Scalable and Efficient Large-Scale Log Analysis with LLMs: An IT Software Support Case Study

Pranjal Gupta[1], **Karan Bhukar**[2*], **Harshit Kumar**[1], **Seema Nagar**[1], **Prateeti Mohapatra**[1], **Debanjana Kar**[1]

[1]IBM Research, India
[2]Amazon, India

Abstract

IT environments typically have logging mechanisms to monitor system health and detect issues. However, the huge volume of generated logs makes manual inspection impractical, highlighting the importance of automated log analysis in IT Software Support. In this paper, we propose a log analytics tool that leverages Large Language Models (LLMs) for log data processing and issue diagnosis, enabling the generation of automated insights and summaries. We further present a novel approach for efficiently running LLMs on CPUs to process massive log volumes in minimal time without compromising output quality. We share the insights and lessons learned from deployment of the tool - in production since March 2024 - scaled across 70 software products, processing over 2000 tickets for issue diagnosis, achieving a time savings of 300+ man hours and an estimated $\$15,444$ per month in manpower costs compared to the traditional practices.

Code — https://github.com/Log-Analyzer/LogAn
Extended version — https://arxiv.org/abs/2511.14803
Demo — https://tinyurl.com/demo-logan (2025a)

Introduction

In an enterprise setting, software issues encountered by customers are reported to the IT support team with two key inputs: a) a log dump that provides system activity, errors, etc, b) problem description - a description of their understanding of the issue, including the observed symptoms. Since log data volumes are huge, ranging from megabytes to gigabytes, automated log analysis tools play a vital role in IT software support by enabling the processing and extraction of key insights from this data (Zhaoxue et al. 2021).

Traditionally, support engineers (SEs) often rely on manual log analysis using the `grep` command to search for terms like "error" or "failure" or use scripts with predefined rules. However, these methods have various limitations: (a) **Coverage**: Predefined rules and simple keyword searches often fail to diagnose issues because of the diversity in log formats and terminology across different systems. (b) **Maintenance**: As software updates and new features are added, log structures often change. This makes predefined rules outdated, requiring constant maintenance. This is particularly problematic in the fast paced CI/CD environments. (c) **False Positives**: Keywords like "error" or "failure" may appear in non-critical contexts, leading to irrelevant entries being flagged and slowing down diagnosis. (d) **Feedback**: Rule-based systems don't evolve, lacking the ability to learn from false positives or false negatives. This results in a static process that doesn't improve. (e) **Correlation and Causality**: Logs are often scattered across services, making manual correlation and understanding event causality difficult, especially in large-scale systems with high log volumes.

To address these limitations, researchers have applied Large Language Models (LLMs) to log analytics, showing promising results and new opportunities. (Almodovar et al. 2024; Liu et al. 2023, 2024; He et al. 2024). Although LLMs excel at generalizing across natural language understanding tasks (Minaee et al. 2024), their high resource and performance costs hinder scalable production deployment (Zhou et al. 2024). In IT Support, this challenge is further exacerbated by the massive scale and heterogeneity of log data across applications. Moreover, resource constraints often necessitate running LLMs on CPUs rather than GPUs, further amplifying the difficulty of achieving efficiency at scale. Consequently, building a unified tool that processes enormous log volumes while enabling LLMs to generalize across diverse application domains remains a critical challenge.

This paper presents an intelligent and scalable log analytics tool that utilises LLMs to overcome the limitations of traditional rule-based approaches, efficiently processing large-scale log data on CPUs in minimal time without compromising overall quality. The LLM predicts three types of insights from log data: (1) Golden Signals, (2) Fault Categories, and (3) Named Entities (Gupta et al. 2023, 2025b). It then generates multiple reports based on the extracted insights, providing different perspectives to assist SEs in issue diagnosis. To enable efficient LLM inference on large scale log data, we introduce Label Broadcasting - a novel technique specifically designed for LLM powered log analysis, which enables our tool to operate entirely on CPU backends.

Notably, our tool doesn't attempt to determine the root cause(s) or remediation, which requires domain knowledge of the underlying application and access to proprietary code or documentation (Saha and Hoi 2022; Chen et al. 2024). Instead, it operates solely on logs to highlight *problematic* log

*This work was done while Karan Bhukar was at IBM.

lines—those containing cues of underlying issues—thereby guiding SEs toward faster, focused diagnosis compared to rule-based analysis.

Our main contributions are: (1) We present the design, implementation and deployment of a scalable log analytics tool that generates insightful reports to reduce information overload for SEs during issue diagnosis. To the best of our knowledge, it is the first system capable of efficiently processing large scale log data using LLMs entirely on CPU while producing reports within practical time limits. (2) We propose Label Broadcasting, a novel technique that enables our log analytics tool to perform LLM inference on large-scale log data, achieving substantial resource savings without compromising inference quality. Experimental results validate its scalability and efficiency in handling high-volume data. (3) We share insights and lessons learned from deploying our tool across 70 IBM Software Support products since March 2024. Additionally, we present a case study with a finance-domain client, demonstrating how the tool aids in effective issue diagnosis, thereby reducing the cognitive load and enhancing the overall productivity.

System Architecture

Analyzing massive volumes of log data is a major challenge. Applying LLMs to every log entry is computationally expensive and slow. To overcome this, our tool comprises three components (refer Figure 1): a) **Preprocessing**: We use log templatization to group similar log lines into a smaller, representative set of clusters. This drastically reduces the data volume. b) **LLM Inferencing**: LLMs analyze the representative log lines to infer necessary labels. These labels are then broadcast to all the log lines within their respective clusters. c) **Report Generation**: The final step is the generation of various reports, including summary, diagnosis, temporal, and causal-graph views. These reports provide concise and actionable insights, allowing SEs to diagnose issues efficiently. The rest of the section details each of the components.

Preprocessing

Log data is often scattered across multiple files, which makes analysis difficult. Our tool addresses this by first consolidating all log lines from different files into a single master file. This process goes beyond simple merging; it ensures that all log lines are sorted chronologically while preserving key information such as original file name, line number, date, timestamp, etc. This is crucial for correctly identifying log entries that span multiple lines. Once consolidated, this unified log data is passed to the log templatizer, which groups similar log lines together, preparing the data for further analysis.

Log Templatisation Our log analytics tool leverages log templatisation (He et al. 2017) to extract dynamic (*variables*) and constant (*templates*) parts of a log line. This component groups similar log lines into one cluster (hereby known as *log cluster*). Below is an example of a log cluster with its corresponding log template (T):

```
T: PacketResponder ⟨*⟩ for block ⟨*⟩ terminating
L1: PacketResponder 0 for block blk_11 terminating
L2: PacketResponder 1 for block blk_12 terminating
L3: PacketResponder 2 for block blk_13 terminating
```

The template (T) captures the syntactic structure of the log data with variables abstracted by wildcards $\langle * \rangle$. Log Templatisation offers many advantages such as bringing structure to data (Mahindru, Kumar, and Bansal 2021) and volume reduction (Pathak et al. 2024).

Representative Set The above example highlights the repetitive nature of log data, where the template of a log line remains consistent and only the value of specific variables changes within a log cluster. Notably, this characteristic may not be present in other natural languages. After templatisation, a log line is randomly selected from each log cluster to form a *representative set*. This reduced set encapsulates the essence of the log dump by clustering similar lines, thereby enabling efficient LLM inference on CPUs, as explained in the following subsection.

LLM Inferencing

For each log line in the representative set, the tool predicts the associated golden signal, fault category, and named entities using fine-tuned LLMs for the log domain. Three task-specific LLMs - for Golden Signal Classification (GSC), fault category prediction (FCP), and Named Entity Recognition (NER) - were finetuned offline on a few manually curated, annotated multi-domain (network, SSH, Linux, etc.) log lines in a few-shot learning setup. (Gupta et al. 2023, 2025b). During runtime, the finetuned models are used directly to infer predictions for incoming logs. Note that the training dataset used for fine-tuning the LLMs does not include logs from products deployed in production. As a result, logs from deployed products are often unseen and entirely new to both the log analytics tool and the fine-tuned LLMs. Despite this limited exposure, the LLMs accurately classify logs from unseen domains, demonstrating strong generalizability and robustness (refer Appendix A).

Label Broadcasting To scale LLM inference for the three tasks on large-scale log data, our log analytics tool performs inference only on a representative set of the log dump. The inferred labels - including golden signals, fault categories and named entities are *broadcasted* back to all the log lines within their respective log clusters. This Label Broadcasting (LB) approach reduces the number of LLM calls required, whereas conventional LLM inference optimization techniques (Donisch, Schacht, and Lanquillon 2024) still invoke the LLM for every log line. LB exploits the repetitive nature of log data, while such inference optimization techniques are domain-agnostic. Moreover, LB can be combined with these techniques to achieve even further inference speedups. In production, our tool uses the LB to perform LLM inference on large-scale log data entirely on CPUs.

Enriched Logs After broadcasting the inferred labels to all log lines in the input dump, we obtain *enriched logs*, where each line is annotated with its corresponding golden

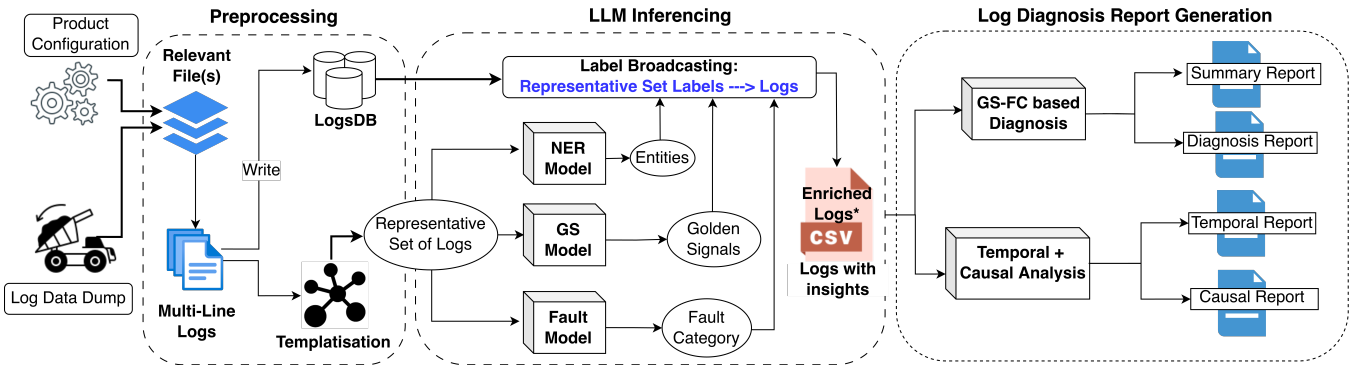


Figure 1: System Architecture of the proposed log analytics tool

signal, fault categories, and named entities. These enriched logs are further processed to generate various reports/views.

Log Diagnosis Report Generation

Following LLM inference, our log analytics tool generates the following diagnostic reports based on insights from enriched logs.

Summary Report presents a table of the *enriched* representative log lines (refer Figure 4) — each with its predicted golden signals, fault categories, and named entities — along with the frequency of its occurrence in increasing order. The rarest one are the most important one and needs attention and therefore at the top. By using this approach, we've found that the tool can reduce the data volume by up to 90%, since most log lines are informational. For easier analysis, one can filter the report by golden signals, fault categories, or named entities, which are color-coded for better readability.

Diagnosis Report presents a chronologically ordered set of relevant log windows with user-configurable granularity (e.g., 30s, 1m). A window is deemed relevant if it contains at least one erroneous or faulty signal identified through GSC or FCP via LLM inference. While conventional methods indiscriminately display the complete log dataset, the Diagnosis Report shows only log windows relevant to the issue. By inspecting this report, SEs can conduct pre/post analysis for issue diagnosis. Additionally, it offers a keyword-based search interface on named entities, providing a more effective alternative to manual grep commands or predefined rules. For example, logs can be filtered based on customer-provided problem descriptions or entity types such as ErrorCode. (refer Figure 11 in Appendix D)

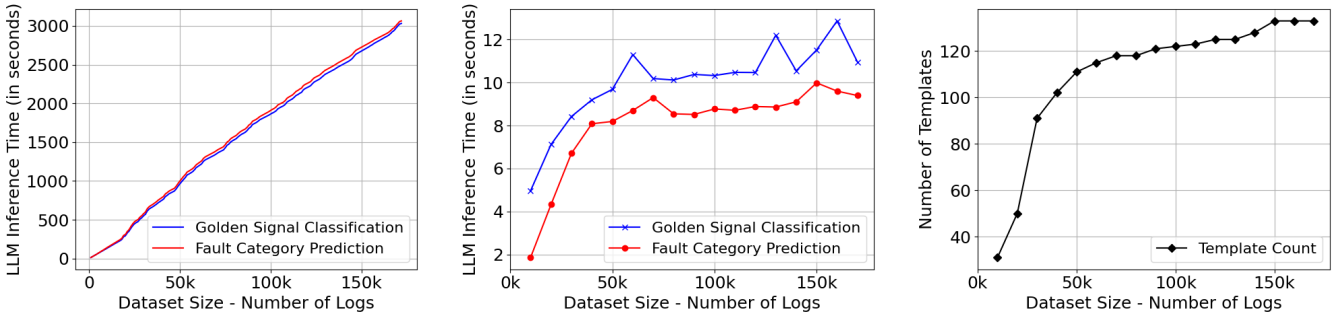
Temporal Trend Since log dumps may span from days to months or even years, identifying the exact date and time of an issue often requires considerable effort, especially when such information is not explicitly provided. To mitigate this challenge, the Temporal Trend report (Figure 5) visualizes the distribution of golden signals over time, thereby helping SEs track the fault progression, revealing their characteristics and severity. By analyzing spikes in erroneous signals, they can quickly understand the issue's progression and the interplay of contributing factors.

Causal Graph View Modern IT systems comprise numerous interdependent services, where a single underlying issue can trigger a cascade of related failures. These failures often manifest as disparate log lines scattered across different components, making manual issue diagnosis cognitively demanding for SEs. To overcome this, our log analytics tool builds a causal graph that captures the cause-effect relationships among offending/problematic representative log lines. It captures the sequence of frequently co-occurring errors, enabling SEs to prioritize investigation and trace the origins of cascading faults. To build the causal graph, our log analytics tool utilizes enriched logs (via LLM inference) along with their associated log clusters (refer Log Templatisation section). A multi-variate time series is then created, representing the count occurrence of each log template within a defined time interval window. We apply a statistical causal inference method based on Granger causality (Granger 1969) on the time-series data to detect directional influence among log clusters. The resulting causal graph contains nodes representing log clusters and directed edges denoting inferred causal relationships (Figure 6).

Tool Usage Workflow An effective workflow of our tool begins with the Summary Report, which offers a tabular view of representative log lines annotated with golden signals, fault categories, and entities. The Temporal Trend guides in identifying the period of fault manifestation, followed by the Causal Graph to identify cause-effect relationships, which prioritizes log lines for investigation. Finally, the Diagnosis Report presents relevant log windows within the selected time range for in-depth analysis. This workflow streamlines issue diagnosis, reducing information overload and improving overall productivity.

Experiments

Our log analytics tool introduces two contributions: (1) **Templatisation with Label Broadcasting (LB):** An efficient and scalable LLM inference technique that runs entirely on CPUs and enriches logs with predicted labels from three tasks - Golden Signal Classification, Fault Category Prediction and Named Entity Recognition, and (2) **Report Generation:** which leverages the enriched logs to produce reports that reduce the cognitive load on SEs.



(a) Tool w/o Label Broadcasting (b) Tool w/ Label Broadcasting (c) Dataset vs Representation Set Size

Figure 2: Label Broadcasting: Computation Time Analysis

To assess the effectiveness of LB, we address the following research questions: **RQ1:** How effective is LB in reducing computation time during LLM Inference? **RQ2:** Does LB affect the LLM inference quality? To assess the practical utility of the reports generated by our log analytics tool, we present findings from a real case study.

Dataset Overview We evaluate our approach on logs from four real-world software applications (D1, D2, D3, D4) spanning the domains of data analytics, security, inventory management, and finance. More details on the datasets and statistics are provided in Appendix A.3.

As a baseline, we use the traditional method that performs LLM inference on each log line, referred to as **Tool w/o LB** and compare it with our approach (**Tool w/ LB**).

LLM Selection and Rationale BERTOps (Gupta et al. 2023) is an LLM pretrained on logs that has shown strong performance on log analysis tasks in few-shot learning setups. In this study, we adopt BERTOps and evaluate it on the above three tasks. Our results validate that it outperforms competing methods, generalizes well when trained on examples from multiple domains, and remains robust when applied to logs from unseen domains. These findings justify its integration into our tool (refer Appendix A.5).

Results and Discussion

RQ1: To analyse our log analytics tool's efficiency, we analyse the computation time required to process log datasets of varying sizes using a Virtual Machine (VM) with 48 cores and 189GB RAM, without GPUs. Table 1 reports the computation time of our log analytics tool to process a dataset of $170K$ log lines from the D3 Domain (refer Appendix A.3 for details). Our approach, Tool w/ LB, yields a significant reduction of $\sim$ 99.7% in inference time compared to the traditional approach of Tool w/o LB.

Figure 2a and 2b illustrate the computation time for inferring Golden Signals (GS) and Fault Category (FC) labels as the dataset size increases in increments of $10K$ loglines. As depicted in Figure 2a, the computation time for Tool w/o LB increases linearly, which is expected since it infers a label for each log line. In contrast, Figure 2b shows that Tool w/ LB maintains significantly lower processing times for GS and

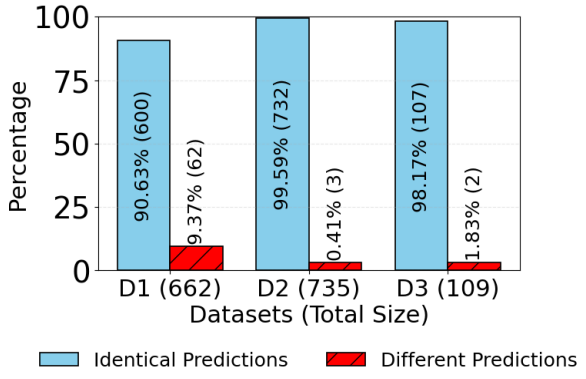
Method	#Data Size	Time Taken (in seconds)		
		Templatization	GSC	FCP
Tool w/o LB	170k	31.1s	3031.1s	3063.6s
Tool w/ LB			**10.95s**	**9.39s**

Table 1: Influence of Label Broadcasting (LB)

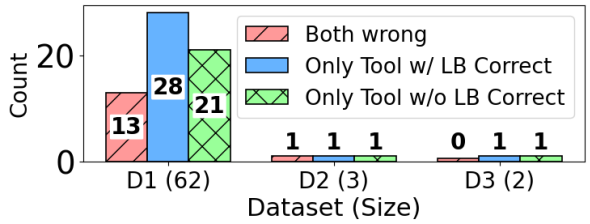
FC label computation, with a maximum of only 10 seconds. Initially, for Tool w/ LB, computation time increases as the dataset grows. This is because for each new template discovery by the log templatizer, GS and FC labels are predicted by invoking the LLM, resulting in a steep increase in computation time. However, after processing approximately $50K$ log lines, the computation time plateaus. This is because fewer new templates are identified beyond this point (as shown in Figure 2c), allowing our tool to retrieve GS and FC labels via a simple lookup rather than invoking the LLM model. These results indicate that Tool w/ LB achieves substantial resource savings during LLM inference, empirically validating its ability to scale efficiently and handle large-scale log data with minimal increase in computation time.

RQ2: To evaluate the effect of Label Broadcasting (LB) on LLM inference quality, we compare the predictions generated by the two methods - Tool w/ LB and Tool w/o LB - across the three datasets (D1, D2, and D3). Figure 3a presents the comparison results. Across all the datasets, the two methods produce identical outputs for the majority of the cases. For datasets D2 and D3, over 98% of log lines yield the same inference output under both methods. This indicates that applying Tool w/ LB doesn't alter the outcome compared to Tool w/o LB, thereby leaving LLM inference quality unaffected for these cases. The figure also highlights the small proportion of cases where Tool w/ LB and Tool w/o LB produce different predictions. For D2 and D3, such instances are extremely rare, containing only $3/735$ and $2/109$ examples, respectively. In contrast, dataset D1 is an exception, with predictions differing in 9.37% of the cases.

Figure 3b further analyses these differing cases in D1 by comparing each method's predictions against the ground



(a) Statistics: Identical vs Different Predictions



(b) Different Prediction Cases: Tool w/ LB & Tool w/o LB

Figure 3: Label Broadcasting: Output Comparison Analysis

truth, resulting in three categories: (1) **Both wrong:** In 2% of cases (13 examples), both methods produce incorrect predictions, indicating that the model fails to generate the correct label regardless of the approach. Here, Tool w/ LB is preferable as it reduces the number of LLM invocation calls. (2) **Only Tool w/ LB correct:** In 4.2% of cases (28 examples), Tool w/ LB predicts the correct label, while Tool w/o LB is incorrect. In these cases, Tool w/ LB not only improves LLM inference quality but also reduces resource usage by minimizing LLM calls. (3) **Only Tool w/o LB correct:** In 3.2% of cases (21 examples), Tool w/o LB predicts the correct label while Tool w/ LB is incorrect. Examples from this category are shown in Table 9 (refer Appendix), suggesting that variables in log lines offer *important* cues that help the LLM make accurate predictions. For such rare cases, performing inference on each log line individually is advantageous, as Tool w/ LB can negatively impact prediction quality. Nevertheless, in the remaining 96.8% of cases, Tool w/ LB yielded the *preferred* outputs.

As supported by our experiments and prior work (Gupta et al. 2024), the occurrence of such *important* variables is rare. Since Tool w/ LB produces *preferred* predictions for nearly 96% of log lines, the relatively rare cases in category 3 are considered an acceptable operational trade-off. Achieving perfect accuracy for these few instances would require GPU-based inference on every log line, which would significantly increase resource requirements. In contrast, Tool w/ LB enables our log analytics tool to operate entirely on CPU, greatly reducing its memory footprint while scaling efficiently to handle large datasets across multiple products. Hence, our experimental results validate that Label Broadcasting can be employed for LLM inference for logs without degrading overall prediction quality.

Case Study

To assess the practical utility of our log analytics tool, we present a real-world case study from an application in the financial domain (D4). The client provided log data from July 3–5, 2023, consisting of 425,000 log lines. The client reported that the application was encountering issues, resulting in unexpected terminations, but they were unclear how to go about diagnosing it. Their request made to us was: *"We need a way to correlate and make sense of this data, at least to reduce the cognitive load on our subject matter experts (SMEs) and provide actionable insights. We need conclusive, data-driven evidence for such correlations."* The following discussion highlights the effectiveness of our tool in debugging and diagnosing issues.

(a) Summary Report: Our tool reduced the dataset from 425,000 log lines to a representative set of just 74 log lines, achieving a reduction of 99.9%. The golden signals distribution within this set was: {*error:18, availability:9, latency:5, saturation:2, information: 40*}. Since the log lines under *information* signal mainly contain debugging context, these can be safely ignored (Gupta et al. 2023) for initial issue diagnosis. Among the remaining log lines, the high count of error and availability signals highlight them as prominent concerns. Figure 4 shows a snapshot of the summary report with its color-coded entities, golden signal and fault categories. To illustrate its utility, we detail the log lines by explicitly indicating which log analysis task extracted the corresponding information (shown in brackets). For instance, Template ID (TID) 15: "Session ID XX does not exist" (NER) correspond to an "availability" issue (GSC) at the "application" level (FCP). Similarly, for TID 17: "Operation Timed out" (NER) resulting in "latency" issue (GSC) at the "network" stack (FCP).

Moreover, our tool also extracted critical attributes such as host/application name and process/session identifiers. For example, TID 23, "Service Failure" caused by "invalid argument" for "Host XX, PID YY, SessionID ZZ" (NER) is categorized as an "error" (GSC) at the "application and I/O" level (FCP). This significantly reduces the cognitive load on SEs by highlighting key entities that would otherwise require manual search, thus improving productivity.

(b) Temporal Trend View: Figure 5 presents the hourly temporal trend of various golden signals in the log data, tracking issue progression and its manifestation. For instance, the trend indicates a sharp escalation after July 4th (vertical black line), when the client system's instability became more pronounced. Before July 4th, there were only a few errors and availability events. However, these early signs of trouble gradually intensified after July 4th, leading to increased occurrences of errors, availability and latency events, signalling a broader system degradation.

(c) Causal Graph View: Figure 6 depicts the causal graph view generated by the tool, where the graph represents the cause-effect relationships among different Template IDs (TIDs). In the UI, hovering over a node reveals the corresponding representative log line for that TID, with

Template ID	Representative Logline	Golden Signal (All)	Fault Category
15	**[2023-07-04 13:24:43 GMT]$_{Timestamp}$** WARN **[ssm.ssmcore.ARM]$_{Component}$** - Code[S32009]: ssmARM.cpp:7731 **[Domain <SOAM>: Session ID XX]$_{KV\ Pair}$** does not exist. Check your session ID and try again	availability	['application']
17	2023-07-04 12:24:59.938 GMT WARN **[ssm.ssmcore.ARM]$_{Component}$** ARM::onBlockHostHelper **[Operation timed out in method call.]$_{Cause}$**	latency	['network']
23	.... GMT WARN **[ssm.ssmcore.ARM]$_{Component}$** - Domain <Application>: **[Service failure in method onSessionEnter():]$_{Cause}$** Python exception [<type 'exceptions.IOError'> (22, 'Invalid argument') .... **[Host: XX PID: YY]$_{KeyValue}$**: Failed **[host:XX, PID: YY, SessionID: ZZ]$_{KeyValue}$**	error	['application', 'I/O']

Figure 4: Summary Report showing representative log lines along with their golden signal, fault categories and named entities.

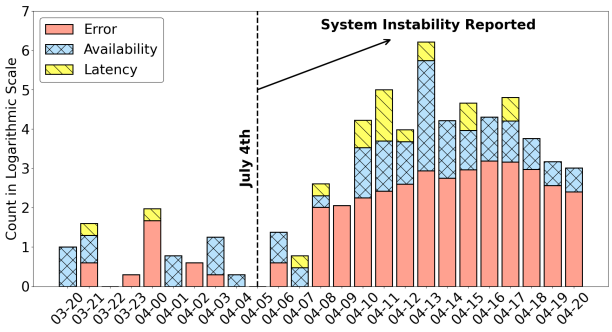


Figure 5: Temporal View of Fault Progression (X axis: Date in DD-HH format)

Template ID	Corresponding Log Line (with extracted entities in bold)
29	**[soam.EventAgent]$_{component}$ [Code [S75017]]$_{ThreadID}$ [Application <X>]$_{KeyValue}$** is blocked due to service **[instance <SVC4>]$_{KeyValue}$** with **[process ID <A>]$_{KeyValue}$** Reason: **[Operation timed out in method call]$_{Cause}$**
16	**[sim.simcore.ServiceBroker]$_{component}$** - **[Code [S60083]]$_{ThreadID}$** Service: **[Session<X> service <A>]$_{KeyValue}$ [failed due to the session being suspended or resource is being reclaimed]$_{Cause}$**
10	**[soam.common.EventAgent]$_{component}$ [Code[S75017]]$_{ThreadID}$ [Application: <X> session: <id: Y name:A>]$_{KeyValue}$ [aborted on host <B> with reason <connection broken>]$_{Cause}$**
Causal Graph: (Cause ⟶ Effect)	29 ⟶ 16 ⟶ 10

Figure 6: Causal Graph View

the relevant entities highlighted in bold. From the graph in Figure 6, it is evident that the tool identified cause–effect relations among three nodes. A closer inspection highlights the following insights: (1) TID 29: "Operation Timed Out" (NER) associated with "latency" issue (GSC). (2) TID 16: "Session being suspended" (NER) associated with "error" (GSC), and (3) TID 10: "Connection Broken" (NER) associated with "error" (GSC) . A plausible explanation for this correlation is that the latency observed in TID 29 leads to a suspended session in TID 16, which eventually results in the broken connection error in TID 10. The graph thus reveals connections among logs that might otherwise appear unrelated. This capability enables SEs to identify hidden patterns for more effective issue diagnosis.

In essence, the Summary Report presented diverse insights extracted via LLM inference while reducing the log data volume by 99.9%. The Temporal Trend indicates that system degradation began around July 4th, whereas the Causal Graph highlights latency as a prevalent concern. Building on these insights, SEs can leverage the Diagnosis Report (Figure 11 in Appendix D) to focus only on the *relevant* log windows and conduct deeper analysis on the *problematic* log lines identified by the other reports. These findings, along with explanations and reasoning, were shared with the client, who appreciated the tool's contributions for effective issue diagnosis.

Lessons Learned From Deployment

This section outlines the system design of our log analytics tool's deployment pipeline and shares key insights and lessons learned from its adoption across 70 IBM Software Support products.

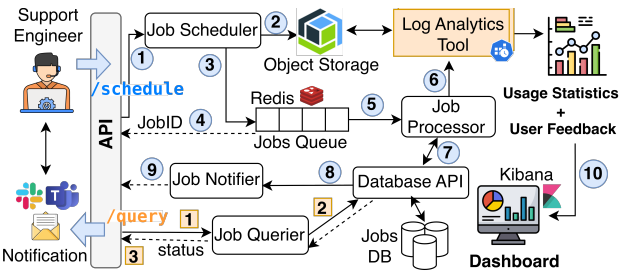
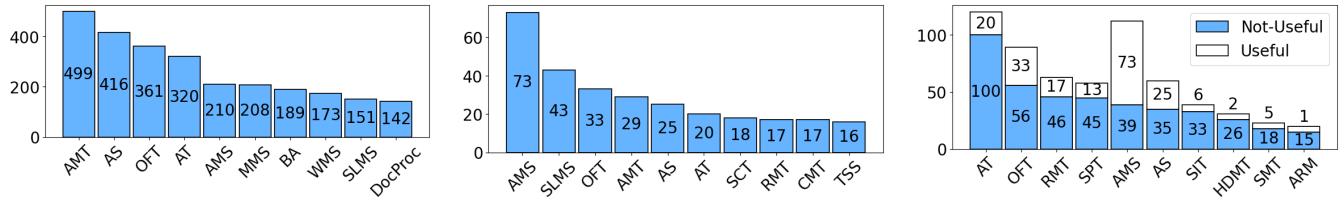


Figure 7: System Design: Workflow of `/schedule` API (blue, circle, 1–10) and `/query` API (orange, square, 1–3)

System Design

Figure 7 depicts the microservices-based architecture of our log analytics tool's pipeline. A SE uploads a log dump (hereby known as *job*), via the `/schedule` API endpoint. The Job Scheduler assigns a unique *job_id* to each uploaded log dump and stores the data in Cloud Object Storage (COS). Then the *job_id* is placed in the Job Queue, and the user is notified that the job has been scheduled. When the job is ready for execution, the Job Processor retrieves the job ID, launches an independent instance of our log analytics tool, and begins processing. Multiple jobs are processed concurrently, with their statuses tracked in a database (Jobs DB). Users receive status updates via configured channels such as Slack or Microsoft Teams, or by querying the `/query` endpoint. Additionally, runtime and usage statistics—such as total processing time, number of files, and data size—are



(a) Top 10 Products Based on Usage (X: Products, Y: Count)

(b) Top 10 Products Benefiting Most from the Tool (X: Products, Y: Count)

(c) Top 10 Products that didn't find the Tool useful (X: Products, Y: Count)

Figure 8: **Summary of Product Usage with the Log Analytics Tool.** (Acronyms[1]: AMS: Application Monitoring Solution, AMT: Asset Management Tool, ARM: Application Resource Management, AS: Application Server, AT: Automation Tool, BA: Business Automation, CMT: Content Management Tool, DocProc: Document Processing Software, HDMT: Device Management Tool, MMS: Microservice Management Tool, OFT: Order Fulfillment Tool, RMT: Requirements Management Tool, SCT: Software Collaboration Tool, SIT: Software Integration Tool, SPT: System Performance Tool, SLMS: Software Lifecycle Management Solution, SMT: System Monitoring Tool, TSS: Test Suite Software, WMS: Workplace Management System.)

stored in Elasticsearch and visualized through a Kibana dashboard, providing stakeholders with insights for monitoring and iterative improvement.

Lessons Learned - Deployment

Since March 2024, the log analytics tool has been deployed in production, with 70 products onboarded to leverage it for analyzing customer tickets containing log data. Figure 8a

Figure 9: Distribution of time savings achieved with our tool.

highlights the top 10 products that used the tool for log analysis. The number of times the tool was triggered per product ranges from a minimum of 1 to a maximum of 499 times, with an average of 69 times. Over the 15 months, the tool has been invoked $4,227$ times across $2,394$ unique tickets, processing approximately 1.49 TB of data and 2.76 billion log lines. On average, each ticket involved a 653 MB log dump and approximately 1 million log lines. The tool has spent an average of 50 minutes processing each ticket end-to-end (refer Production Usage Dashboard in Appendix B).

To assess the time savings from using the tool, we solicited feedback from SEs using a brief three-question survey at the end of each session:

- Q1. Was the final output useful? (Yes/No)
- Q2. How much time did the tool save? a) No Savings, b) $1-5$ minutes, c) $6-15$ minutes, d) More than 30 minutes
- Q3. Open-ended feedback: What additional features or improvements would make the tool more useful?

The first question assesses whether the tool aided SEs in issue diagnoses. The second evaluates whether the tool contributed to time savings. The third question elicits user feedback to identify shortcomings and gather feature requests. This structured feedback approach enabled us to measure the tool's impact and identify opportunities for improvement.

As the feedback was optional, we received responses for 565 tickets, representing 23.51% of the total tickets. Of these responses, 46.72% of the respondents found the tool useful. Some of the positive and negative feedback, along with feature requests, are illustrated in Appendix C. The top 10 products that received positive feedback are illustrated in Figure 8b. These products resulted in savings of 316 man hours, with the distribution in terms of savings illustrated in Figure 9. A major portion is attributed to savings of more than 30 minutes.

Among the 53.24% of the population that did not find the tool useful, the respective top 10 product distribution is illustrated in Figure 8c. The reason for equally high negative sentiment can be attributed to negativity bias. Individuals are generally more likely to provide feedback after negative experiences than positive ones, a tendency explained by negativity bias (Baumeister et al. 2001; Anderson 1998; Oliver 1980), where adverse events exert greater cognitive and behavioral impact. As a result, feedback may overrepresent negative sentiments, potentially skewing perceptions of a tool's effectiveness. In this case, however, the gap between negative and positive feedback is small, suggesting that many users valued and found the tool useful.

To further analyze negative feedback, we identified the top 10 products where users found the tool not useful. For these products, we also calculated how often it was marked useful. Figure 8c shows that in 9 of 10 cases, "not useful" counts exceed "useful," indicating consistent usability or applicability issues. This pattern indicates a possible misalignment between the tool's capabilities and the logging patterns or diagnostic needs of these products. AMS is a positive outlier, as more people find it useful than not. For all the prod-

[1]Product names are obfuscated.

ucts illustrated in Figure 8c, we manually reviewed some of the tickets for each product and examined their log dumps. Also, we interviewed the SEs who worked on those tickets, and based on their responses to Q3, the key reasons are:

1. Some users, especially those who are domain experts, believe that when provided with the log data and problem description, they are better equipped to manually filter related files to diagnose the underlying issue.
2. Expert users felt that the tool processing was excessive; given their expertise in handling such cases, they could manually diagnose the issue in the same amount of time.
3. We found that for three products - Software Integration Tool (SIT), Application Resource Management (ARM), and Order Fulfilment Tool (OFT) - the log dumps were significantly large, ranging from several tens of gigabytes. As these are microservice-based applications, they generate multiple files, one for each pod and service. Processing such a large volume of files demands considerable time and resources, slowing the overall log analytics pipeline and, in some cases, causing it to crash entirely.
4. Manual investigation of log dumps provided further insights into data quality. For the product Hardware Device Management Tool (HDMT), the logs were mostly related to hardware (disk), very different from the kind of data that the models were trained on. The log analytics tool is appropriate for software application logs and not hardware logs. Similarly, logs from mainframe-based applications were not very effective for the tool to process.
5. Another aspect of the quality of log data that affected the efficacy of the log analytics tool is related to the presence of JSON objects embedded with textual data. Such kind of intermingling of JSON objects with textual data jeopardises the templatization algorithm, leading to too many clusters, and gains from label broadcasting are not efficient, thereby slowing the entire pipeline.

In summary, over a 15-month deployment in production, the tool has scaled to 70 products, efficiently processing $2,394$ tickets, with an average ticket size of 700 MB, resulting in savings of 316 man-hours. Our novel approach demonstrates the efficiency and effectiveness of processing large volumes of log data with LLMs in a CPU-constrained environment. Usage patterns suggest two main future directions: (a) processing logs more efficiently, and (b) better handling of JSONs embedded in logs to improve log templates quality and processing speed.

Related Works

This section presents the related work in two directions:

(i) Use of LLMs in log-related tasks: Recent works have used LLMs to perform various log-related tasks. Some studies used LLMs for log anomaly detection (Almodovar et al. 2024; Liu et al. 2023, 2024; He et al. 2024). Almodovar et al. continually pretrain RoBERTa with log data for anomaly detection (Almodovar et al. 2024). Liu et al. conducted a case study on logs and found that the anomalies detected by ChatGPT were partially aligned with those identified by on-call engineers (Liu et al. 2023). Other studies have leveraged LLMs for log parsing (Ma et al. 2024; Zhong et al. 2024; Xu et al. 2024; Jiang et al. 2024b,a; Ma, Kim, and Chen 2025; Huang et al. 2025). Ma et al. proposed an LLM-based log parser based on generative LLMs and few-shot tuning (Ma et al. 2024). Zhong et al. leveraged LLMs for log parsing, merging syntactic and semantic insights (Zhong et al. 2024). Most of these papers utilise decoder models, and none examine case studies on how such models are deployed in real-world production environments with employing CPUs for inferencing. Most recently, Liu et. al (Liu et al. 2025) introduced an instruction-tuned model for diverse log analysis tasks into a single, unified instruction-response format. The fine-tuned model on LLaMA2-7B is deployed within a software and network platform of Huawei. However, no further analysis on generalizability, user feedback, time complexity or productivity has been reported.

(ii) Enterprise-grade frameworks that provide log analysis and discovery capabilities: Other works (Duan et al. 2024; Cheng et al. 2023) and several prominent products in the community (including New Relic (Relic 2024), Dynatrace (Dynatrace 2024), DataDog (Datadog 2024), Instana (Instana 2024) offer comprehensive log monitoring platforms and frameworks. However, none of these works currently integrate LLMs into their frameworks at scale. Also, while these tools share similar functionalities, we focus on a detailed comparison with one well-known product, New Relic, to illustrate the distinct advantages of our log analytics tool. NewRelic displays log streams generated by applications, highlights basic entities in the log lines such as service names and log levels, shows the temporal evolution of logs, and detects anomalies, but doesn't explain the cause and has no human-readable summaries generated. The proposed log anlaytics tool, on the other hand, displays relevant *windows* of correlated log streams (Diagnosis View); highlights complex entity types such as cause/symptom phrases, process ID, component, etc. (through Summary View); shows a more comprehensive temporal evolution which includes insights from the golden signal, and fault categories, correlates logs from different microservices to derive causal relations among log lines which helps in effective and efficient issue diagnosis.

Conclusions and Future Work

This paper presents a log analytics tool that use Large Language Models for log analysis and causal graph mining from log data. One of the defining characteristics of the tool is that it can use LLM to process log data originating from disparate software applications, in time defined manner, with minimum resource consumption - running on CPU, and yet not compromising on the final output. Over 15 months in production, our tool scaled across 70 products, processing 2,394 tickets (avg. 700 MB), saving 316 man-hours. Results show the effectiveness of our LLM-based approach in CPU-constrained environments, with future work focusing on a more efficient and faster way of preprocessing logs.

References

Almodovar, C.; Sabrina, F.; Karimi, S.; and Azad, S. 2024. LogFiT: Log Anomaly Detection Using Fine-Tuned Language Models. *IEEE Transactions on Network and Service Management*, 21(2): 1715–1723.

Anderson, E. W. 1998. Customer satisfaction and word of mouth. *Journal of service research*, 1(1): 5–17.

Baumeister, R. F.; Bratslavsky, E.; Finkenauer, C.; and Vohs, K. D. 2001. Bad is stronger than good. *Review of general psychology*, 5(4): 323–370.

Chen, J.; Qian, J.; Zhang, X.; and Song, Z. 2024. Root-KGD: A Novel Framework for Root Cause Diagnosis Based on Knowledge Graph and Industrial Data. arXiv:2406.13664.

Cheng, Q.; Saha, A.; Yang, W.; Liu, C.; Sahoo, D.; and Hoi, S. 2023. LogAI: A Library for Log Analytics and Intelligence. arXiv:2301.13415.

Datadog. 2024. Log Management. https://docs.datadoghq.com/logs/. Accessed: Oct 4, 2024.

Donisch, L.; Schacht, S.; and Lanquillon, C. 2024. Inference Optimizations for Large Language Models: Effects, Challenges, and Practical Considerations. arXiv:2408.03130.

Duan, Y.; Bao, H.; Bai, G.; Wei, Y.; Xue, K.; You, Z.; Zhang, Y.; Liu, B.; Chen, J.; Wang, S.; and Ou, Z. 2024. Learning to Diagnose: Meta-Learning for Efficient Adaptation in Few-Shot AIOps Scenarios. *Electronics*, 13(11).

Dynatrace. 2024. Dynatrace: Unified observability and security. Accessed: 2024-10-14.

Granger, C. W. 1969. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: journal of the Econometric Society*, 424–438.

Gupta, P.; Bhukar, K.; Kumar, H.; Nagar, S.; Mohapatra, P.; and Kar, D. 2025a. LogAn: An LLM-Based Log Analytics Tool with Causal Inferencing. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering*, ICPE '25, 54–56. New York, NY, USA: Association for Computing Machinery. ISBN 9798400711305.

Gupta, P.; Kar, D.; Kumar, H.; Mohapatra, P.; and Aggarwal, P. 2025b. Few-Shot Learning for Structure Extraction from Heterogeneous Log Data. In *2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)*, 694–702.

Gupta, P.; Kumar, H.; Kar, D.; Bhukar, K.; Aggarwal, P.; and Mohapatra, P. 2023. Learning Representations on Logs for AIOps. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, 155–166.

Gupta, P.; Mohapatra, P.; Kar, D.; Nagar, S.; Ahn, J.-w.; Paradkar, A.; and Srivatsa, M. 2024. Decoding Logs for Automatic Metric Identification. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 336–343.

He, M.; Jia, T.; Duan, C.; Cai, H.; Li, Y.; and Huang, G. 2024. LLMeLog: An Approach for Anomaly Detection based on LLM-enriched Log Events. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, 132–143.

He, P.; Zhu, J.; Zheng, Z.; and Lyu, M. R. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, 33–40. IEEE.

Huang, J.; Jiang, Z.; Chen, Z.; and Lyu, M. 2025. No More Labelled Examples? An Unsupervised Log Parser with LLMs. *Proc. ACM Softw. Eng.*, 2(FSE).

Instana. 2024. IBM Instana Observability. Accessed: 2024-10-14.

Jiang, Z.; Liu, J.; Chen, Z.; Li, Y.; Huang, J.; Huo, Y.; He, P.; Gu, J.; and Lyu, M. R. 2024a. LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. *Proc. ACM Softw. Eng.*, 1(FSE).

Jiang, Z.; Liu, J.; Huang, J.; Li, Y.; Huo, Y.; Gu, J.; Chen, Z.; Zhu, J.; and Lyu, M. R. 2024b. A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We? In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, 223–234. New York, NY, USA: Association for Computing Machinery. ISBN 9798400706127.

Liu, J.; Huang, J.; Huo, Y.; Jiang, Z.; Gu, J.; Chen, Z.; Feng, C.; Yan, M.; and Lyu, M. 2023. Scalable and Adaptive Log-based Anomaly Detection with Expert in the Loop.

Liu, Y.; Li, Z.; Li, Y.; Ren, J.; Zhang, Z.; Ren, J.; Lin, M.; Su, J.; Wang, Y.; Chen, S.; Li, X.; and Zhang, F. 2025. LogLM: From Task-based to Instruction-based Automated Log Analysis. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*. ACM.

Liu, Y.; Tao, S.; Meng, W.; Wang, J.; Ma, W.; Chen, Y.; Zhao, Y.; Yang, H.; and Jiang, Y. 2024. Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies. ICPC '24, 35–46. New York, NY, USA: Association for Computing Machinery. ISBN 9798400705861.

Ma, Z.; Chen, A. R.; Kim, D. J.; Chen, T.-H. P.; and Wang, S. 2024. LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing . In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 1209–1221. Los Alamitos, CA, USA: IEEE Computer Society.

Ma, Z.; Kim, D.; and Chen, T.-H. P. 2025. LibreLog: Accurate and Efficient Unsupervised Log Parsing Using Open-Source Large Language Models. In *Proceedings of the 47th International Conference on Software Engineering*, 924–936.

Mahindru, R.; Kumar, H.; and Bansal, S. 2021. Log anomaly to resolution: Ai based proactive incident remediation. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1353–1357. IEEE.

Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; and Gao, J. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Oliver, R. L. 1980. A cognitive model of the antecedents and consequences of satisfaction decisions. *Journal of marketing research*, 17(4): 460–469.

Pathak, D.; Verma, M.; Chakraborty, A.; and Kumar, H. 2024. Self Adjusting Log Observability for Cloud Native

Applications. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 482–493. IEEE.

Relic, N. 2024. New Relic: Intelligent Observability. Accessed: 2024-10-14.

Saha, A.; and Hoi, S. C. H. 2022. Mining root cause knowledge from cloud service incident investigations for AIOps. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE '22, 197–206. ACM.

Xu, J.; Yang, R.; Huo, Y.; Zhang, C.; and He, P. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 2457–2468.

Zhaoxue, J.; Tong, L.; Zhenguo, Z.; Jingguo, G.; Junling, Y.; and Liangxiong, L. 2021. A survey on log research of aiops: Methods and trends. *Mobile Networks and Applications*, 26(6): 2353–2364.

Zhong, A.; Mo, D.; Liu, G.; Liu, J.; Lu, Q.; Zhou, Q.; Wu, J.; Li, Q.; and Wen, Q. 2024. LogParser-LLM: Advancing Efficient Log Parsing with Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, 4559–4570. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704901.

Zhou, Z.; Ning, X.; Hong, K.; Fu, T.; Xu, J.; Li, S.; Lou, Y.; Wang, L.; Yuan, Z.; Li, X.; Yan, S.; Dai, G.; Zhang, X.-P.; Dong, Y.; and Wang, Y. 2024. A Survey on Efficient Inference for Large Language Models. arXiv:2404.14294.