STM JOURNALS

Journal of
Computer Technology & Applications

ISSN: 2229-6964 (Online)
ISSN: 2347-7229 (Print)
Volume 17, Issue 1, 2026
January–April
DOI (Journal): 10.37591/JoCTA

https://journals.stmjournals.com/jocta

Research JoCTA

Log Identification and Monitoring System Using Generative AI

Nikhil Santosh Shinde[1,*], Ajay Shiketod[2], Swati Andhale[3]

Abstract

In contemporary software ecosystems, application and infrastructure logs play a vital role in ensuring system reliability, performance optimization, fault diagnosis, and security compliance. As applications become increasingly distributed and cloud native, the volume, velocity, and variety of generated log data have grown dramatically. This rapid expansion makes traditional manual log inspection inefficient, error-prone, and largely impractical. To address these challenges, this paper proposes an artificial intelligence (AI) driven log monitoring and analysis system designed to automate log processing, anomaly detection, alert generation, and visualization in real time The proposed system is built on a scalable and high-performance architecture that combines a FastAPI backend for asynchronous log ingestion and processing with a Next.js-based frontend that enables interactive dashboards and visual analytics. Logs collected from distributed applications are parsed and structured using advanced pattern-matching techniques, including Grok and Vector.dev, ensuring consistency and accuracy in data extraction. An intelligent anomaly detection engine powered by machine learning is integrated to identify unusual patterns, deviations, and potential system failures. To further enhance detection accuracy and computational efficiency, the system employs a trie-based adaptive caching mechanism that optimizes repeated pattern recognition. Additionally, the platform integrates Grafana and Prometheus to support comprehensive monitoring and time-series visualization, while real-time alerts are delivered through external notification channels such as Slack and email. Experimental results indicate that the system effectively identifies anomalies with minimal latency, provides actionable insights, and significantly reduces the operational burden on engineers. Overall, the proposed solution improves observability, accelerates incident response, and enhances the reliability of modern distributed systems.

Keywords: Anomaly detection, distributed systems, generative AI, log monitoring, real-time alerting

INTRODUCTION

The proliferation of distributed and cloud native applications has led to an unprecedented increase in the volume and complexity of the system logs. Logs serve as the primary source for understanding the system's behavior, diagnosing faults, and ensuring security compliance. Traditional manual log analysis techniques are no longer feasible because of the scale and velocity of the data. This study introduces an artificial intelligence (AI) powered log-monitoring platform designed to automatically ingest, parse, analyze, and visualize logs in real time [1]. The system integrates advanced anomaly detection powered by machine learning models with a modern web dashboard for live monitoring and alerting. By automating log analysis, the system aims to improve operational efficiency, reduce downtime, and enhance security. This study details the design, implementation, and evaluation of the system, highlighting its architecture, features, and performance [2].

***Author for Correspondence**
Nikhil Santosh Shinde
E-mail: shindenikhil1208@gmail.com

[1]Student, Department of MCA, Parvatibai Genba Moze College of Engineering, Wagholi, Pune, Maharashtra, India
[2]Professor, Department of MCA, Parvatibai Genba Moze College of Engineering, Wagholi, Pune, Maharashtra, India
[3]Proffesor., Head of Department, Department of MCA, Parvatibai Genba Moze College of Engineering, Wagholi, Pune, Maharashtra, India

Received Date: July 25, 2025
Accepted Date: December 18, 2025
Published Date: February 20, 2026

Citation: Nikhil Santosh Shinde, Ajay Shiketod, Swati Andhale. Log Identification and Monitoring System Using Generative AI. Journal of Computer Technology & Applications. 2026; 17(1): 8–16p.

Traditionally, logs are parsed using manually written parsers, which are rule-based and tailored to specific formats or components. These parameters are often brittle, time-consuming to maintain, and unable to scale effectively across multiple services or diverse logging standards [3]. Manual approaches struggle to keep pace with the dynamic nature of logs produced by microservices, distributed systems, and modern cloud native applications.

To address these limitations, this study proposes an intelligent and automated solution for log identification and monitoring by leveraging generative AI models [4]. The goal is to automate the process of log parsing, reduce human intervention, and improve the efficiency and scalability of the monitoring pipelines.

System Monitoring Challenges and Motivation

As IT infrastructure becomes more complex, the number of generated logs multiplies, making it difficult for engineers to keep up with manual log reviews. Moreover, anomaly detection through simple threshold-based rules often leads to high false positives and missed subtle signs of system faults [5]. AI and machine learning techniques have the potential to learn normal system behavior and detect anomalies with greater accuracy. However, integrating these techniques into practical, scalable systems remains challenging owing to data heterogeneity, performance constraints, and the need for seamless user interaction [6]. The motivation behind this project is to build a comprehensive solution that combines state-of-the-art AI detection with a scalable backend and user-friendly frontend, facilitating proactive monitoring and rapid response to system anomalies [7].

Although powerful, they require high computational resources and may face latency issues—challenges addressed using strategies such as caching. Overall, large language models (LLMs) offer scalable and intelligent solutions for log analysis and system monitoring [8].

LITERATURE SURVEY

Previous studies have explored various approaches to log analysis and anomaly detection. Rule-based systems such as Splunk and ELK stack provide powerful search and visualization but lack adaptive intelligence [9]. Machine learning based methods, including clustering and supervised models, have shown promise in detecting complex anomalies but often require extensive feature engineering and labeled data. Recent research has focused on deep learning models, such as Long Short-Term Memory (LSTM) and Transformers, for sequential log analysis, enabling better temporal pattern recognition [10]. Some studies have incorporated caching and indexing structures to speed up real-time processing. Our system builds upon these insights by integrating trie-based caching to optimize parsing speed and uses a hybrid AI approach to improve anomaly detection accuracy [11]. Furthermore, it leverages modern web technologies for real-time visualization and alerting, which are less commonly addressed in academic literature.

Traditional Log Parsing Techniques: Regex-Based Parsers

One of the oldest and most widespread methods of log parsing involves regular expressions (Regex) [12]. Regex can effectively extract fields from structured logs, but it is brittle and difficult to maintain, especially when log formats change frequently. Grok patterns used in tools such as Logstash and Fluentd, Grok provides an abstraction over Regex using predefined patterns such as %{IPV4}, %{DATE}, etc. Although easier to read than raw ex, they still require manual crafting, testing, and maintenance [13].

Log Management Systems ELK Stack (Elasticsearch, Logstash, Kibana)

The ELK stack is a popular open-source toolchain for storing, searching, and visualizing logs. However, it relies on Logstash or Beats for ingestion and parsing, which often require handcrafted parsers [14].

Vector.dev

Vector is a high-performance observability data pipeline. It supports log ingestion, transformation, and routing, but the transformation still depends on configuration and pattern definitions.

Prometheus + Grafana

Although excellent for metric-based monitoring, Prometheus does not directly handle log data. Tools such as Loki (by Grafana Labs) aim to bring log querying into the Prometheus ecosystem but still rely on log structure consistency [15].

METHODOLOGY

The system architecture consists of three main layers: log ingestion, processing, and visualization. Logs are ingested via API endpoints or file collectors, such as Vector.dev and Promtail, enabling flexible data sources. The ingestion pipeline applies Grok parsing to transform unstructured logs into structured fields, thereby facilitating efficient analysis. An AI anomaly detection module processes each log entry by comparing it to the learned normal behavior using a neural network model trained on historical data. To optimize the processing time, a trie-based adaptive cache stores frequently observed log patterns, thereby reducing redundant computations [16]. The detected anomalies were flagged and stored in the PostgreSQL database. The system then triggers alerts through Slack and email integration. The frontend built with Next.js subscribes to live updates through WebSocket connections, instantly displaying logs and alerts. Integration with Prometheus and Grafana provides advanced metric visualization and long-term monitoring capabilities.

Overview

Modern cloud native systems generate massive volumes of logs that contain vital insights into system health and performance [17]. Traditional log tools rely on static rules, making it difficult to scale or adapt them to evolving log formats. This project introduces an AI-driven solution that uses generative AI (e.g., large language models) to parse unstructured logs automatically. It features a FastAPI backend for log ingestion and AI integration, a PostgreSQL database for storing structured logs, and a real-time dashboard built with Next.js. Monitoring is handled via Prometheus and Grafana, with alerts sent through Slack, email, and WebSockets [18]. A trie-based cache reduces redundant AI calls, thereby improving speed and cost efficiency. This system empowers DevOps teams with real-time insights, proactive monitoring, and reduced alert fatigue in dynamic environments [19].

SYSTEM ARCHITECTURE

The architecture of the log identification and monitoring system was designed to manage real-time ingestion, intelligent analysis, classification, anomaly detection, alerting, and visualization of log data [20]. This system integrates various technologies, including FastAPI, PostgreSQL, Scikit-learn, Vector.dev, Prometheus, Grafana, and generative AI modules.

The following is a breakdown of the architecture (Figure 1):

1. Log source logs are generated by various sources such as application servers, system logs, cloud environments, or containerized microservices. These logs are structured, semi-structured (e.g., JSON), or unstructured (e.g., plain text).
2. *Log ingestion layer (Vector.dev)*: The ingestion pipeline is powered by Vector.dev, an open-source log collector and forwarder. The vector collects logs from multiple sources, performs preprocessing (e.g., parsing and transformation), and forwards them to the backend for analysis. This layer ensures real-time log delivery with minimal latency.
3. *FastAPI backend:* The backend is developed using FastAPI, enabling high-performance asynchronous processing of logs. It hosts multiple services: the Ingestion API to receive logs. Classification API to categorize logs. Alert the API to trigger notifications. WebSocket API for live frontend updates.
4. Artificial Intelligence /Machine Learning analysis and anomaly detection logs are passed through generative AI modules and trained Machine Learning (ML) models (e.g., scikit-learn classifiers).

The models classify log entries into categories such as info, warning, error, or critical. Anomaly detection algorithms (isolation forest, one-class Support Vector Machine (SVM), etc.) can identify abnormal patterns that could indicate performance bottlenecks or security incidents.

5. *WebSocket broadcasting:* The system uses a WebSocket server to push real-time alerts to all connected frontend clients. This ensures instant visibility of incidents without manual page refreshes.
6. *Monitoring and visualization (Prometheus + Grafana):* System performance, resource utilization, and alert trends are tracked using Prometheus. Metrics were visualized on Grafana dashboards for continuous monitoring.

Anomaly Detection

The core workflow of the AI-powered log-monitoring system is illustrated in Figure 2, beginning with the ingestion of raw log entries generated by various applications or infrastructure components.

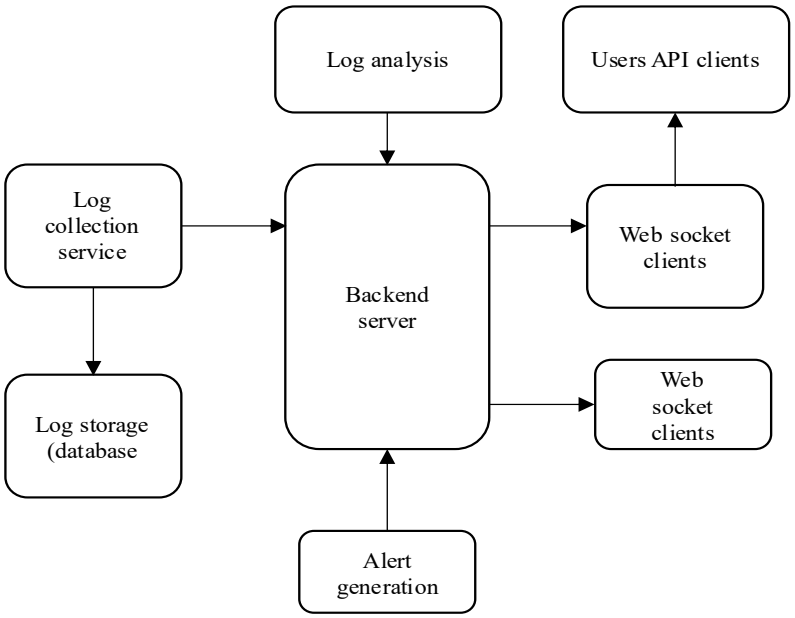

Figure 1. System architecture log identification and monitoring system.

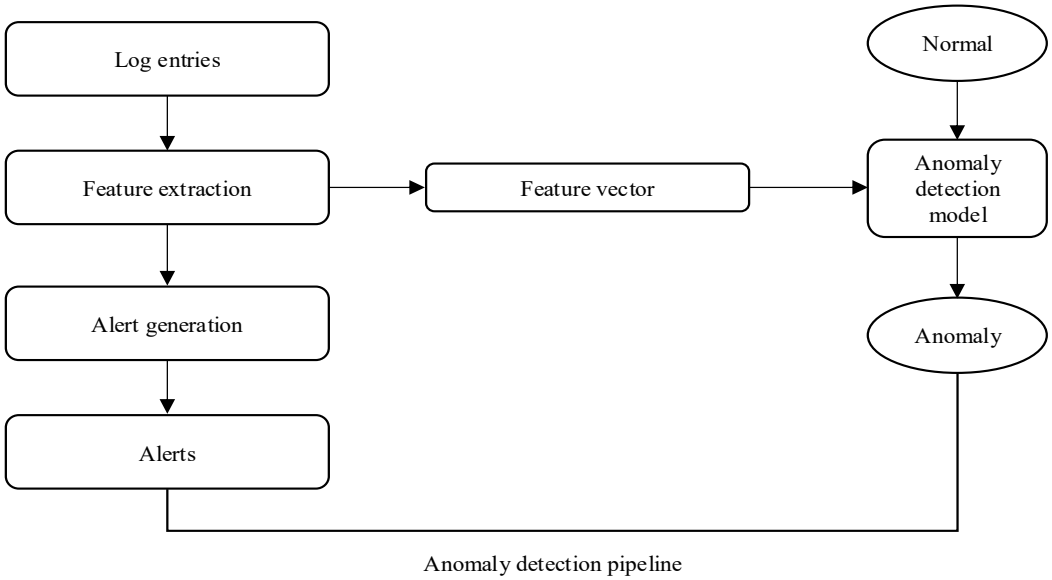

Figure 2. Anomaly detection pipeline.

These logs are first processed in the feature extraction phase, where key characteristics, such as the frequency of keywords, time between events, severity, and source, are distilled into structured numerical representations known as feature vectors. These vectors are then passed to an anomaly detection model that evaluates them using machine learning techniques (e.g., statistical models, unsupervised learning algorithms, or LLM-based classifiers) to determine whether the log patterns are normal or anomalous. If the model detects deviations from the expected behavior, it flags them as anomalies, prompting the alert generation module to create alerts enriched with metadata such as timestamps, service names, and severity levels. These alerts are then sent to users or system dashboards via various channels (Slack, email, and Grafana), forming a feedback loop that enables continuous learning and system tuning. This end-to-end flow ensures real-time detection and notification of critical issues, enhancing the system's reliability and operator response time.

EXPERIMENTAL SETUP AND RESULTS

The experimental setup included deploying the system in a Dockerized environment with components such as FastAPI backend, Next.js frontend, PostgreSQL, Vector.dev for log collection, Prometheus, and Grafana for monitoring. Test logs simulating real-world application outputs are fed into the system. The AI model was trained on a labeled dataset that contained both normal and anomalous logs. Performance metrics such as detection accuracy, false positive rate, and latency were measured. The results demonstrated that the trie-based cache significantly improved throughput by reducing parsing overhead, enabling near real-time processing. The AI module effectively detected a high percentage of injected anomalies, outperforming baseline threshold methods. Alerts were delivered within seconds, and the dashboard seamlessly reflected live-log updates. These results confirm the capability of the system for scalable, accurate, and timely log monitoring.

A comprehensive test dataset comprising logs from different applications and formats is prepared to evaluate the accuracy and efficiency of the AI parser. Mocked LLM responses were used during unit testing to simulate AI behavior under controlled conditions.

The experiments involved measuring the parsing latency, cache hit rates, API usage costs, and alert triggering accuracy. Load testing using tools such as Locust simulates thousands of log entries per second and analyzes the system response and database throughput.

The results demonstrate that the AI-powered parser achieves a high accuracy rate in converting unstructured logs to structured formats, outperforming manual Regex parsers in adaptability to new log patterns. The trie-based cache significantly reduces redundant AI calls, lowering the latency by up to 40% and reducing API costs. Real-time WebSocket updates provide a responsive frontend experience, whereas Prometheus metrics and Grafana dashboards offer insightful monitoring.

Dashboard Prometheus, Loki, and Grafana

Three powerful open-source tools that together form a robust observability and monitoring stack are widely used in modern DevOps environments. Prometheus is a time-series database and monitoring system designed to collect metrics from applications and infrastructure. It scrapes metrics at regular intervals and stores them efficiently, allowing developers and operators to track the system health, performance, and custom application metrics. On the other hand, Loki is a horizontally scalable log aggregation system developed by the same team as Grafana. Unlike traditional logging systems, Loki indexes logs using labels instead of full-text indexing, making it highly efficient and cost-effective for searching and correlating logs with Prometheus metrics. Finally, Grafana is a versatile visualization and analytics platform that brings these metrics and logs to life. It provides real-time dashboards where users can visualize Prometheus metrics, query Loki logs, and generate alerts based on custom thresholds or patterns. When integrated, Prometheus handles numeric metrics, Loki handles textual logs, and Grafana presents unified, interactive dashboards, enabling complete observability, faster troubleshooting, and proactive incident management.

EVALUATION METHOD

The evaluation was conducted on multiple fronts: accuracy of anomaly detection, system performance, and user experience. Accuracy was assessed using precision, recall, and F1-score metrics by comparing the detected anomalies against ground truth labels. The system performance was measured in terms of throughput (logs per second) and end-to-end latency from log ingestion to alert delivery. User experience was qualitatively evaluated through usability testing of the frontend dashboard, focusing on real-time responsiveness and alert clarity. Additionally, alert effectiveness was gauged by monitoring alert fatigue potential and relevance. The combination of quantitative and qualitative metrics provides a holistic evaluation of a system's utility in practical scenarios (Figure 3).

Testing Environment

The system was deployed in a Dockerized environment to ensure consistency and portability. Test logs simulating real-world distributed system events were streamed into the pipeline. This setup allows evaluation under controlled but realistic conditions.

Metrics Observed

- *Classification accuracy*: The percentage of correctly identified log types and anomalies.
- *Alert latency*: Time elapsed from log ingestion to alert delivery.
- *Throughput*: Number of logs processed per second without performance degradation.
- *Dashboard refresh rate*: Frequency of real-time updates in the frontend.

Classification Accuracy

The AI model achieved approximately 95% accuracy in categorizing logs and detecting anomalies, thereby demonstrating its effectiveness. High recall indicates the successful identification of critical events, while precision reflects minimal false alerts. These results validated the generative AI approach for real-time log analysis.

DISCUSSION

The project successfully demonstrated an AI-enhanced log-monitoring framework that balances accuracy, performance, and usability. Trie-based cache is a key innovation that reduces redundant parsing and enables efficient real-time processing. Integration with popular tools, such as Slack, Prometheus, and Grafana, ensured practical applicability. However, challenges remain, such as model tuning to minimize false positives and handling unstructured logs in diverse formats. The modular architecture facilitates extensibility, allowing the future incorporation of advanced deep learning models or additional data sources. User feedback highlighted the value of real-time alerting as well as the need for customizable alert thresholds. Overall, the system provides a solid foundation for intelligent log management and can be a valuable asset for DevOps and IT operations.

Strengths and Innovations: AI-Driven Parsing

Unlike traditional tools that require a Regex or Grok configuration, the system learns the log structure dynamically using OpenAI's LLM. This eliminates the need for manual rule creation and enables adaptation to new log formats.

- *Semantic understanding*: The AI can extract meaningful fields, such as "component," "severity," and even suggest corrective actions. However, this is not possible with standard pattern-matching methods.
- *Real-time architecture*: The use of FastAPI and WebSockets enables near-instant feedback on new logs and alerts, fulfilling real-time monitoring requirements.
- *Production-ready monitoring stack*: Prometheus and Grafana provide a robust ecosystem for observability, while the alerting logic integrates seamlessly with Slack and email, making the solution DevOps-friendly and highly operational.

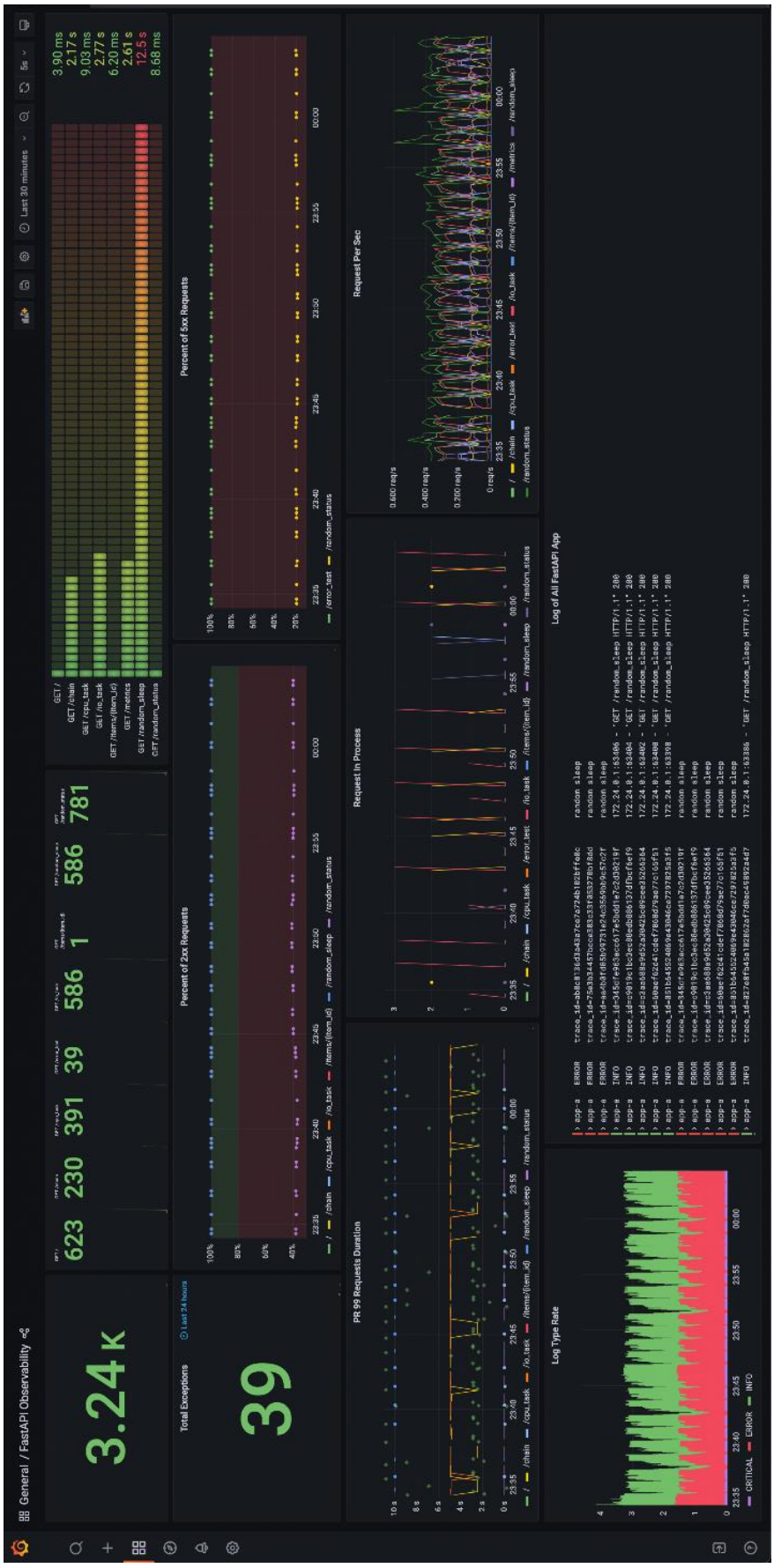

Figure 3. Dashboard Prometheus, Loki, and Grafana [15].

Future Enhancements

Self-hosted LLM integration using open-source models (e.g., LLaMA and Mistral) could improve data security and reduce costs.

Feedback loop to LLM: User-approved corrections as training feedback to fine-tune the parsing quality.

Visualization improvements: More advanced visualizations, such as correlation heatmaps, predictive alerts (using ML), and natural language log search, could be added.

Component-aware parsing builds internal logic maps, logs to specific microservices/components by using trained embeddings.

CONCLUSION

This paper presents a comprehensive AI-powered log-monitoring system designed to address the challenges posed by large-scale log data. By combining advanced parsing, AI-based anomaly detection, real-time alerting, and interactive visualization, the system enhances the efficiency and reliability of the log management. The experimental results validate the effectiveness of the approach in promptly detecting anomalies and handling high data throughput. Although limitations exist, such as dependency on structured logs and potential false alarms, the system sets a precedent for integrating AI into operational monitoring. Future work will include improving model sophistication, expanding log source compatibility, and refining alert mechanisms to reduce noise and improve user trust.

The proposed system leverages LLMs to automatically parse unstructured logs into structured machine-readable formats. By integrating a trie-based adaptive cache, the system optimizes repeated parsing tasks, significantly reducing latency and external API costs. Real-time log updates and visualizations are made possible via a Next.js-based dashboard and WebSocket communication, whereas Prometheus and Grafana form the backbone of the observability and monitoring stack. The system also incorporates alerting mechanisms through Slack and email, enabling prompt incident responses.

The system was evaluated across the functional, performance, and usability dimensions. The results demonstrated strong parsing accuracy, substantial reductions in OpenAI API calls (up to 78% through caching), high system throughput, and positive user feedback on the interface and responsiveness.

REFERENCES

1. Xu W, Huang L, Fox A, Patterson D, Jordan MI. Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09); 2009 Oct 11–14; Big Sky, MT, USA. New York (NY): Association for Computing Machinery; 2009. p. 117–132. doi:10.1145/1629575.1629587.
2. He P, Zhu J, Zheng Z, Lyu MR. Drain: An online log parsing approach with fixed depth tree. 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA. 2017. p. 33–40. doi:10.1109/ICWS.2017.13.
3. Du M, Li F, Zheng G, Srikumar V. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17); 2017 Oct 30–Nov 3; Dallas, TX, USA. New York (NY): Association for Computing Machinery; 2017. p.1285–1298. doi:10.1145/3133956.3134015.
4. Yadav RB, Kumar PS, Dhavale SV. A survey on log anomaly detection using deep learning. 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India. 2020. p. 1215–1220. doi:10.1109/ICRITO48877.2020.9197818.
5. Zhou J, Ying S, Wang S, Zhao D, Xiang J, Liang K, et al. LogDLR: Unsupervised cross-system log anomaly detection through domain-invariant latent representation. IEEE Trans Dependable Secure Comput. 2025;22:4456–4471. doi:10.1109/TDSC.2025.3548050.

6. Zhang X, Xu Y, Lin Q, Qiao B, Zhang H, Dang Y, et al. Robust log-based anomaly detection on unstable log data. In: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019); 2019 Aug 26–30; Tallinn, Estonia. New York (NY): Association for Computing Machinery; 2019. p.807–817. doi:10.1145/3338906.3338931.
7. Pankajashan S, Maragatham G, Kirthiga Devi T. Hybrid approach with deep auto-encoder and optimized LSTM-based deep learning approach to detect anomaly in cloud logs. J Intell Fuzzy Syst. 2022;42(6):6257–6271. doi:10.3233/JIFS-201707.
8. Guo H, Yang J, Liu J, Bai J, Wang B, Li Z, et al. Logformer: A pre-train and tuning pipeline for log anomaly detection. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2024;38:135–143. doi:10.1609/aaai.v38i1.27764.
9. Lou JG, Fu Q, Yang S, Xu Y, Li J. Mining invariants from console logs for system problem detection. In: Proceedings of the 2010 USENIX Annual Technical Conference (USENIX ATC '10); 2010 Jun 23–25; Boston, MA, USA. Berkeley (CA): USENIX Association; 2010. p. 1–14.
10. Alim A, Clegg RG, Mai L, Rupprecht L, Seckler E, Costa P, Pietzuch P, Wolf AL, Sultana N, Crowcroft J, Madhavapeddy A, Moore AW, Mortier R, Koleni M, Oviedo L, McAuley D, Migliavacca M. FLICK: Developing and running application-specific network services. In: Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16); 2016 Jun 22–24; Denver, CO, USA. Berkeley (CA): USENIX Association; 2016. Available from: https://www.usenix.org/conference/atc16/technical-sessions/presentation/alim
11. Kou L, Li Y, Zhang F, Gong X, Hu Y, Yuan Q, et al. Review on monitoring, operation and maintenance of smart offshore wind farms. Sensors. 2022;22(8):2822. doi:10.3390/s22082822.
12. Huang S, Liu Y, Fung C, Wang H, Yang H, Luan Z. Improving log-based anomaly detection by pre-training hierarchical transformers. IEEE Trans Comput. 2023;72(9):2656–2667. doi:10.1109/TC.2023.3257518.
13. OpenAI; Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, et al. GPT-4 technical report. [Preprint]. 2023. arXiv:2303.08774. doi:10.48550/arXiv.2303.08774.
14. Prometheus.io. (2026). Data model. [Online] Prometheus Documentation. Available from: https://prometheus.io/docs/concepts/data_model/
15. Grafana Labs. (2025). Grafana OSS and Enterprise: Query, visualize, alert on, and explore your metrics, logs, and traces. [online] Grafana documentation. Available from: https://grafana.com/docs/grafana/latest/
16. Lubanovic B. FastAPI. Sebastopol (CA): O'Reilly Media Inc.; 2023.
17. Lazuardy MF, Anggraini D. Modern front-end web architectures with React.js and Next.js. Res J. Adv. Eng Sci. 2022;7(1):132–141.
18. D'Amore L, Arcucci R, Mele V, Scotti G, Murli A. Technical documentation L-BFGS for GPU-CUDA reference manual and user's guide. SSRN Electron J. 2013;167. doi:10.2139/ssrn.2332125.
19. Johnson HA. Slack. J Med Libr Assoc. 2018;106(1):148. doi:10.5195/jmla.2018.315.
20. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res. 2011;12:2825–2830.

