, unsupervised or rule-based systems.

- ✓ Supervised Systems:

  These systems are only able to detect attack types which are labeled in the used training data; they can't catch zero-day anomalies that haven't been encountered before.

- ✓ Unsupervised Systems:

  Although these systems can catch anomalies that haven't been encountered before by learning the "normal behavior" of the system and detecting deviations, they usually label the output using a dual tag such as "normal" or "abnormal".

- Lack of Actionable Information**:**

The "abnormal" label of the current unsupervised systems, has not fulfilled the needs of a system manager or a data security analyst. These systems have yet to provide a risk score (e.g.: "dangerous with a 90% possibility") that highlights the severity of the detected anomaly resulting in alert fatigue, which occurs when analysts can't comprehend which "abnormal" alert out of hundreds to prioritize.

- Hybrid Integration Gap:

A lack of a holistic approach that unites the speed of the rule-based systems, the abilities of unsupervised learning to detect unknown anomalies and supervised learning to interpret known threats to create and assign a significant risk score into an integrated, multilayered architecture has been noticed in the literature.

This thesis has aimed at the integration problem mentioned above. The problem has been identified not only as developing an isolated system that detects only the known or unknown anomalies, but as designing a smart hybrid system architecture that presents these different abilities to the system manager as a prioritized and actionable "risk score".

1.3 Motivation

As mentioned above, the huge amounts of log data that have been produced by modern software systems have become impossible to manually trace. Hence, the detection of these anomalies has come to be a highly important research topic among both industry professionals and academic researchers. Although some approaches exist, they are yet to fulfill the needs that have emerged in this field, which has been the motivation behind this project.

Among these approaches, the system proposed by Yang et al. (2023) has presented a hybrid approach, combining rule based filtering and Isolation Forest (IF), which results in a concept that is highly similar to the objective this project has been designed to achieve. Increasing the speed by using simple rules, improving the accuracy by using IF, and producing successful results compared to other unsupervised methods have been aimed at by Yang et al's approach. However, the system hasn't been able to fill the gap in system log detection needs since it only classifies the results as "normal" or "abnormal" based on a threshold. In this thesis, however, a project that assigns a risk score to such anomalies has been intended to be developed. Yang et al's log processing steps have also lacked advanced NLP based techniques, and have relied on basic encoding methods instead. LogNomaly has used the hybrid structure of Yang et al., but it has improved upon it by adding a risk scoring layer (Layer 3) that determines the urgency of the detected anomaly, therefore producing more meaningful results.

DeepLog, developed by Du et al. (2017), on the other hand, has considered log data as a sequential language and has learned normal behaviors to flag outlier sequences as anomalies using LSTM (Long Short-Term Memory) models. It has provided an extensive detection ability by analyzing both log order and parameter values. Although these are noticeable positive traits, DeepLog, as a pure deep learning (DL) approach, has fallen short in detecting simple anomalies along with its output being classified into two categories based on whether it is within the result 'g' that the model predicted as most possible. Hence, it has been quite limited, and by its unsupervised nature, determining the anomaly's type or risk is beyond its capability. Unlike DeepLog, LogNomaly has introduced a hybrid architecture which uses fast rule-based filtering for simple anomalies (Layer 1), and has improved upon DeepLog's output classification process by determining the urgency of the anomaly using risk scoring (Layer 3).

LogRobust, presented by Zhang et al. (2019) has focused on the instability of log data, has used semantic vectorization (FastText and TF-IDF) and Attention-based Bi-LSTM to handle "unseen" logs that have been produced by log evolution or noise. This has provided robustness against changing log formats by capturing semantic similarities. However, LogRobust is a supervised approach that is trained by labeled data, and unlike

LogNomaly, it lacks an unsupervised layer that detects unknown anomalies. Also, it does not provide a hybrid risk score. LogNomaly has adopted the semantic robustness idea of LogRobust, but it has integrated that into its own hybrid and multi-layered architecture, and by the unsupervised layer (IF) that LogRobust lacks, it has provided the ability to detect completely new anomalies that do not exist in the training data.

Lastly, the experimental study conducted by Yu et al. (2024) has stated that DL methods are not superior to classical ML methods at all times, in fact, simple algorithms such as KNN can be better in both accuracy and speed. This study has questioned the necessity of DL by highlighting data leaks and unnecessary preprocessing in existing datasets. These findings have supported the hybrid design of LogNomaly, which has aimed to balance the disadvantages of pure DL or classical ML by merging a rule-based approach for simple anomalies (Layer 1), the classical but efficient unsupervised ML methods (Layer 2) to detect unknown anomalies, and supervised ML (Layer 3) to interpret the results. This study has highlighted the importance of focusing on the classical ML methods and considering the efficiency in this project.

Though these studies provide valuable contributions, none of them has succeeded in merging the speed of rule-based systems, the power of unsupervised learning to discover the unknown, and the ability of supervised learning to detect known threats and conduct a risk assessment in an integrated, multi-layered hybrid system. The fundamental contribution of the LogNomaly project has been to fill this particular gap. LogNomaly has presented an efficient and informative anomaly detection system that is more extensive than existing approaches by integrating fast rule-based filtering (Layer 1) for simple anomalies, an unsupervised Isolation Forest (Layer 2) to detect unknown anomalies, and a supervised classifier (Layer 3) to produce a risk score based on findings.

1.4 Goal/Contribution

The main purpose of this thesis has been to develop a hybrid, multi-layered log anomaly detection system prototype called LogNomaly, to automatically detect the cyber attacks and operational anomalies by analyzing the large-volume log data produced in

modern software systems. For this purpose, a new system architecture that has filled the gap in the literature has been designed.

The LogNomaly system includes a rule-based filtering layer (Layer 1) which rapidly detects known and easily detectable attacks (e.g., brute force) as the first step. Then, an unsupervised learning algorithm, Isolation Forest (Layer 2), has been employed. The aim of this layer has been to detect unknown, completely new anomalies, and behavioral anomalies that do not exist in the training data. Lastly, the system includes a supervised risk scoring layer (Layer 3) that merges the findings from previous layers and potential semantics of the logs. This layer assigns a probabilistic "risk score" to each anomaly detected (e.g., 95% probability of brute force threat) and a simple explanation of the threat, thereby providing a meaning to the findings.

The scientific and technological contribution of this study has been to provide a new and integrated hybrid approach which merges the rule-based, unsupervised, and supervised paradigms that are usually treated separately, under a single integrated system. Unlike the "normal/abnormal" output of many existing systems (e.g., Yang et al., 2023), the ability of LogNomaly to assign a probabilistic risk score and provide a possible classification of the threat or simple explanation has aimed to provide useful and valuable information to system administrators and security analysts. The system has been designed to cover a large threat spectrum that includes known attacks, unknown behavioral anomalies and probabilistic classification of known attacks. This has provided an important advance by adding the ability of discovering the unknown, that is left out by supervised systems such as LogRobust (Zhang et al., 2019). Lastly, this study has not only recommended a theoretical model or algorithm, but it also has developed a webbased working prototype of the designed architecture.

In short, the contribution of this thesis has been to present a multi-layered, hybrid architecture and it has aimed to add a probabilistic risk scoring ability to develop a working prototype system which integrates the strengths of different approaches in the literature for log anomaly detection, and balances their weaknesses.

1.5 Project Scope

In the scope of the thesis, a hybrid log anomaly detection system called LogNomaly has been developed. The system includes the implementation of three main modules. Firstly, a rule-based filtering module has been developed to detect known attack patterns. Secondly, an unsupervised learning module based on Isolation Forest has been developed to detect behavioral deviations that do not exist in the training data. Thirdly, to classify anomalies and determine their risk scores a supervised classifier based on Random Forest has been trained and integrated by using open source log datasets. The design and coding of these modules has constituted the main framework of the project.

Also, a web application has been developed where users can upload their log files and examine the anomaly detection results on a dashboard. This user interface has been developed by using C# language and ASP.NET MVC framework. On the other hand, the main log processing, model training and anomaly detection part has been developed by using Python language. The communication between .NET Core MVC and Python has been established by a REST API. To train and evaluate the system, open source and labeled log datasets such as HDFS and BGL has been used and results have been measured with standard classification metrics such as precision and F1-score.

There have been some critical points that are beyond this study's limitations. The developed prototype primarily focuses on the analysis of uploaded log files or bulk datasets, therefore the real-time processing of log flows has been excluded from the scope of the thesis. Even though the system analyzes the anomalies, classifies them and assigns a risk score, it does not include an analysis module that automatically detects their root causes, and the presented explanation is limited to predicted anomaly type or triggered rules. Processing of the logs has covered basic parsing and feature extraction methods, but has not focused on deep semantic and context analysis that are covered in studies such as LogRobust (Zhang et al., 2019). Also, designing new cyber attack scenarios, developing a mobile interface or meeting the commercial scalability requirements of the system have been excluded from the project's scope, and the designed system serves as a research prototype.

The fundamental technologies used in the development of the project have been Python and C# programming languages. On the Python side, libraries and frameworks such as Pandas, NumPy, Flask and FastAPI have been used. On the .NET side, ASP.NET Core MVC has been used. To detect anomalies, rule-based Isolation Forest and Random Forest algorithms have been applied. Docker technology has been used to pack different components of the application and ease the distribuiton. This study has focused on and proposed an anomaly detection solution to the analysis of general purpose system logs that are gathered from various digital environments such as web servers, distributed systems or supercomputers, without depending on a specific physical location.

1.6 Standards, Ethics, Constraints and Conditions

This project has been conducted around general academic integrity and software engineering principles. Certain standards, ethical rules, constraints and conditions have been taken into account during the design, development and evaluation processes of the project.

1.6.1 Standards

The software elements of the project, which utilize Python and C#, have been developed according to the best current practices of used frameworks such as ASP.NET Core. The readability, maintainability and modularity of the code have been prioritized. The performance of the developed Machine Learning models has been transparently evaluated via metrics such as Precision, Recall, F1-Score and ROC Curve, which are standards in this field. The original format of the public datasets used in the project has been respected, and the data pre-processing steps applied have been documented in detail. Although it is not a must to follow a certain log format, general principles of structured logging have been applied.

1.6.2 Ethical Rules

Considering that Log data could potentially contain sensitive data such as IP addresses and usernames, the usage of public and anonymized datasets has been prioritized. In the event of working with real data in the future, it has been recognized that ensuring data protection according to legal regulations such as KVKK and GDPR is essential. The aim

has been to avoid processing sensitive data during the prototype phase. The transparency of the decision-making mechanisms of Machine Learning models has been given importance, and a basic explainability analysis has been conducted via methods like feature importance ratings. Considering the potential effects of misinterpretation or abuse of anomaly detection systems, the principle that the system should only serve as a decision support tool with the ultimate decision being made by humans has been adopted. The potential outcomes of false positives have been considered.

1.6.3 Constraints

This work has been limited to a strict timetable due to the fact that it has been done for an undergraduate thesis. This time constraint has affected the scope of the project directly, such as the fact that the real-time analysis feature has been excluded from the model. The development and testing processes have been limited to current personal computer hardware and potential university resources; this has especially bound the process of training models on large data sets. The high computational cost required by deep learning models has been evaluated within these limitations. Due to a lack of a dedicated budget, cloud services have been implemented to the project with limits such as free tiers and low-cost options. As elaborated in Chapter 5, automatic root cause analysis along with advanced NLP has pushed out of the scope of this thesis.

1.6.4 Conditions

The success of the project has been contingent upon accessing good-quality and labeled public log datasets obtained from sources such as LogHub – it has been assumed that these sources are useable -. The project has been built on technologies like Python, .NET Core, Scikit-learn and it has been taken into consideration that potential changes and compatibility issues of these technologies might affect the work that has been done. Lastly, the progress and direction of the project have been shaped by feedback obtained from regular meetings with the advisor.

CHAPTER TWO

LITERATURE REVIEW

2.1 Related Works

2.1.1 Data Sets and Main Challenges

Zhu et al. (2023) have aimed to address the lack of standard and labeled dataset that is one of the biggest obstacles in AI-based log analysis research. In the study, a comprehensive and open source data repository called “Loghub” was created, containing more than 77 millions of log messages gathered from various systems such as HDFS, BGL, and Thunderbird. The researchers have stated that this dataset creates a common benchmark foundation for researchers, and provides a fair comparison of the performance of the developed algorithms. This study has contributed to accelerate scientific progress in the log analysis field, and objective evaluation of approaches, by standardizing the datasets used in the academic literature.

Landauer et al. (2024) have critically examined the reliability of datasets (HDFS, BGL etc.) used as standards in array-based anomaly detection studies, and their ability to represent real-world scenarios. In the study, the structural features of anomalies on these datasets has been analyzed, and the main reasons behind the high success rates achieved by modern techniques has been questioned. The researchers have stated that most of the anomalies in common datasets are actually very simple (e.g., such as emergence of a new event type), and complex deep learning models are just memorizing these simple patterns. This study has shaken the perception of “high score equals good model” in the literature, and pointed out that the researchers must pay more attention to quality of the dataset, and complexity of the anomaly when evaluating the approaches they developed.

2.1.2 Deep Learning and Modern Approaches

Meng et al. (2019) have aimed to simultaneously detect both sequential and quantitative anomalies in unstructured system logs. In the study, a method developed to represent log templates semantically called “template2vec”, and a LSTM based deep

learning architecture for modeling log flows has been used. The researchers have observed that the LogAnomaly model they have developed has achieved higher accuracy on HDFS and BGL datasets than existing supervised and unsupervised methods. This study has provided an important contribution to the literature, especially in processing unseen events better by detecting semantical relationships in log templates.

To the detect system log anomalies, Han et al. (2023) have aimed to fill the gap in standard model's deficiency in processing long arrays and inability to catch different scaled patterns. In the study, a unique transformer-based architecture called "LTAnomaly" has been developed, analyzing log data both locally and globally, and learning long term dependencies. In tests performed on standard data sets, the researchers have observed that the proposed model has a higher F1-score than existing methods, especially in resolving complex relationships in long termed log arrays. This study has proven that the integration of multi-scale representation techniques in log analysis has increased the model's distinctiveness and long term memory capacity significantly.

Xie et al. (2021) have focused on the problem of traditional sequential models overlooking the complex and nonlinear structural relationships between log events. In the study, an architecture called "LogGD" has been proposed, modeling log data as graphs representing relationships between events, and using Graph Neural Networks (GNN) to analyze these structures. The researchers have reported that this graph-based approach has performed higher performance than standard LSTM or CNN models, particularly in detecting simultaneous anomalies in complex events. This study has contributed a new approach to the literature, by proving that not only the temporal order but also the structural relationships between events is a critical information source in log analysis.

Based on the problem of traditional deep learning models' inability to comprehend complex semantical relationships and context, Yang and Harris (2023) have researched LLM's potential in this field. In the study, fine tuning has been made on log data and a transformer-based productive log anomaly detection mechanism has been developed based on LLaMA architecture. The researchers have observed that the developed LogLLaMa model has produced more successful results in interpreting unseen log

templates and complex attack scenerios than previous LSTM or CNN based models. This study has contributed an innovative perspective in semantic log analysis to the literature by showing the adaptation of pre-trained generative AI models in cyber security.

Pospieszny et al. (2023) have aimed to develop an adaptive and unsupervised anomaly detection method in system environments where static rules are insufficient and constantly changing. In the study, based on transformer architecture with the aim of learning contextual and syntactic structures, the ADALog system using self-attention and Masked Language Model techniques has been presented. The researchers have reported that this model overperforms the traditional RNN or LSTM based approaches in catching long term dependencies between logs, and adapting to dynamic log flows. This study has presented to the literature that how advanced language models in the NLP field (such as BERT variations) can be transferred to semantic analysis of system logs and anomaly detection problems successfully.

Pan et al. (2024) have aimed to solve the problem of "hallucination" (generating unreal knowledge) that LLMs sometimes produce, and equip the model with institutional memory. In the study, the RAGLog system based on Retrieval Augmented Generation (RAG) has been developed which feeds the generative AI with related and similar examples taken from historical log database. The researchers have reported that this method has based the model's decisions on concrete past records, and significantly improved the detection accuracy in comparison with standard LLM approaches. This study has provided an up to date and innovative contribution to the literature by showing how LLM's can be strengthened by dynamic knowledge bases without being trapped in static training knowledges.

2.1.3 Log Parsing Problem

Khan et al. (2022) have experimentally analyzed the effect of log parsing process' quality and accuracy on the final performance of deep learning based anomaly detection models. In the study, a comprehensive impact analysis has been made by using four different log parsers and three different deep learning models that are popular in the literature. The researchers have observed that the mistakes done in the parsing step has

lowered the anomaly detection accuracy (F1-Score) significantly, and incorrect parsing has sabotaged the model's learning process. This study has drawed attention to preprocessing quality that is usually overlooked in the literature, and proven that for successful anomaly detection not only the model but also the log parsing strategy must be chosen wisely.

Le and Zhang (2021) have aimed to get rid of the log parsing step which is the most fragile link in traditional anomaly detection processes and the information loss caused by it. In the study, instead of parsing log messages into strict templates, an innovative architecture called "NeuralLog" that uses directly raw text and using BERT based vector representations has been proposed. The researchers have reported that this approach is not effected by parsing errors, and has surpassed parsing based models by demonstrating a high generalization ability to varying log formats. This study, has contributed a parsing free and semantic based new analysis standard to the literature by proving that structural parsing is not necessary in log analysis.

Almodovar et al. (2024) have developed the "LogFiT" approach to reduce dependency of traditional anomaly detection methods to parsing process, and benefit from the language model's semantic abilites. In the study, pre-trained language models are put to fine-tuning process to learn unique structure and syntax of system logs. The researchers have reported that this methods processes the logs as a natural language flow therefore reaching high accuracy scores and also resistant to structural changes. This study has contributed the importance of linguistic context in anomaly detection to the literature by proving that Transfer Learning techniques can be successfully integrated into cyber security field.

2.1.4 A Critical Examination: Is Deep Learning Always Necessary?

Le and Zhang (2022) have questioned the generalizability and real performance gains of deep learning models which have become the standard in log anomaly detection in recent years, compared to traditional methods. They have conducted a comprehensive comparison by reapplying popular deep learning models from the literature alongside classical machine learning algorithms on multiple datasets. The researchers have found

that complex deep learning models have been unexpectedly vulnerable to noise in training data and have not performed significantly better than classical machine learning models in most scenarios. This study has provided a critical viewpoint that encourages evaluating simpler and more stable methods before incurring high computational costs and challenges the assumption that "deep learning is always better."

Chen et al. (2021) have aimed to evaluate the industrial applicability and performance limits of deep learning–based log analysis methods. In their study, four prominent deep learning models from the literature (e.g., DeepLog, LogAnomaly) and traditional approaches have been compared on large-scale, publicly available datasets. The researchers have observed that although deep learning models have achieved higher detection success rates, they have also possessed significant disadvantages in terms of computational cost and sensitivity to training noise. This report has provided a roadmap for practical applications by emphasizing that efficiency and appropriate model selection are critically important alongside raw accuracy.

Yang et al. (2023) have aimed to re-evaluate the potential of statistical methods which are much simpler than today's widespread deep learning models, by questioning how necessary these complex architectures truly are. They have optimized the PCA method for log anomaly detection and have compared it to complex deep learning models in the literature. The researchers have shown that their improved PCA method has had considerably lower computational cost than algorithms such as DeepLog or LogAnomaly, while offering competitive and sometimes superior performance. This study has demonstrated that more complex models do not always yield better results in log analysis and has encouraged researchers to adopt lightweight and practical approaches.

Ying et al. (2021) have introduced an improved anomaly detection method designed to address the lack of labeled data required by supervised learning algorithms and to reduce the computational cost of the KNN algorithm. They have used a fast variation of the KNN algorithm optimized for anomaly detection and a clustering-based automatic labeling mechanism to eliminate the need for manual labeling. The researchers have shown that the proposed method has achieved highly competitive accuracy with much

shorter training times than complex deep learning models. This study has proven that high-cost deep learning models are not always necessary in the log analysis process and has contributed to a practical solution to reduce computational overhead.

Keyogeg et al. (2024) have aimed to develop a system that automatically detects ransomware targeting Windows Active Directory services which is a critical component of institutional networks. In their study, they have analyzed attack signatures and abnormal access patterns using machine learning algorithms such as Random Forest on log data from the AD environment. Researchers have observed that the proposed approach has detected the preparatory phases of ransomware attacks before the encryption process with high success rates while minimizing false alarms. This study has emphasized the importance of integrating log analysis and machine learning in protecting critical infrastructures and has contributed to the development of a proactive security mechanism against ransomware.

2.1.5 Hybrid Architecture and Explainable AI (XAI)

Wang et al. (2024) have presented a hybrid framework that combines Deep Learning and traditional methods to detect complex security breaches and anomalies in computer networks. The study has integrated Isolation Forest to detect unknown threats, GANs (Generative Adversarial Networks) to address data imbalance, and a transformer architecture for time-series analysis. The researchers have demonstrated that the multi-layered structure they have developed achieves higher detection performance and resilience particularly in complex network traffic data compared to single-focused models. This study has further established that the combined use of different artificial intelligence paradigms (unsupervised learning and deep learning) offers effective solutions to modern cybersecurity challenges.

Yu et al. (2024) have developed a multistage approach called “LogMS” based on multi-source information fusion to overcome noise in log data and hardships presented by labeling ambiguities. In the study, a “probabilistic label prediction” mechanism to confirm suspicious labels was used along with a hybrid structure that combines both the statistical and semantic features of logs. Researchers have reported that this multistage

approach has increased the amount of correct anomaly detection rates in noisy datasets and decreased false alarms significantly. This study has shown to the literature that not only model architecture, but strategies towards increasing data quality and label reliability should also be focused on during anomaly detection.

Brown et al. (2018) have aimed to develop an interpretable model that could overcome the "black box" structure of Deep Learning models and explain the reasonings behind log anomalies. The study has integrated RNN to learn the temporal relationships in log sequences, along with Attention Mechanisms to determine the log events the model focuses on while making a decision. Researchers have observed that this approach not only provides a high anomaly detection rate, but also visual insights to system managers regarding the source of the anomaly. This study has produced a great contribution which strengthens the human-machine cooperations in cyber-security operations and increases reliability by bringing transparency to complex Deep Learning models.

Han et al. (2021) have focused on the issue of security analysts not being able to securely use Deep Learning-based anomaly detection algorithms although they have high success rates due to their "black box" structure. In the Study, a system called "DeepAID", which outputs a cause-effect reasoning for every anomaly detected and provides analysts with interpretable proofs has been developed. Researchers have proven that when integrated with the current Deep Learning models, this approach shortens the investigation time of analysts and performs a better classification for unknown threat types. This study has shown that purely high accuracy rates are not enough on their own for cyber security applications, and interpretability is an indispensable element for system usability.

Zou and Petrosian (2020) have made it a goal for themselves to solve the transparency and security issues caused by the "black box" structure of machine learning-based complex anomaly detection models. In the study, The Shapley Value, which is based on the game theory and is used to calculate the marginal contribution of every value, has been utilized as an explainer. Researchers have shown that this approach acquired a high success rate for making sense of the root causes of detected anomalies and made the

decision-making process of the model more understandable for humans. This study has made it clear that the detection success isn't the only criteria for security in critical systems, and it has provided the literature with the Shapley values' mathematical strength of interpreting complex models.

2.2 Comparison with the Existing Solutions

The work that has been examined in this literature review has shown that Deep Learning-based studies (DeepLog, LogAnomaly, LogRobust, etc.) are overpowering in the field of log anomaly detection. However, these current solutions have three basic deficiencies mentioned below in terms of applicability in industrial fields:

- Computational Cost and Complexity:

  LSTM-based approaches, such as those proposed by Meng et al. (2019) and Du et al. (2017), along with template-based models, require a high computational cost power for training and inference processes although they reach high accuracy rates. Empirical studies done by Chen et al. (2021) and Le & Zhang (2022) have proven that these complex models do not always perform significantly better compared to basic machine learning algorithms, and are sensitive to noisy data. In order to overcome this problem, LogNomaly has utilized a hybrid and lightweight structure that combines rule-based filtering, Isolation Forest and Random Forest algorithms instead of heavy and complex Deep Learning architectures. This approach has aimed at providing similar accuracy rates with significantly less resource consumption.

- Explainability:

  Most of the current studies have been functioning as a "black box" and are yet to provide the analyst with an explanation why a certain log is abnormal. Han et al. (2021) and Brown et al. (2018) have emphasized that model transparency is critical for security operations. LogNomaly has integrated the SHAP approach to fill this gap in the literature and made it possible for the reasoning behind every risk-score to be explained by a visualized metric.

- Output Variety:

The standard datasets (HDFS, BGL) and models in the literature generally present the output as a binary classification such as “Normal” or “Abnormal”. However, in operational processes, analysts need a rating to understand the severity of the anomaly. LogNomaly has gone beyond this binary classification to overcome this deficiency and produced a probability-based risk score between the values 0 and 1 along with classifying the anomaly between certain types such as Brute Force or System Failure.

The comparative analysis of the suggested LogNomaly project with other prominent studies in literature is shown in Table 2.1.

Table 2.1 Comparison of LogNomaly with leading methods in existing literature.

Study/Method	**Core Algorithm**	**Output Type**	**Explainability (XAI)**	**Computational Cost**
DeepLog (Du et al., 2017)	Deep Learning (LSTM)	Binary (Normal/Abnormal)	Low (Black Box)	High
LogAnomaly (Meng et al., 2019)	Deep Learning (Template2Vec)	Binary (Normal/Abnormal)	Low	High
LogRobust (Zhang et al., 2019)	Deep Learning (Attention-based)	Binary (Normal/Abnormal)	Medium (via Attention)	High
PCA / KNN (Ying et al., 2021)	Statistical / Traditional ML	Binary	Medium	Low
LogNomaly (Proposed)	**Hybrid (Rule-based + IF + RF)**	**Risk Score + Class**	**High (SHAP-supported)**	**Low / Medium**

CHAPTER THREE

REQUIREMENTS/REQUIREMENT ENGINEERING

This chapter details the technical specifications of the LogNomaly system. The requirements are defined in detail to guide the implementation phase, ensuring that inputs, processing logic, exception handling, and expected outputs are clearly described for every module. This section serves as a blueprint for the coding phase.

3.1 Functional Requirements

Functional requirements define the specific behaviors and operational functions of the LogNomaly system. Each requirement is structured to include input parameters, the processing logic (including algorithms and validations), and the expected output states.

3.1.1 Log Data Ingestion and Parsing (Module 1)

This module acts as the entry point for the system, responsible for handling raw unstructured data and converting it into a structured format suitable for machine learning analysis.

3.1.1.1 FR-01: Log File Upload

- Description:

The system must allow users to upload log files via the web interface.

- Input:

File object (Extensions*: .log, .txt, .csv*; Max Size: 50MB).

- Process / Pseudo-code:
    1. User selects a file via *FileUploader* widget.
    2. System validates file extension: *if file.ext not in ['.log', '.txt','.csv'] -> Raise ValidationError.*
    3. System validates file size: *if file.size > 50MB -> Raise SizeLimitError.*
    4. Generate a unique *SessionID* and save the file to a temporary secure directory: */temp/uploads/{SessionID}/*.

- Output:

HTTP 200 OK with *file_path* reference or *HTTP 400 Bad Request* with error message.

3.1.1.2 FR-02: Log Parsing and Normalization

- Description:

The system must parse raw log lines into structured fields (Timestamp, Level, Component, Content) using Regex or template matching.

- Input:

Raw text lines from the uploaded file.

- Process / Pseudo-code:
    1. Load file content line by line.
    2. For each *line* in *file*:
        - Apply Regex Pattern (e.g., *^(\d{4}-\d{2}-\d{2})\s+(\w+)\s+\[(.*?)\]\s+(.*)$*).
        - *if match:* Extract groups -> *{Date, Level, Service, Message}*.
        - *else:* Mark line as "Unparseable" and log a warning.
    3. Convert Date string to *DateTime* object.
    4. Normalize *Level* (e.g., map "WARN", "Warning" -> "WARNING").
- Output:

A structured List of Dictionaries or a Pandas DataFrame.

- Exception Handling:

If >20% of lines are unparseable, abort process and notify user of "Invalid Log Format".

An example log data and expected output after parsing operation is shown in Table 3.1 below.

Table 3.1 Example log data and expected parsing output.

Data Type	Expected Content
Input (raw log)	2025-01-20 14:30:15 ERROR [AuthService] Failed login attempt from IP 192.168.1.5
Parsing Logic	Regex:^(\d{4}-\d{2}-\d{2})\s+(\d{2}:\d{2}:\d{2})\s+(\w+)\s+\[(.*?)\]\s+(.*)$
Output (structured)	**Date:** 2025-01-20
	Time: 14:30:15
	Level: ERROR
	Service: AuthService
	Message: Failed login attempt.

3.1.1.3 FR-03: Feature Extraction

- Description:

The system must convert text data into numerical vectors for ML models.

- Input:

Parsed Pandas DataFrame.

- Process / Pseudo-code:
    1. Label Encoding: Convert categorical *Level* (INFO=0, ERROR=1) to integers.
    2. TF-IDF / CountVectorization:
        - Initialize *TfidfVectorizer(max_features=100).*
        - Fit and transform the *Message* column.
    3. Time Delta Calculation: Calculate time difference (Δt) between consecutive logs.
    4. Construct *FeatureMatrix X* combining encoded columns and vectors.
- Output:

A serialized Feature Matrix (NumPy array) ready for model inference.

3.1.2 Hybrid Anomaly Detection (Module 2)

This module constitutes the core intelligence of the system, employing a multi-layered approach to detect anomalies.

3.1.2.1 FR-04: Layer 1 - Rule-Based Filtering

- Description:

Rapid detection of known simple threats using a predefined blacklist/rule engine.

- Input:

Structured Log Entry.

- Process / Pseudo-code:
    1. Load *RuleDictionary* (e.g., {"Pattern": "failed password", "Threshold": 5 in 1 min}).
    2. For each log entry:
        - Check if *Message* contains any signature in *RuleDictionary*.
        - *if match*: Set *IsKnownThreat = True*, *ThreatType = [MatchedRuleName]*.
        - *else:* Set *IsKnownThreat = False.*
- Output:

Boolean Flag *IsKnownThreat*.

3.1.2.2 FR-05: Layer 2 - Unsupervised Detection (Isolation Forest)

- Description:

Detects unknown (zero-day) anomalies by identifying outliers in the feature space.

- Input:

Feature Matrix (X) from FR-03.

- Process / Pseudo-code:
    1. Load pre-trained Isolation Forest model: *model = load('iso_forest.joblib')*.
    2. Execute prediction: *anomaly_score = model.decision_function(X).*
    3. Execute classification: *prediction = model.predict(X)* (Returns -1 for Anomaly, 1 for Normal).
    4. *if prediction == -1*: Send to Layer 3.
    5. *else*: Mark as "Normal".

- Output:

Binary classification (Normal/Potential Anomaly) and Raw Anomaly Score.

3.1.2.3 FR-06: Layer 3 - Supervised Classification (Random Forest)

- Description:

Classifies the specific type of anomaly if Layer 2 flags it as suspicious.

- Input:

Feature Vector of the potential anomaly.

- Process / Pseudo-code:
    1. Load pre-trained Random Forest Classifier: *rf_model = load('rf_classifier.joblib')*.
    2. Execute probability prediction: *probs = rf_model.predict_proba(X).*
    3. Identify class with max probability: *predicted_class = argmax(probs).*
    4. Extract confidence score: *confidence = max(probs).*
- Output:

PredictedClass (e.g., Brute Force, SQLi) and *ConfidenceScore*.

3.1.3 Risk Scoring and Prioritization (Module 3)

This module synthesizes outputs from detection layers to assign a unified risk score.

3.1.3.1 FR-07: Probabilistic Risk Scoring

- Description:

Calculates a final Risk Score (0.0 - 1.0) based on the hybrid analysis.

- Input:

Outputs from FR-04 (Rule), FR-05 (IF Score), FR-06 (RF Probability).

- Process / Pseudo-code:
    1. *if* Layer 1 (Rule) is True:
        - *FinalRiskScore = 1.0* (Critical).

2. *else if* Layer 2 (IF) is Normal:
    - *FinalRiskScore = 0.0* to *0.2* (Based on IF score normalization).
3. *else* (Layer 2 is Anomaly):
    - *FinalRiskScore = (w1 * Normalized_IF_Score) + (w2 * RF_Confidence)*
    - (Where *w1* and *w2* are weighting factors, e.g., 0.4 and 0.6).

- Output:

Float value *FinalRiskScore.*

3.1.3.2 FR-08: Dynamic Thresholding & Alerting

- Description:

Categorizes the risk score into actionable levels.

- Input:

FinalRiskScore.

- Process / Pseudo-code:
    1. *if score < 0.50:* Level = **Low** (Green).
    2. *if 0.50 ≤ score < 0.80*: Level = **Medium** (Orange)
    3. *if score ≥ 0.80*: Level = **High/Critical** (Red).
- Output:

Enum Value *RiskLevel.*

Risk score intervals, levels and corresponding actions is shown in Table 3.2.

Table 3.2 Risk score levels and decision matrix.

Risk Score Interval	**Risk Level**	**Example Situation**	**Required Action**
0.00 - 0.49	Low (Safe)	Routine logs, known bugs, or standard operations.	**Log Only:** Save to database without alerting.
0.50 - 0.79	Medium (Suspicious)	Rare event sequences, failed login attempts, or policy violations.	**Review Recommended:** Flag for analyst review on the dashboard.
0.80 - 1.00	High / Critical	Known attack signatures (e.g., SQLi), brute force, or system crashes.	**Immediate Alert:** Trigger high-priority alarm and email notification.

3.1.4 Explainable AI (XAI) Reporting (Module 4)

3.1.4.1 FR-09: SHAP Value Generation

- Description:

Generates mathematical explanations for why a specific log was flagged as high risk.

- Input:

The specific log's Feature Vector and the Random Forest Model.

- Process / Pseudo-code:
    1. Initialize Explainer: *explainer = shap.TreeExplainer(rf_model).*
    2. Calculate values: *shap_values = explainer.shap_values(feature_vector).*
    3. Map values to feature names: *explanation = zip(feature_names, shap_values).*
    4. Sort by absolute contribution descending.
- Output:

JSON object containing Top-5 features influencing the decision (e.g., *{"Source_IP": +0.45, "Time_Hour": -0.12}*).

3.1.5 Interactive Dashboard Visualization

3.1.5.1 FR-10: Dashboard Rendering

- Description:

Provides the user interface to visualize results.

- Input:

Analysis Results from Database.

- Process / Pseudo-code:
    1. **Overview API:** Query DB for *count(TotalLogs), count(Anomalies), avg(RiskScore)*.
    2. **Chart API:** Query DB for *RiskLevel* distribution (Group By Level).
    3. **Frontend:** Render Chart.js or D3.js graphs based on API JSON response.

- Output:

Interactive HTML/JS Dashboard.

3.2 Non-Functional Requirements

Non-functional requirements specify the quality attributes, performance constraints, and security standards of the system.

3.2.1 Performance Efficiency

- NFR-01 (API Latency):

The Python Backend API must respond to single-log analysis requests within **200 milliseconds** to ensure a responsive user experience.

- NFR-02 (Throughput):

The system must be capable of processing a batch upload of **10,000 log lines** within **60 seconds** on the reference hardware (minimum 4 vCPU, 8GB RAM).

- NFR-03 (UI Rendering):

The Dashboard page Load Time (LCP - Largest Contentful Paint) must be under **2.5 seconds**.

Aimed performance metrics are given in Table 3.3.

Table 3.3 Key performance indicators (KPIs).

Metric	Aimed Value	Explanation
API Response Time	< 500 ms	The maximum latency for the Python Backend to return an analysis result for a single log entry.
Loading Dashboard	< 3 sec	The total time required for the web interface to fully render graphs and tables.
Model Accuracy	F1-Score ≥ %80	The minimum acceptable performance threshold for the hybrid anomaly detection model.

3.2.2 Reliability and Robustness

- NFR-04 (Fault Tolerance):

If the parsing module encounters a malformed log line, it must skip that specific line, log an internal error, and continue processing the rest of the file (Fail-Safe).

- NFR-05 (Availability):

The system web server should maintain a theoretical uptime of **99.5%** during business hours.

3.2.3 Data Security and Privacy

- NFR-06 (Data Privacy):

The system must not permanently store uploaded raw log files after analysis. Raw files must be deleted from the server temp storage immediately after parsing is complete.

- NFR-07 (Secure Transmission):

All data transmission between the Client (Browser), Frontend (.NET), and Backend (Python) must be encrypted using **TLS 1.2/1.3** (HTTPS).

- NFR-08 (Input Validation):

All file uploads must be validated against MIME types to prevent malicious file execution (e.g., preventing .exe uploads disguised as .log).

3.2.4 Scalability

- NFR-09 (Modular Design):

The Machine Learning engine must be decoupled from the Web Interface via REST API, allowing the ML component to be scaled horizontally (e.g., multiple Docker containers) independently of the frontend.

- NFR-10 (File Support):

The system must support processing of log files up to **500 MB** in size using stream processing (reading line-by-line) to prevent Memory Overflow (OOM) errors.

3.2.5 Interoperability

- NFR-11 (API Standards):

All inter-module communication must use strictly typed **JSON** format following RESTful principles (GET, POST methods).

- NFR-12 (Browser Compatibility):

The web dashboard must be fully functional on current versions of major browsers (Chrome, Firefox, Edge, Safari) without requiring third-party plugins.

CHAPTER FOUR

DESIGN

4.1 Architectural View

Figure 4.1 illustrates the high-level architecture of the LogNomaly system. The system is essentially composed of three main blocks: **Preprocessing, the Hybrid Model Core, and the Web Application/XAI Layer.**

- Preprocessing Layer:

  Raw log data is collected, structured through the Log Parsing module, and transformed into numerical feature vectors via Feature Extraction.

- Hybrid Model (LogNomaly Core):

  This layer represents the brain of the system. The data passes through three sequential stages: first, known simple threats are filtered out using rule-based filtering (Layer 1). Next, unknown anomalies (outliers) are detected using Isolation Forest (Layer 2). Finally, these anomalies are classified using Random Forest (Layer 3), and a risk score is assigned.

- Web Application and XAI:

  The risk scores generated by the Python-based backend and the explainability data produced by the SHAP engine are transmitted to the ASP.NET Core interface via a REST API and presented to the user.

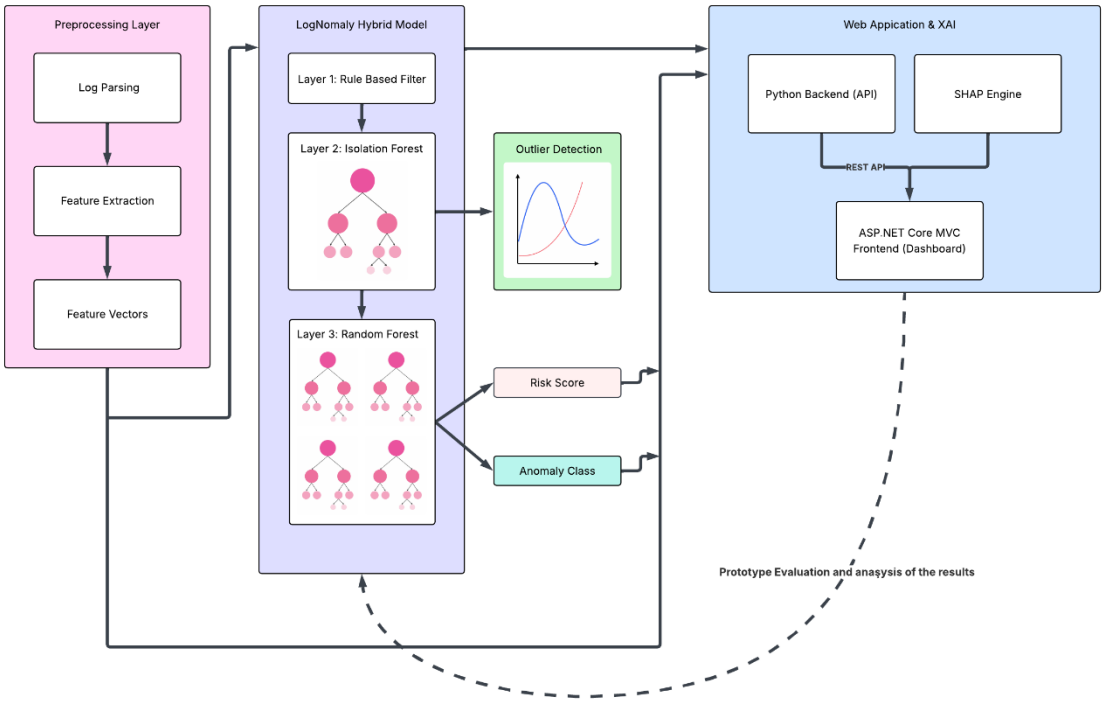

Figure 4.1 Architectural view.

4.2 Database Design/ER Diagram

The data management design of the system is implemented using an in-memory session-based structure rather than a persistent database, and is presented in Figure 4.2. At the core of this design are the *LogFile* structure, which represents uploaded files, and the *LogEntry* structure, which represents each individual log line within those files.

- *UserSession:*

Stores user session information in memory; each file upload operation is associated with a specific session.

- *LogEntry:*

Holds parsed log data (Time, Content, Level) in an in-memory structure.

- *AnalysisResult:*

This is the most critical structure in the system. It has a one-to-one relationship with *LogEntry*. For each log line, the calculated *RiskScore*, the detected *AnomalyType*, and the *SHAP_Values* generated for explainability (stored in JSON format) are kept in session memory.

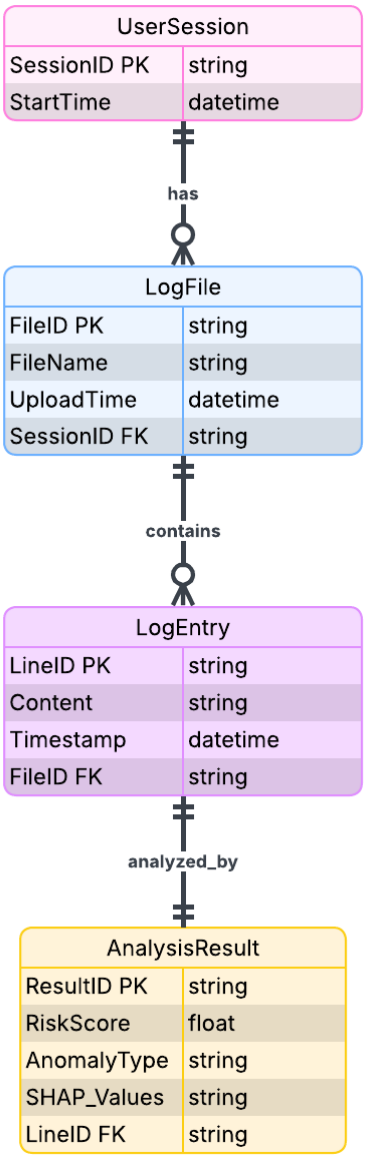

Figure 4.2 LogNomaly ER diagram.

4.3 UML Class Diagram

The class diagram illustrates the static structure of the LogNomaly backend system, detailing the relationships between the log processing utilities, machine learning models, and the main control logic. The design adheres to Object-Oriented Programming (OOP) principles to ensure modularity, maintainability, and scalability.

Figure 4.3 depicts the class hierarchy and relationships. The system architecture is built around a central *HybridDetector* class that acts as a facade, orchestrating the interaction between data parsing, anomaly detection models, and the explanation engine.

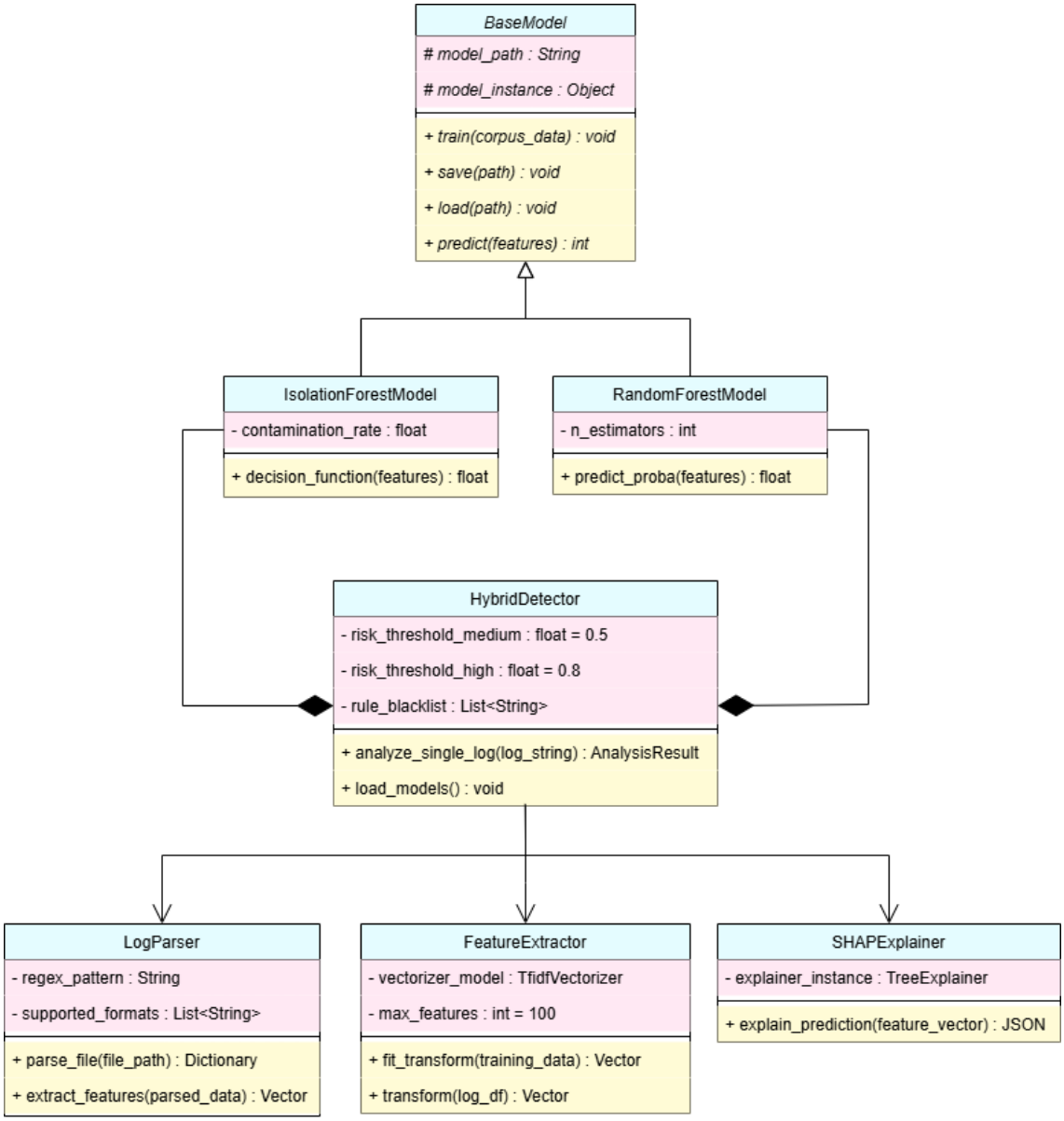

Figure 4.3 LogNomaly class diagram.

4.3.1 Class Descriptions and Responsibilities

The core components of the system are defined as follows:

- Data Processing Layer:

  ✓ *LogParser:*

  This utility class is responsible for transforming unstructured raw text into structured data. It handles file I/O operations and applies regex patterns to extract fields (Timestamp, Level, Message).

- Key Method: *parse_file(file_path):* Reads a file and returns a Pandas DataFrame.

✓ *FeatureExtractor:*

Handles the transformation of structured text into numerical vectors required by ML algorithms. It manages vectorizers (e.g., TF-IDF) and label encoders.

- Key Method: *transform(log_df):* Converts text data into a feature matrix (X).

- Machine Learning Abstraction (Polymorphism):

✓ *BaseModel* (Abstract Interface):

Defines the standard contract that all machine learning models must follow. This interface ensures that if a new model (e.g., Deep Learning) is added in the future, it will fit seamlessly into the system.

- Methods: *train(), predict(), save(), load().*

✓ *IsolationForestModel:*

A concrete implementation of *BaseModel* specialized for unsupervised outlier detection. It implements the logic for detecting unknown anomalies.

- Unique Method: *decision_function():* Returns the raw anomaly score.

✓ *RandomForestModel:*

A concrete implementation of *BaseModel* used for supervised classification of known threat types.

- Unique Method: *predict_proba():* Returns the confidence score (probability) of the predicted class.

- Core Logic and Controller:

✓ *HybridDetector:*

The primary controller class that composes the entire analysis pipeline. It instantiates the parser and models, executes the logic flow defined in the

Functional Requirements (Layer 1 -> Layer 2 -> Layer 3), and aggregates the final Risk Score.

- Relationship: It has a **composition** relationship with *IsolationForestModel* and *RandomForestModel* (it "owns" these instances).
- Key Method: *analyze_single_log(raw_log_string):* Executes the full pipeline for a single entry.

✓ *XAI_Explainer:*

A specialized class dedicated to interpretability. It uses the SHAP library to calculate feature importance values for high-risk predictions.

- Key Method: *explain_prediction(feature_vector):* Returns a JSON object with feature contributions.

4.3.2 Design Patterns Applied

- Strategy Pattern:

Used in the *BaseModel* hierarchy, allowing the system to switch between different algorithmic strategies (Unsupervised vs. Supervised) using a common interface.

- Facade Pattern:

The *HybridDetector* class acts as a facade, providing a simplified interface (*analyze()*) to the API Controller, hiding the complexity of the underlying parsing and multi-stage modeling processes.

4.4 UI Design

Figure 4.4 presents the **Overview Dashboard** interface that users encounter upon logging into the system. This screen is designed to summarize the overall health status of the uploaded logs at a glance:

- KPI Cards:

The cards displayed at the top provide real-time situational awareness for administrators by showing the total number of processed logs, the number of detected anomalies, and the system's average risk score.

- Risk Distribution:

The donut chart on the left illustrates the proportional distribution of processed events categorized as **Normal, Suspicious, and Critical**.

- Time Series Analysis:

The bar chart on the right visualizes the temporal density of anomalies, making it easier to identify periods with increased attack attempts.

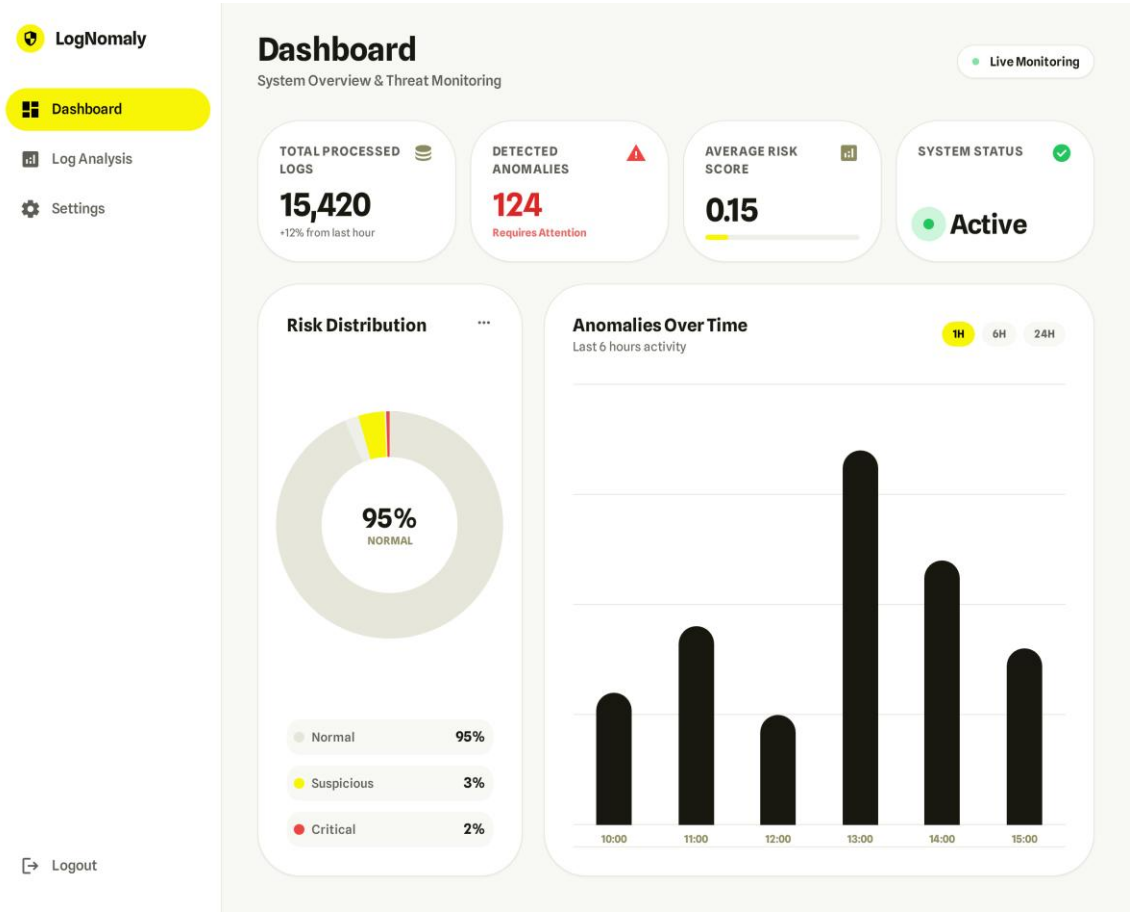

Figure 4.4 LogNomaly dashboard view.

Figure 4.5 presents the **Log Analysis and Upload** interface, where analysts submit data and review the analysis results. Designed in accordance with usability principles, this screen includes the following features:

- Data Ingestion:

The upload area located at the top of the page enables seamless integration of log files into the system using a drag-and-drop mechanism.

- Intelligent Filtering:

The tabular structure categorizes the threats detected by the hybrid model (e.g., Brute Force, SQL Injection) according to their types and risk levels.

- Visual Prioritization:

Color coding applied in the **Risk Score** column (Red: Critical, Yellow: Medium, Green: Low) allows security analysts to quickly focus on the records that require the most urgent attention among thousands of log entries.

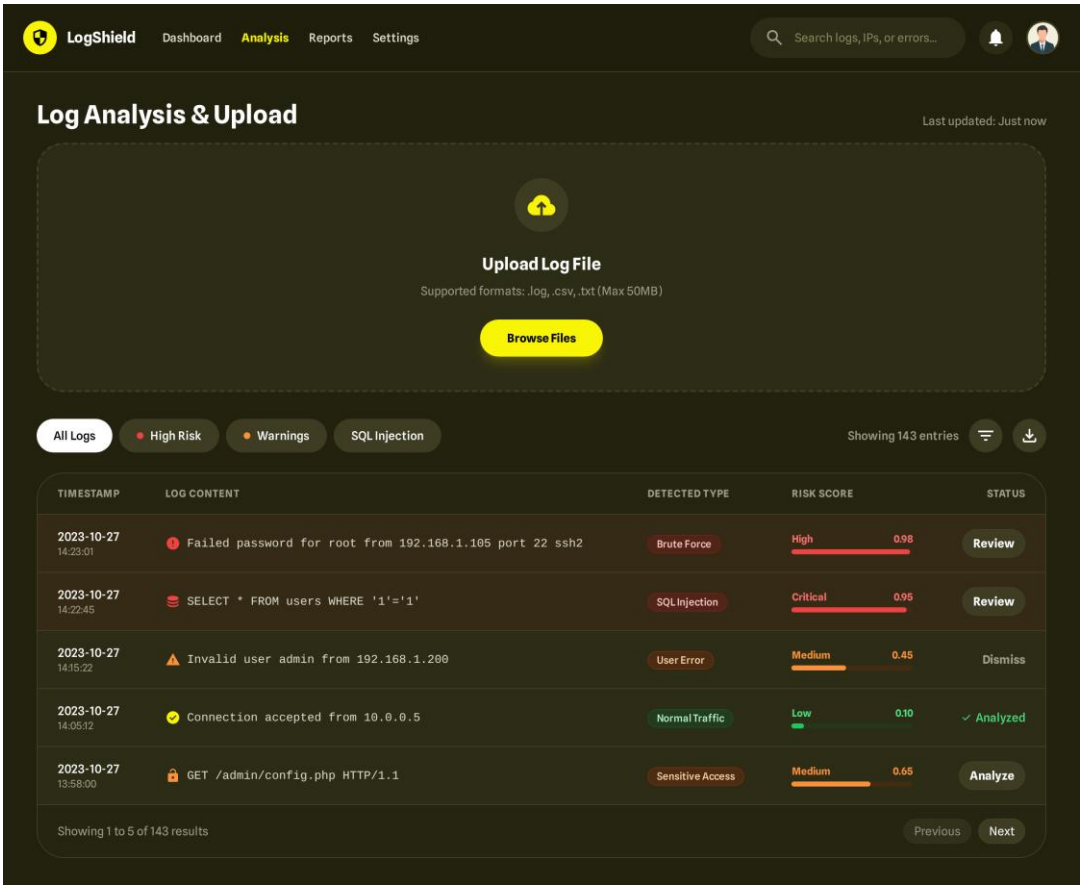

Figure 4.5 LogNomaly log analysis & upload view.

Figure 4.6 illustrates the **Explainable Threat Analysis (XAI)** screen, which represents one of the most significant contributions of the LogNomaly project to the literature. When an analyst clicks on a suspicious record, a detailed view is opened that includes the following components:

- Raw Data (Raw Payload):

  The original JSON format of the log is presented, enabling in-depth technical inspection.

- AI Insight:

  A natural language–generated summary is provided to explain why the system made its decision (e.g., *"The IP address has not been previously observed in the dataset."*).

- SHAP Visualization:

  The chart located at the bottom right visualizes the feature contributions that influenced the machine learning model's decision. By transparently showing how much each feature (e.g., *Source IP* or *Event Type*) increases the risk score, the system effectively addresses the "black box" problem.

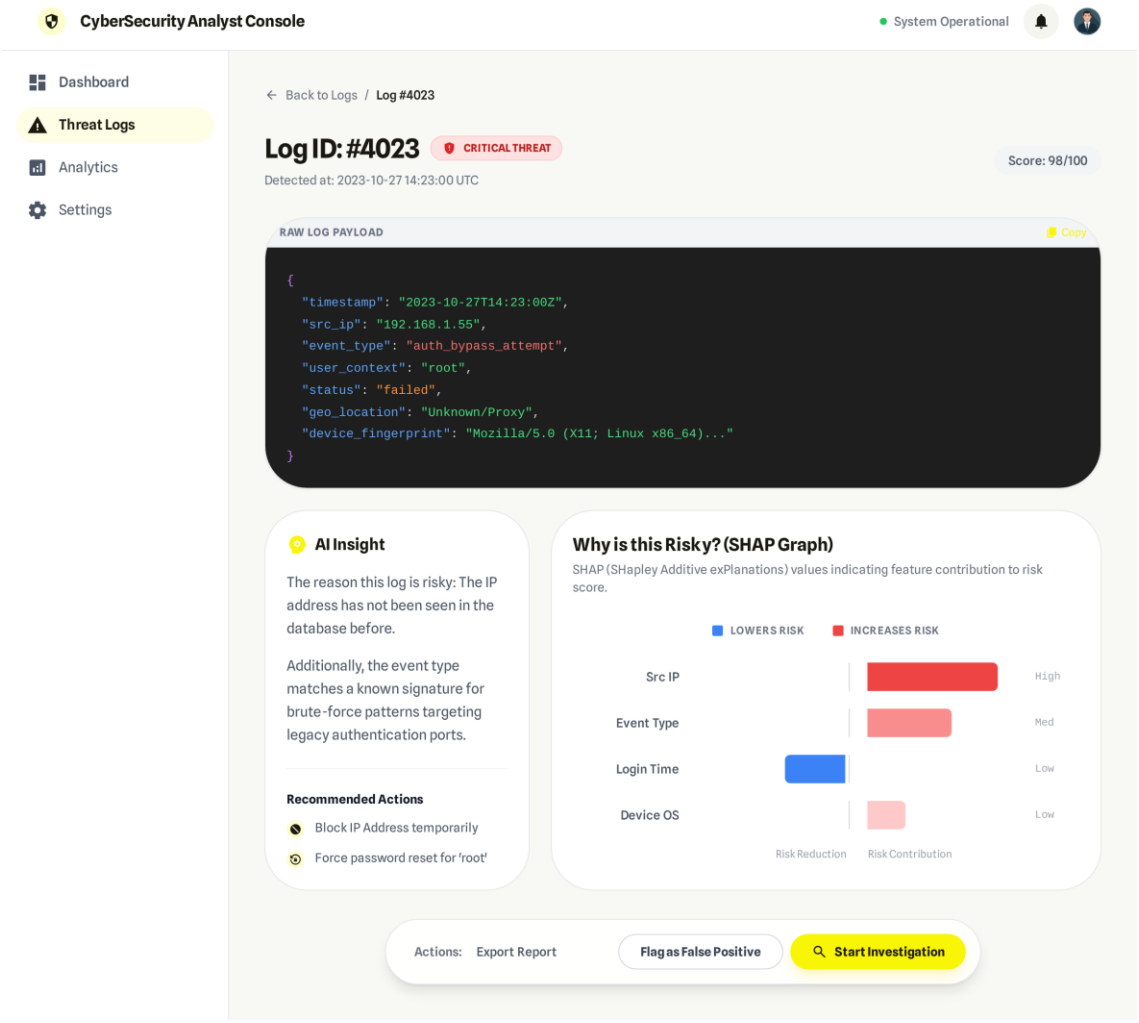

Figure 4.6 LogNomaly XAI view.

Figure 4.7 presents an example of the **Incident Report**, which enables the findings obtained at the end of the analysis processes to be transferred into organizational knowledge. This module includes the following components:

- Executive Summary:

Provides a high-level overview of the system's overall status, the number of prevented threats, and system uptime for senior management, without overwhelming them with technical details.

- Threat Vectors:

Displays the distribution of detected attacks by type, highlighting the areas in which the organization is most vulnerable.

- Recommendation System:

Serves as a decision-support mechanism by offering automated action recommendations for detected critical vulnerabilities (e.g., *"Patch the SQL server"* or *"Inspect the IP address"*).

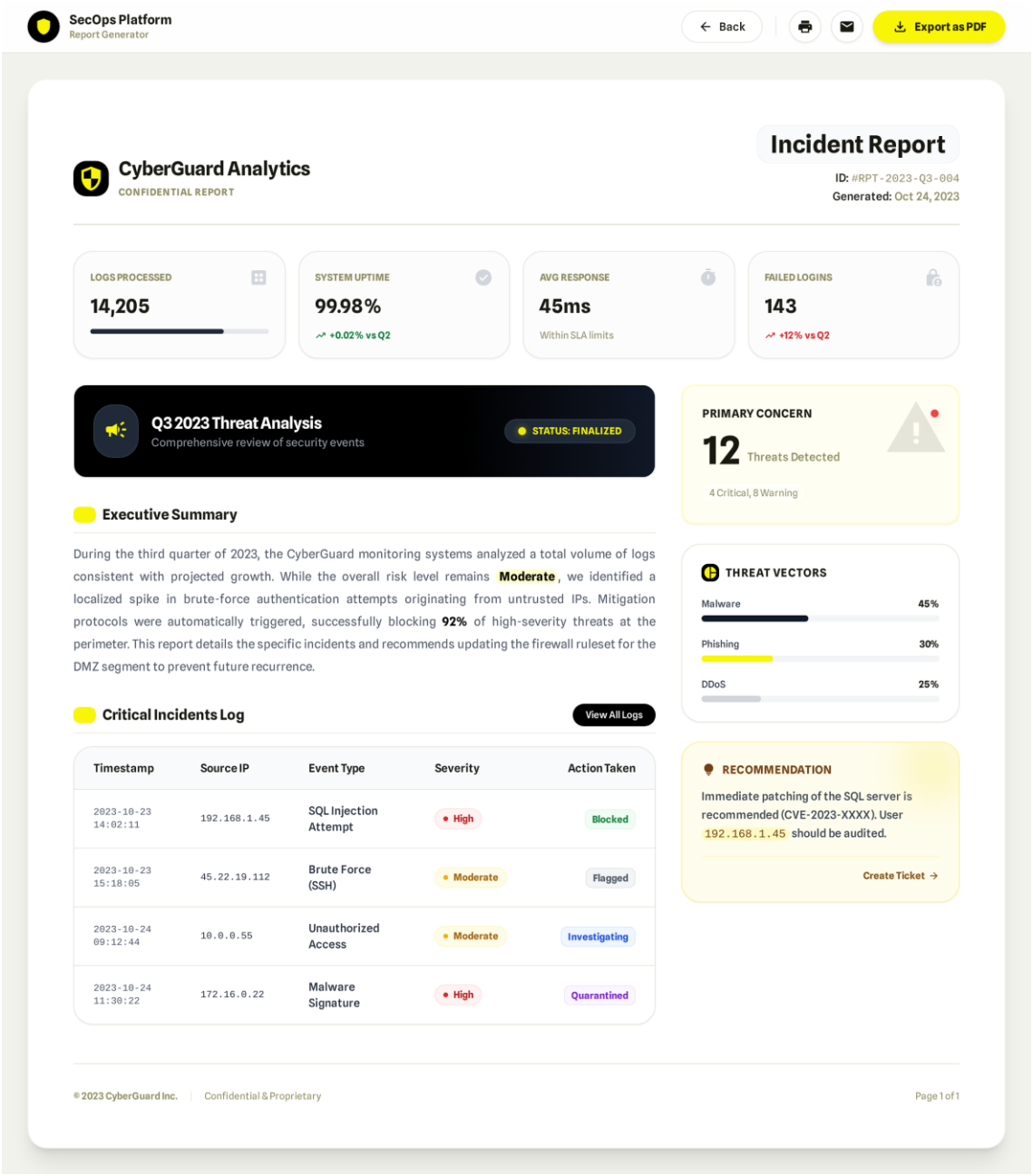

Figure 4.7 LogNomaly incident report view.

4.5 Use Cases

The Use Case diagram depicts the functional interactions between the actors (users) and the LogNomaly system. It defines the system's boundaries and illustrates the primary goals that a user can achieve using the application.

Figure 4.5 illustrates the primary use cases. The system is designed around a single primary actor, the **Security Analyst**, who interacts with the web interface to perform analysis and review results.

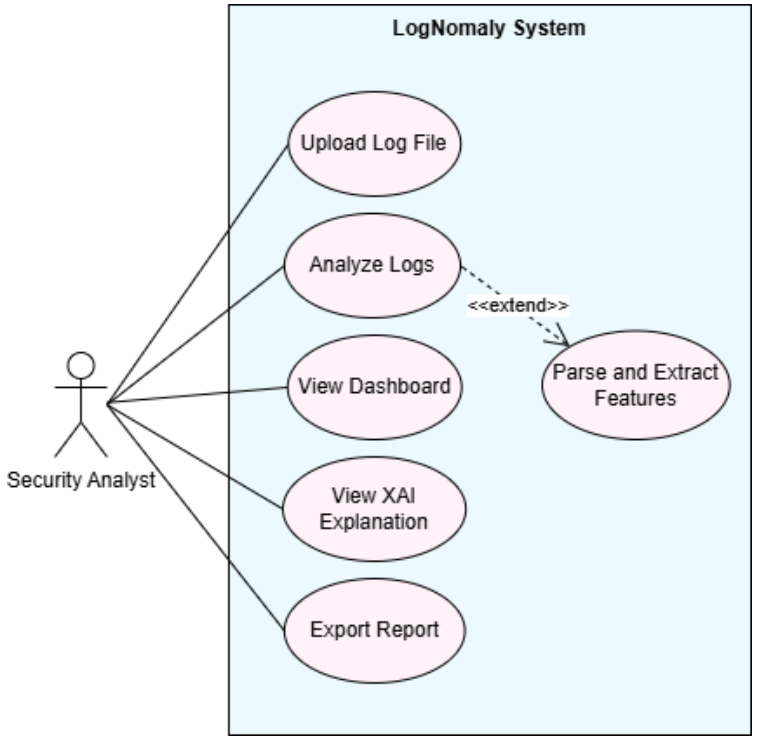

Figure 4.8 LogNomaly system use case diagram.

4.5.1 Actors

- Security Analyst:

The primary user of the system. This actor is responsible for uploading log files, initiating the analysis process, interpreting the risk scores, and generating incident reports.

4.5.2 Use Case Descriptions

The core functionalities shown in the diagram are detailed below:

- UC-01: Upload Log File:

  ✓ Description:

  The analyst selects and uploads a raw log file (.log, .txt) to the system.

  ✓ Pre-condition:

  User must be logged into the web interface.

  ✓ Post-condition:

  File is validated, saved to temporary storage, and ready for parsing.

  ✓ Exceptions:

  If the file format is invalid, the system displays an error message.

- UC-02: Analyze Logs (Hybrid Detection):

  ✓ Description:

  This is the central function where the system processes the uploaded file through the hybrid model (Rule-based + Isolation Forest + Random Forest).

  ✓ Includes:

  This use case automatically includes the "Parse Logs" and "Extract Features" sub-processes.

  ✓ Post-condition:

  Anomalies are detected, risk scores are calculated, and results are stored in the database.

- UC-03: View Dashboard & Visualization:

  ✓ Description:

  The analyst views the overview dashboard, which displays charts for risk distribution, anomaly counts over time, and system health status.

✓ Trigger:

Successful completion of the analysis or clicking the "Dashboard" tab.

- UC-04: View XAI Explanation (Inspect Threat):

✓ Description:

The analyst clicks on a specific high-risk log entry to view the detailed explanation.

✓ Extensions:

This use case extends the "View Dashboard" use case. It displays the SHAP force plot showing why the model flagged that specific log.

- UC-05: Filter and Search:

✓ Description:

The analyst filters the results based on specific criteria (e.g., "Show only Critical Risks" or "Show logs from IP 192.168.1.5").

- UC-06: Export Incident Report:

✓ Description:

The analyst generates and downloads a PDF summary report of the detected anomalies for documentation purposes.

4.6 Sequence Diagram

Figure 4.9 details the time-based execution of an analysis scenario performed on the LogNomaly system and the inter-component messaging flow. The analysis process follows the steps below:

1. User Interaction:

The workflow begins when the end user uploads a log file through the frontend interface.

2. API Request:

The frontend sends the file to the Python-based **BackendAPI** service for analysis via a *POST /analyze* request.

3. Data Preparation:

The BackendAPI invokes the *parse()* method of the **LogParser** component to structure the incoming raw log data and transform it into a format suitable for machine learning.

4. Hybrid Analysis and XAI:

The parsed data is forwarded to the **HybridModel** component for anomaly detection using the *predict()* method. As illustrated in the diagram, while the model layer performs the prediction, it simultaneously triggers the **SHAPExplainer** component via the *explain()* method to generate explanations for the results.

5. Result Response:

The risk scores and explainability data are consolidated and returned to the frontend in JSON format, where they are presented to the user. This sequence diagram demonstrates how the system's modular architecture operates in a synchronized manner.

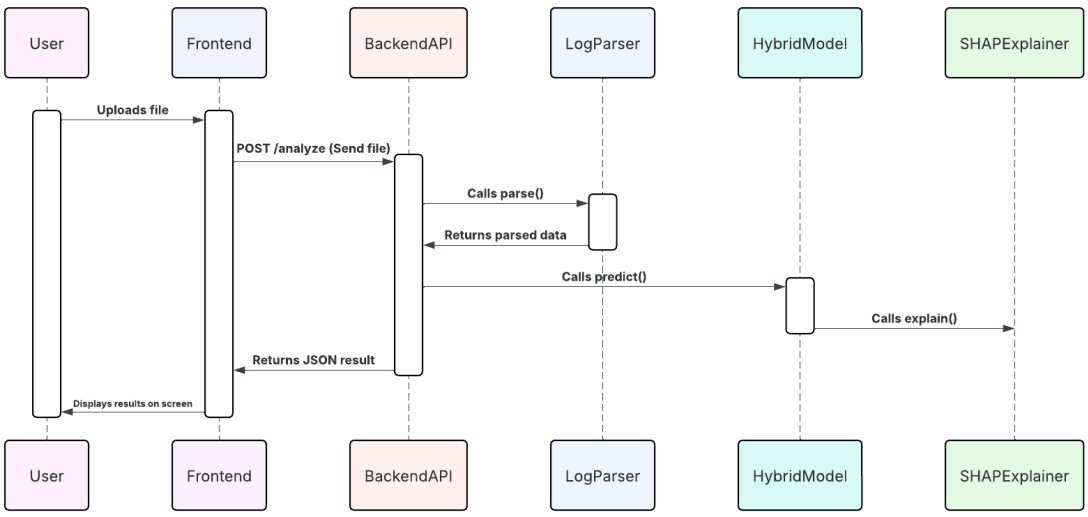

Figure 4.9 LogNomaly system inter-component messaging flow sequence diagram.

4.7 Activity Diagrams

Figure 4.10 details the algorithmic decision logic of the hybrid anomaly detection engine that forms the foundation of LogNomaly. The data flow begins with log parsing and feature extraction, followed by a three-stage filtering mechanism:

1. Rule-Based Control:

In the first stage, the data is compared against predefined threat rules. If a match is found (e.g., a known attack signature), the risk score is directly labeled as **1.0 (Critical Threat)** without passing through subsequent models, in order to reduce computational cost.

2. Unsupervised Learning (Isolation Forest):

Data that does not match any rule is forwarded to the Isolation Forest algorithm for unknown anomaly detection. If the model does not detect an anomaly, the data is labeled as **Normal (Risk: 0.0)**.

3. Supervised Learning and XAI (Random Forest & SHAP):

Data marked as *anomalous* by the Isolation Forest enters the Random Forest layer for final validation and classification. At this stage, the final **Risk Score** is computed, and the SHAP engine is activated to generate explainability data that clarifies the reasons behind this score.

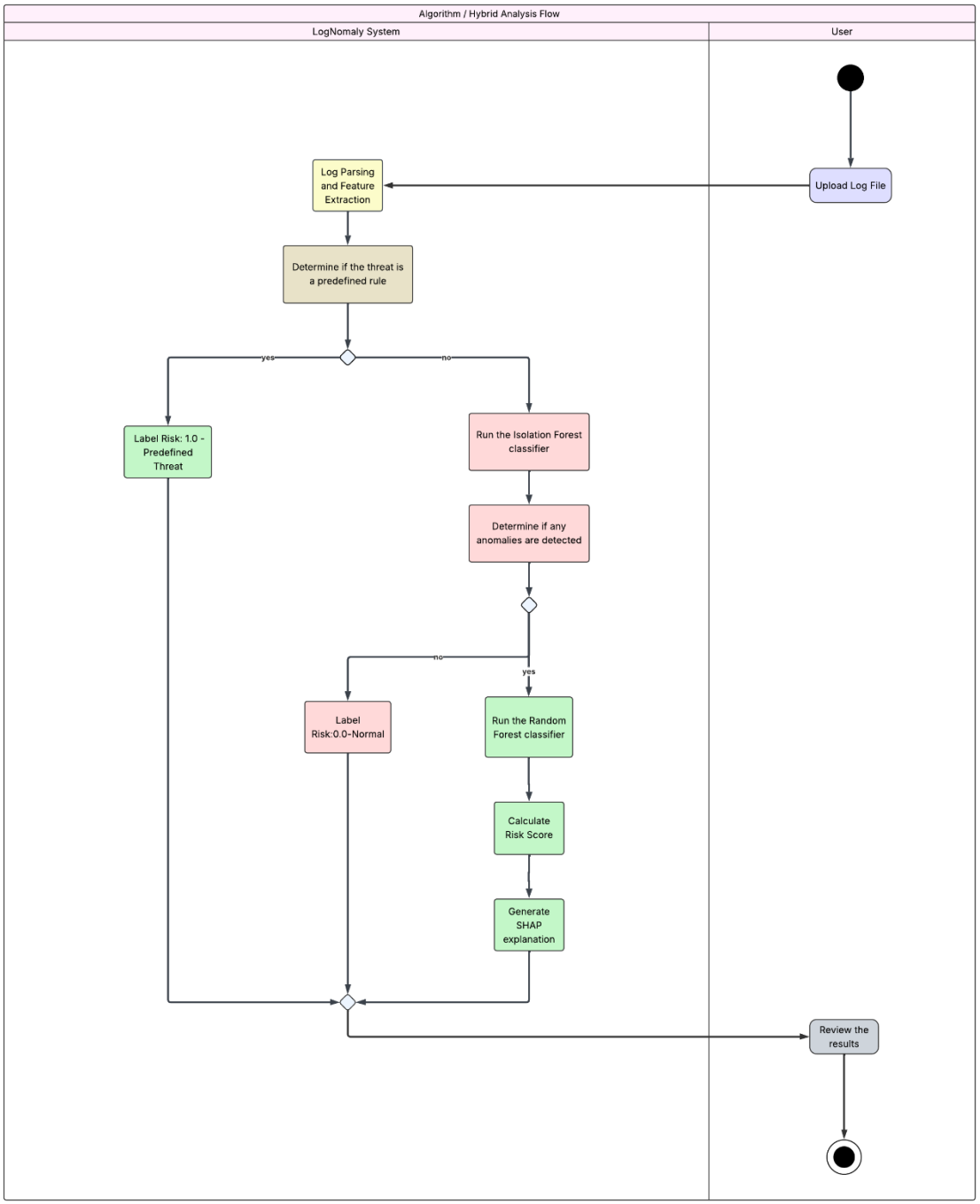

Figure 4.10 Algorithmic decision logic activity diagram of the hybrid anomaly detection engine.

Figure 4.11 illustrates the end-to-end workflow of the **File Upload and Analysis** process, in which the end user interacts with the system. This diagram models both the system's robustness against invalid data inputs and the steps of a successful analysis process:

- Validation:

When a user uploads a log file, the system first verifies the validity of the file format and its integrity. For unsupported or corrupted formats, the process is terminated and an error message is displayed to the user.

- Processing Pipeline:

Valid files are processed sequentially through log parsing, execution of the hybrid analysis engine, and visualization of the results (dashboard generation).

- Result Visualization:

The process concludes with the user reviewing the analysis results and detailed reports. This workflow diagram clarifies how the system tolerates user errors and enforces the correct processing sequence.

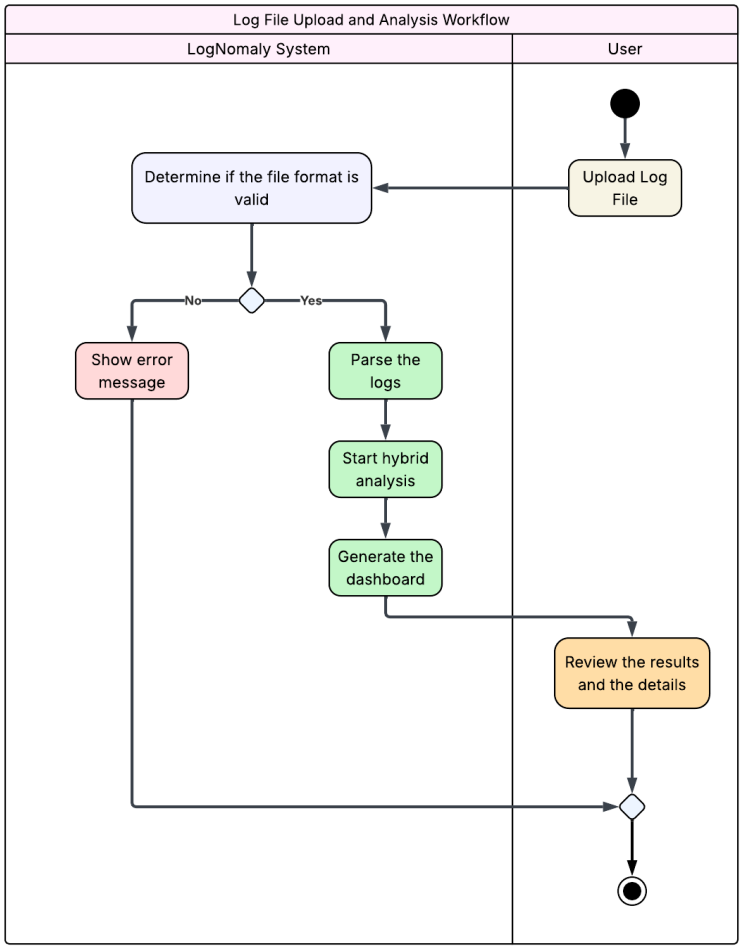

Figure 4.11 Activity diagram of the file upload and analysis process.

4.8 Deployment Diagrams

Figure 4.12 illustrates the physical deployment architecture and runtime environments of the LogNomaly system. The system is designed around two main nodes: the **Client** and the **Application Server**:

- Client Workstation:

  This node serves as the access point for end users. Secure communication is established via the HTTPS protocol over TCP port 443 using any modern web browser.

- Application Server:
  - ✓ **.NET Core Runtime:** Executes the *LogNomaly.dll* component, which handles the frontend and business logic.
  - ✓ **Python 3.x Environment:** Hosts the machine learning models (Isolation Forest and Random Forest) and the Flask-based API exposed via *app.py*.
  - ✓ Communication between these two environments is achieved through internal HTTP requests using a REST API architecture, enabling seamless interoperability between different technologies and supporting a polyglot architecture approach.

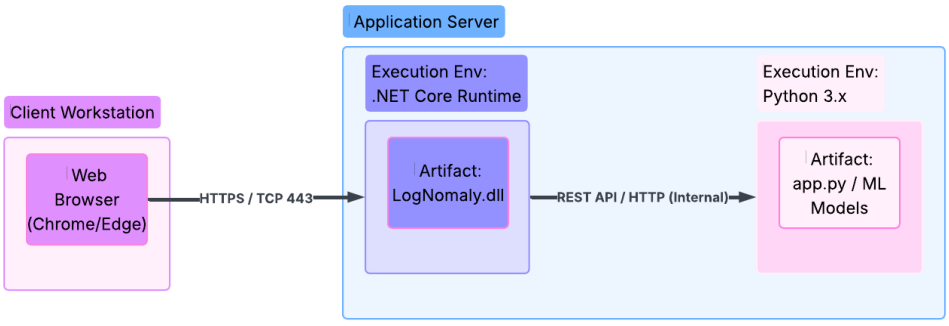

Figure 4.12 Deployment architecture and runtime environments of the LogNomaly system.

Figure 4.13 details the container-based architecture of the system in the **production environment** and its data management strategy. This structure, built using Docker technology, eliminates dependency issues and ensures service isolation:

- Container Isolation:

The frontend (.NET Core) and backend (Python API) services run in separate, independent Docker containers. While the frontend is exposed to the external world via ports 80/443, the backend API is accessible only through the internal network (port 5000).

- Data Persistence and Sharing (Shared Log Storage):

One of the most critical components of the system is the shared storage area created using **Docker volumes**. Log files uploaded by the frontend are written to this shared space and are directly read by the Python-based analysis engine. This approach prevents repeated network transfers of large files, thereby improving I/O performance.

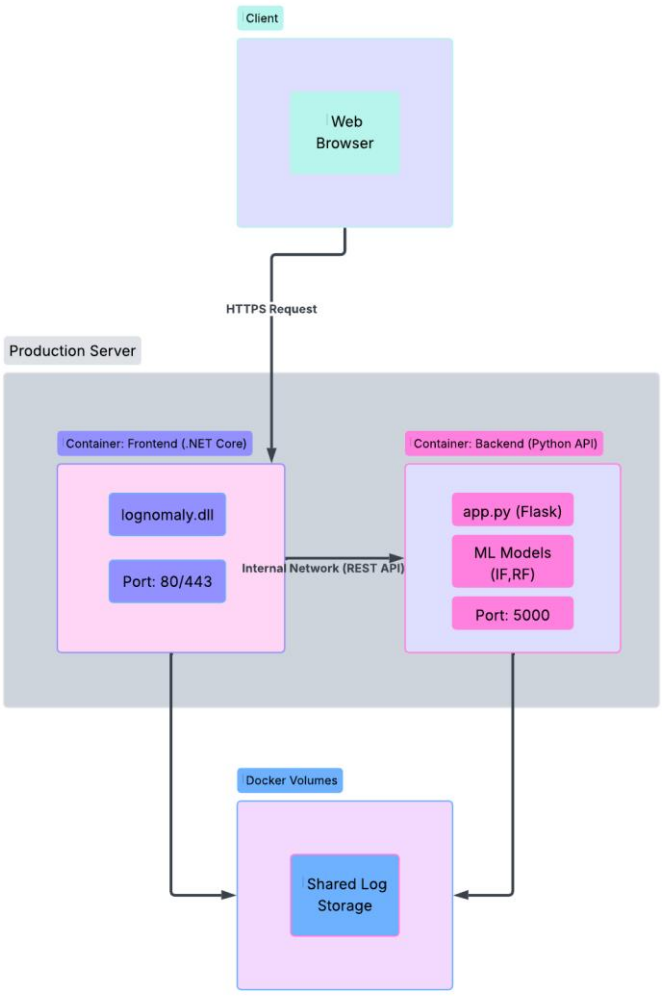

Figure 4.13 Deployment architecture and runtime environments of the LogNomaly system.

REFERENCES

Almodovar, C., Sabrina, F., Karimi, S., & Azad, S. (2024). LogFiT: Log anomaly detection using fine-tuned language models. *IEEE Transactions on Network and Service Management*, *21*(2), 1715-1723.

Brown, A., Tuor, A., Hutchinson, B., & Nichols, N. (2018, June). Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the first workshop on machine learning for computing systems* (pp. 1-8).

Chen, Z., Liu, J., Gu, W., Su, Y., & Lyu, M. R. (2021). Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908*.

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017, October). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 1285–1298). ACM.

Han, D., Sun, M., Li, M., & Chen, Q. (2023). LTAnomaly: A transformer variant for syslog anomaly detection based on multi-scale representation and long sequence capture. *Applied Sciences*, *13*(13), 7668.

Han, D., Wang, Z., Chen, W., Zhong, Y., Wang, S., Zhang, H., ... & Yin, X. (2021, November). Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (pp. 3197-3217).

Keyogeg, B., Thompson, M., Dawson, G., Wagner, D., Johnson, G., & Elliott, B. (2024). Automated detection of ransomware in windows active directory domain services using log analysis and machine learning. *Authorea Preprints*.

Khan, Z. A., Shin, D., Bianculli, D., & Briand, L. C. (2024). Impact of log parsing on deep learning-based anomaly detection. *Empirical Software Engineering*, *29*(6), 139.

Landauer, M., Skopik, F., & Wurzenberger, M. (2024). A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. *Proceedings of the ACM on Software Engineering*, *1*(FSE), 1354-1375.

Le, V. H., & Zhang, H. (2021, November). Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 492-504). IEEE.

Le, V. H., & Zhang, H. (2022, May). Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th international conference on software engineering* (pp. 1356-1367).

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., ... & Zhou, R. (2019, August). Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI* (Vol. 19, No. 7, pp. 4739-4745).

Pan, J., Liang, W. S., & Yidi, Y. (2024, May). Raglog: Log anomaly detection using retrieval augmented generation. In *2024 IEEE World Forum on Public Safety Technology (WFPST)* (pp. 169-174). IEEE.

Pospieszny, P., Mormul, W., Szyndler, K., & Kumar, S. (2025). ADALog: Adaptive Unsupervised Anomaly detection in Logs with Self-attention Masked Language Model. *arXiv preprint arXiv:2505.13496*.

Wang, S., Jiang, R., Wang, Z., & Zhou, Y. (2024). Deep learning-based anomaly detection and log analysis for computer networks. *arXiv preprint arXiv:2407.05639*.

Xie, Y., Zhang, H., & Babar, M. A. (2022, December). Loggd: Detecting anomalies from system logs with graph neural networks. In *2022 IEEE 22nd International conference on software quality, reliability and security (QRS)* (pp. 299-310). IEEE.

Yang, L., Chen, J., Gao, S., Gong, Z., Zhang, H., Kang, Y., & Li, H. (2024). Try with simpler-an evaluation of improved principal component analysis in log-based anomaly detection. *ACM Transactions on Software Engineering and Methodology*, *33*(5), 1-27.

Yang, Z., & Harris, I. G. (2025). LogLLaMA: Transformer-based log anomaly detection with LLaMA. *arXiv preprint arXiv:2503.14849*.

Yang, Z., Li, H., Yang, X., Peng, H., Shi, J., Peng, M., ... & Bai, H. (2023, May). User log anomaly detection system based on isolation forest. In 2023 2nd International Joint Conference on Information and Communication Engineering (JCICE) (pp. 79–84). IEEE.

Ying, S., Wang, B., Wang, L., Li, Q., Zhao, Y., Shang, J., ... & Geng, J. (2021). An improved KNN-based efficient log anomaly detection method with automatically

labeled samples. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *15*(3), 1-22.

Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., ... & He, P. (2024, February). Deep learning or classical machine learning? An empirical study on log-based anomaly detection. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (pp. 1–13). IEEE/ACM.

Yu, Z., Yang, S., Li, Z., Li, L., Luo, H., & Yang, F. (2024). LogMS: a multi-stage log anomaly detection method based on multi-source information fusion and probability label estimation. *Frontiers in Physics*, *12*, 1401857.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., ... & Zhang, D. (2019, August). Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 807–817). ACM.

Zhu, J., He, S., He, P., Liu, J., & Lyu, M. R. (2023, October). Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 355-366). IEEE.

Zou, J., & Petrosian, O. (2020). Explainable AI: Using Shapley value to explain complex anomaly detection ML-based systems. In *Machine learning and artificial intelligence* (pp. 152-164). IOS Press.