AI-Powered Application log Analysis and Monitoring System

Dr. S. Athinarayanan[1], aathithe@gmail.com[1]
Faculty, Department of Computer Science and Engineering
Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, Tamil Nadu, India
K. Jaya Prakash[2], K. Karthikeya[3], A. Vishnuvardhan Reddy[4]
vtu32116@veltech.edu.in, vtu33416@veltech.edu.in
Students, Department of Computer Science and Engineering
Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, Tamil Nadu, India

ABSTRACT— Modern software applications generate huge volumes of runtime statistics, which include informational messages, warnings, mistakes, and exception details. Monitoring these activities manually is time-ingesting and might postpone the identification and resolution of utility disasters. This paper proposes an AI-Powered Application Analysis and Monitoring System that collects application events and logs, analyses their severity and patterns, and signals the responsible developer whilst an unusual circumstance is detected. The proposed device is designed round a tracking pipeline including log collection, preprocessing, occasion classification, anomaly identity, garage, dashboard visualization, and alert notification. Informational events assist builders understand normal utility activity, warning events suggest conditions that can require attention, and error activities pick out screw ups that want investigation. AI-assisted analysis can help perceive unusual patterns and offer useful context for troubleshooting. The gadget objectives to reduce continuous manual log inspection, improve consciousness of utility fitness, and assist quicker incident reaction. The proposed design is intended to be evaluated using a managed demonstration application and check instances representing regular operations, warnings, and application errors. Actual overall performance outcomes may be mentioned after implementation and trying out.

Keywords— *Application Monitoring, Log Analysis, Artificial Intelligence, Anomaly Detection, Error Alerting, Software Reliability.*

I. INTRODUCTION

Software applications are widely used in schooling, commercial enterprise, verbal exchange, banking, and many other domain names. During execution, packages generate runtime information that facts person moves, gadget activities, warnings, and mistakes. These facts, normally known as software logs, are treasured for know-how software behaviour and identifying problems. However, as programs come to be extra complex, the quantity of generated logs will increase, making guide inspection difficult. Traditional tracking methods often depend on developers checking log files, dashboards, or server consoles to identify screw ups. This process can postpone the detection of critical mistakes, especially while an utility produces many activities. A developer might not right away word a crucial exception among habitual informational messages. Delayed detection can influence software availability, reliability, and user revel in. Automated tracking and alerting can help address this problem through constantly processing application occasions and notifying the accountable developer while attention is needed.

The proposed AI-Powered Application Analysis and Monitoring System specializes in amassing application logs, classifying events, identifying suspicious patterns, and sending signals to developers. The gadget distinguishes informational, warning, and blunders events and offers relevant information thru a monitoring dashboard. AI-assisted evaluation is intended to guide the identity of uncommon log styles and help builders look at feasible causes. The most important goal is to provide well timed visibility into software hobby without requiring developers to look at the utility constantly.

II. LITERATURE SURVEY

Automated log evaluation has been studied as an important method to software program reliability and device protection. Xu et al. [1] proven that console logs can be mined to detect troubles in huge-scale systems. Lou et al. [2] proposed mining invariants from console logs to discover odd execution behavior. These studies hooked up that log facts contains beneficial facts for

detecting machine troubles; however, they also highlighted the need for computerized processing due to the fact guide inspection does no longer scale well.

Later research delivered system-gaining knowledge of and deep-gaining knowledge of strategies for analyzing log sequences. Du et al. [3] proposed DeepLog, which makes use of a Long Short-Term Memory (LSTM) network to analyze normal log styles and become aware of deviations. He et al. [4] suggested on gadget log analysis for anomaly detection, even as Du and Li [5] brought Spell, a streaming method for parsing machine occasion logs. He et al. [6] proposed Drain, an internet log parsing approach that makes use of a hard and fast-intensity tree. These approaches enhance the automation of log processing and provide foundations for monitoring systems that must deal with constantly generated activities.

Recent studies have centered on semantic information, changing log codecs, and broader automatic log-evaluation workflows. Meng et al. [7] delivered LogAnomaly to discover sequential and quantitative anomalies in unstructured logs. Zhang et al. [8] proposed LogRobust to improve detection whilst log records changes through the years. He et al. [9] provided a survey of automatic log evaluation for reliability engineering, and Zhu et al. [10] discussed tools and benchmarks for computerized log parsing. More recent paintings also examine log-based totally anomaly detection without a separate parsing level [11] and deep-studying strategies for log anomaly detection [12], contextual protection event evaluation [13], foundational supercomputer log characterization [14], and distributed structures anomaly detection [15]. These studies motivate the proposed system's mixture of event series, severity class, anomaly evaluation, and developer alerting**.**

III. PROPOSED METHODOLOGY

3.1 System Overview

The proposed device video display units a software application via receiving runtime activities and log messages from the software. Each event is processed, categorized, and stored. The evaluation issue tests the occasion kind, severity, frequency, and available context. When an errors or suspicious pattern is detected, the gadget creates an alert containing relevant records for the developer. A dashboard provides a consolidated view of utility hobby and monitoring status.

3.2 Proposed Architecture

The stop-to-end tracking pipeline includes the following 8 core architectural tiers:

•Monitored Application: The application generates runtime events together with user moves, a success operations, warnings, and exceptions.

•Log Collection: A logging issue captures occasion messages, timestamps, severity tiers, and available source records.

•Log Preprocessing: The accrued messages are wiped clean and prepared. Unnecessary formatting is removed, and useful fields are extracted.

•Event Classification: Events are grouped into categories together with INFO, WARNING, and ERROR. The category may additionally use predefined severity tiers and configurable regulations.

•AI-Assisted Analysis: The AI-assisted evaluation module is a key component of the proposed machine architecture. It examines occasion styles, identifies mistakes, warnings, and uncommon activity styles, and allows discover potential problems inside the software. The layout can be prolonged with machine-learning fashions skilled on labelled or historical log statistics. Analyzed statistics is processed and stored inside the database, whilst essential mistakes are forwarded to the alert control module.

•Database Storage: Processed logs, occasion details, alert statistics, and timestamps are saved for later assessment and ancient auditing.

•Developer Dashboard: Developers can view current occasions, filter logs by means of severity, inspect software pastime, and get hold of real-time visual comments.

•Alert Notification: When a configured crucial condition or anomaly is recognized, the system generates a notification and displays relevant information on the developer dashboard, enabling developers to perceive and solve application troubles successfully.

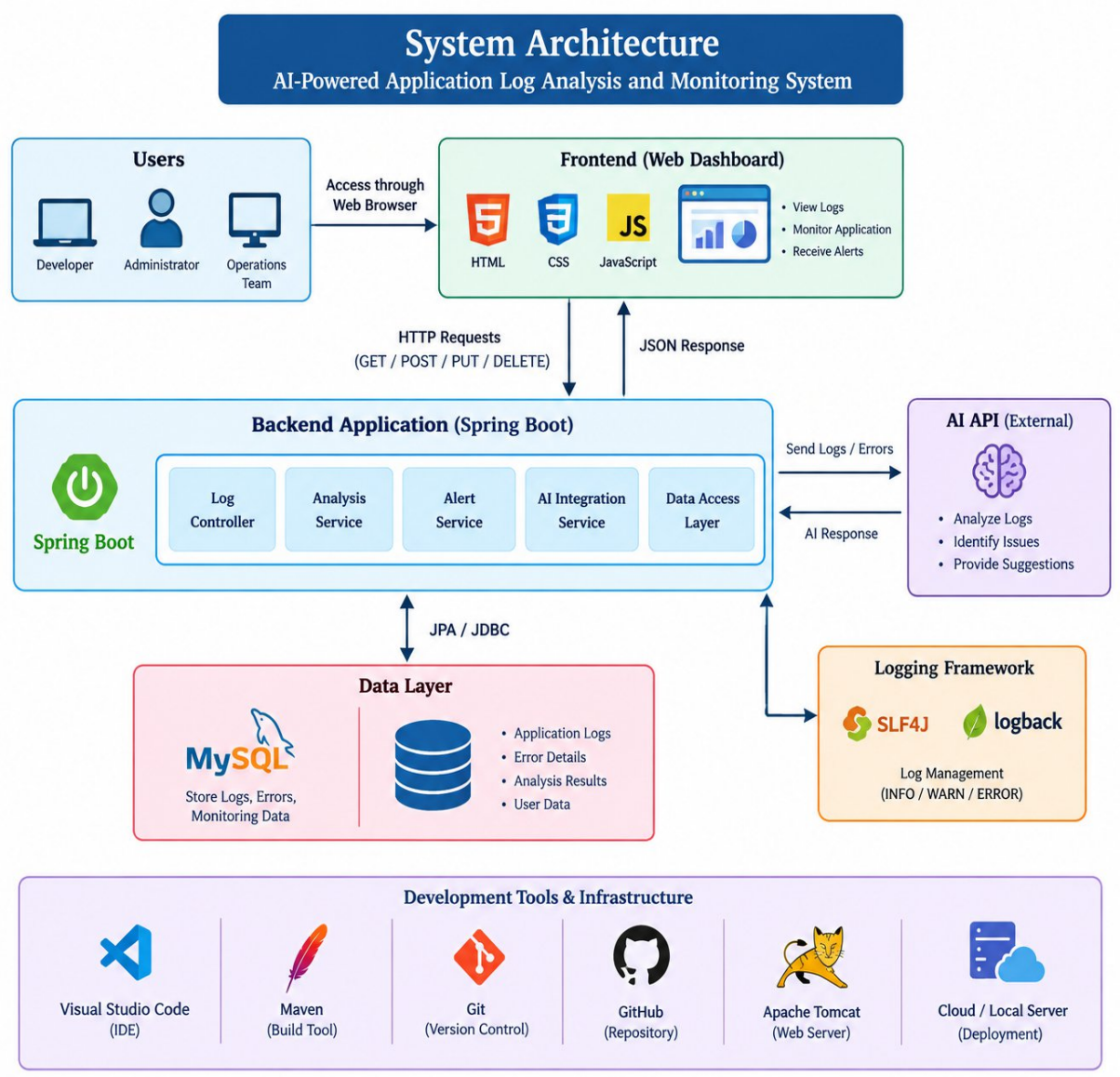


Figure 1: Overall System Architecture of the AI-Powered Application Log Analysis and Monitoring System.

3.3 Working Procedure

The operational workflow of the proposed monitoring framework executes sequentially across eight steps:

Step 1 — Event Generation: The monitored application performs an operation and generates a runtime log event.
Step 2 — Event Capture: The collection component captures the event payload along with its exact timestamp.
Step 3 — Log Preprocessing: The preprocessing component cleans and extracts relevant structured fields from the raw log.
Step 4 — Severity Classification: The classification component assigns or parses the event severity level (INFO, WARNING, ERROR).
Step 5 — Pattern & AI Analysis: The analysis component evaluates configured rules and, where implemented, AI-based anomaly patterns.
Step 6 — Persistence & Visualization: The processed event is stored in the database and rendered on the monitoring dashboard.
Step 7 — Alert Dispatch: If the event satisfies an alert condition, a notification is dispatched to the responsible developer.
Step 8 — Incident Resolution: The developer reviews the alert context, investigates the root cause, and takes corrective action.

3.4 System Modules

A. Application Log Collection Module: Receives log messages from the monitored utility, captures timestamps, event messages, and severity metadata, and supports non-stop series of runtime events.

B. Log Preprocessing Module: Organizes and normalizes incoming log messages, extracts dependent fields for downstream evaluation, and reduces inconsistencies across varying log codecs.

C. Event Classification Module: Classify events as INFO, WARNING or ERROR, distinguish routine operations from actionable issues, and support configurable severity rules.

D. AI-Based Analysis Module: Looks at sequences of events and time patterns. Finds unusual events or deviations from expected behavior. Can be improved with trained models as suitable datasets become available**.**

E. Database Management Module: Stores processed logs, event categories and alert history Allows querying and filtering historical events Maintains an audit trail for investigation

F. Monitoring Dashboard Module: Displays real-time application activity, health indicators, and event summaries while allowing filtering by date, category, and severity.

G. Alert and Notification Module: Generates structured alerts for critical errors or detected anomalies containing event timestamp, severity, and diagnostic message, and notifies the responsible developer through configured channels.

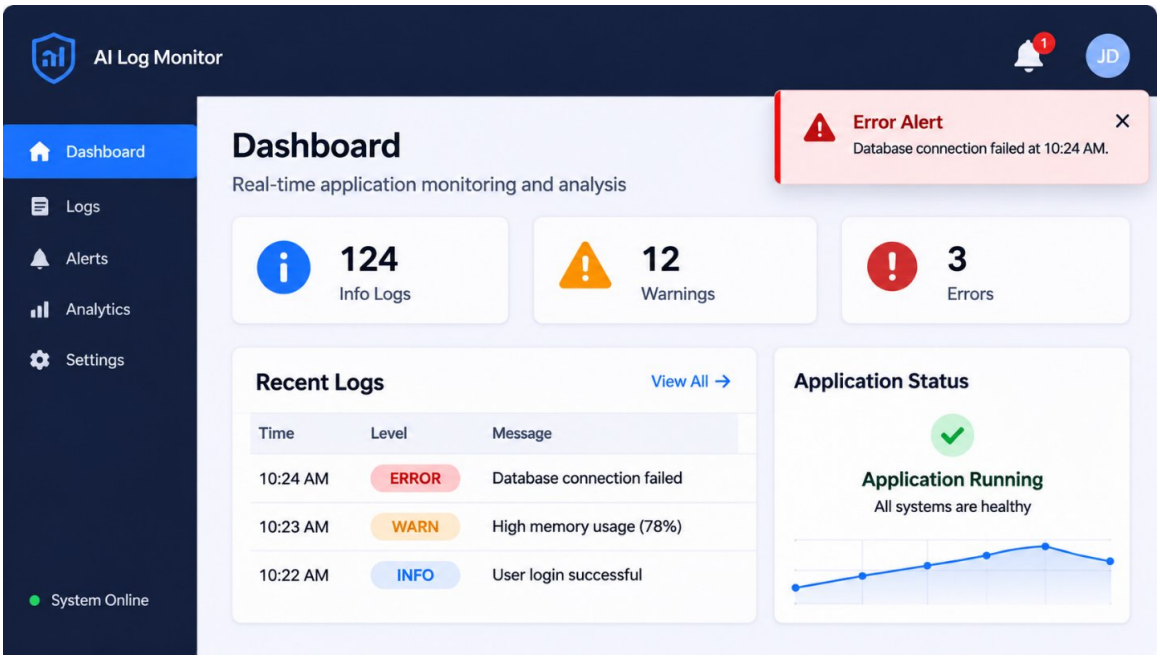


Figure 2: Real-Time Monitoring Dashboard Interface Showing Severity Counters, Recent Logs, and Live Error Alert Notification.

3.5 Proposed Technologies

The reference implementation uses Java and Spring Boot for backend micro-services, RESTful APIs for communication, SLF4J and Logback for structured logging, MySQL (via JPA/JDBC) for persistent relational storage, and a responsive web interface (HTML, CSS, JavaScript) for the monitoring dashboard. Depending on the model selected and available training data, external AI APIs or machine-learning libraries are embedded for anomaly and pattern analysis. The final production report contains only the verified technologies and tools utilized in the completed implementation.

IV. RESULTS AND DISCUSSION

The proposed system will be evaluated with a controlled demonstration application that produces normal events, warning messages and error conditions. During the testing, the monitoring pipeline captures the generated events, classifies them, stores them in the database and shows them on the dashboard. When a critical error or anomaly is generated as configured, the alert module informs the developer in charge about the event with the related event details. The assessment includes the total number of test events, correctly classified events, missed events, false alerts, and notification delivery delay. Because there is no experimental benchmark data measured for this design draft, any claims about numerical accuracy, response-time, and performance are intentionally omitted and will be populated after experimental testing is complete.

.4.1 Functional Verification Test Cases

- **Normal Application Operation:** Verify that routine INFO events are continuously collected, persisted, and displayed accurately on the dashboard.
- **Warning Condition Handling:** Verify that WARNING events (e.g., elevated memory or CPU utilization) are properly categorized and highlighted.
- **Application Exception Detection:** Verify that critical ERROR events (e.g., database connection failures) are stored and trigger an immediate developer alert.
- **Repeated Error Management:** Verify how recurring error bursts are handled and whether duplicate alert notifications are controlled.
- **Dashboard Search & Filtering:** Verify that developers can efficiently query and filter historical events by severity level and time range.
- **Notification Delivery Latency:** Record whether the configured developer notification is successfully received and measure end-to-end delivery delay.



Figure 3: Illustrative Log Analytics View Displaying Log Volume Trends, Severity Distribution, Top Error Sources, and Anomaly Spike Detection.

V. COMPARATIVE ANALYSIS

Table I summarizes the functional differences that the proposed AI-powered framework targets against traditional manual log inspection and basic rule-based monitoring tools. This is an architectural comparison at the design level, not measured runtime metrics for performance.

TABLE I: FUNCTIONAL COMPARISON OF APPLICATION MONITORING APPROACHES

Evaluation Aspect	Manual Monitoring	Basic Rule-Based	Proposed AI-Powered System
Log Collection	Manual or separate file access	Automated ingestion	Automated continuous pipeline
Event Categorization	Manual human inspection	Static predefined rules	Automated severity classification
Pattern Analysis	Human observation only	Limited keyword/threshold rules	AI-assisted pattern & context analysis
Anomaly Identification	Depends entirely on developer	Known static conditions only	Configurable rules + AI anomaly detection

Evaluation Aspect	Manual Monitoring	Basic Rule-Based	Proposed AI-Powered System
Developer Notification	Often significantly delayed	Basic rule-triggered alerts	Real-time contextual alert notifications
Log History & Audit	Scattered separate log files	Depends on specific tool	Centralized relational database storage
Dashboard Visibility	Not always available	Tool-dependent / fragmented	Unified real-time integrated dashboard

Our proposed system will combine centralized log handling, event classification, AI-assisted analysis and developer notification into one unified monitoring workflow. It is designed to automatically process events, instead of manual inspection. The planned AI-assisted analysis can be extended to find complicated patterns that are not represented by simple keyword rules, as in a basic rule-based monitor. The true quantitative benefit will be determined by full implementation and comparative testing.

VI. CONCLUSION

In this paper we present the design of an AI-Powered Application Analysis and Monitoring System for improving application observability and supporting faster error investigation. The proposed approach will collect runtime logs, preprocess messages, classify events, analyse unusual patterns, store event details and notify the responsible developer when configured error or anomaly conditions are met. The monitoring dashboard provides visibility into application activity and alert history. The system is designed to reduce the need for continuous manual log inspection and to help developers respond to problems in a timely manner. The effectiveness, accuracy, and response time of the proposed approach will be determined by future implementation and evaluation.

REFERENCES

[1] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting Large-Scale System Problems by Mining Console Logs," in Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2009.

[2] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining Invariants from Console Logs for System Problem Detection," in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2010.

[3] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1285–1298, 2017. DOI: 10.1145/3133956.3134015.

[4] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE), 2016.

[5] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 859–864, 2016. DOI: 10.1109/ICDM.2016.0103.

[6] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in Proceedings of the IEEE International Conference on Web Services (ICWS), 2017. DOI: 10.1109/ICWS.2017.13.

[7] W. Meng et al., "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," in Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 4739–4745, 2019. DOI: 10.24963/ijcai.2019/658.

[8] X. Zhang et al., "Robust Log-Based Anomaly Detection on Unstable Log Data," in Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2019. DOI: 10.1145/3338906.3338931.

[9] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A Survey on Automated Log Analysis for Reliability Engineering," ACM Computing Surveys, vol. 54, no. 6, Article 130, 2021. DOI: 10.1145/3460345.

[10] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and Benchmarks for Automated Log Parsing," in Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130, 2019. DOI: 10.1109/ICSE-SEIP.2019.00021.

[11] H. Guo, S. Yuan, and X. Wu, "Log-based Anomaly Detection Without Log Parsing," in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021. DOI: 10.1109/ASE51524.2021.9678773.

[12] V.-H. Le and H. Zhang, "Deep Learning for Anomaly Detection in Log Data: A Survey," Machine Learning with Applications, vol. 12, Article 100470, 2023. DOI: 10.1016/j.mlwa.2023.100470.

[13] T. van Ede et al., "DeepCASE: Semi-Supervised Contextual Analysis of Security Events," in Proceedings of the IEEE Symposium on Security and Privacy (SP), 2022.

[14] A. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," in Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN), 2007.

[15] Y. Yu, Y. Ruan, and G. Song et al., "Log-Based Anomaly Detection for Distributed Systems: State of the Art, Industry Experience, and Open Issues," Journal of Software: Evolution and Process, 2024. DOI: 10.1002/smr.2650.