LogBabylon: A Unified Framework for Cross-Log File Integration and Analysis

Yang Lu*
University of Houston
Houston, United States
ylu17@central.uh.edu

Rabimba Karanjai*
University of Houston
Houston, United States
rabimba@cs.uh.edu

Dana Alsagheer
University of Houston
Houston, United States

Keshav Kasichainula
University of Houston
Houston, United States

Lei Xu
Kent State University
Kent State, United States

Weidong Shi
University of Houston
Houston, United States

Shou-Hsuan Stephen Huang
University of Houston
Houston, United States

Abstract

Logs are critical resources that record events, activities, or messages produced by software applications, operating systems, servers, and network devices. However, consolidating the heterogeneous logs and cross-referencing them is challenging and complicated. Manually analyzing the log data is time-consuming and prone to errors. LogBabylon is a centralized log data consolidating solution that leverages Large Language Models (LLMs) integrated with Retrieval-Augmented Generation (RAG) technology. LogBabylon interprets the log data in a human-readable way and adds insight analysis of the system performance and anomaly alerts. It provides a paramount view of the system landscape, enabling proactive management and rapid incident response. LogBabylon consolidates diverse log sources and enhances the extracted information's accuracy and relevancy. This facilitates a deeper understanding of log data, supporting more effective decision-making and operational efficiency. Furthermore, LogBabylon streamlines the log analysis process, significantly reducing the time and effort required to interpret complex datasets. Its capabilities extend to generating context-aware insights, offering an invaluable tool for continuous monitoring, performance optimization, and security assurance in dynamic computing environments.

CCS Concepts

• **Applied computing**; • **Security and privacy → Intrusion detection systems**;

Keywords

Logs, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Anomaly detection, Centralized log consolidation

ACM Reference Format:
Yang Lu, Rabimba Karanjai, Dana Alsagheer, Keshav Kasichainula, Lei Xu, Weidong Shi, and Shou-Hsuan Stephen Huang. 2025. LogBabylon: A Unified Framework for Cross-Log File Integration and Analysis. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3672608.3707883

*Both authors contributed equally to this research, and are equal first authors.

SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707883

1 Introduction

In today's large-scale production systems, log collection plays a crucial role as a source of valuable information. Logs provide key insights into a system's operational status, enabling problem identification, resolution, and performance optimization. However, logs vary in format and content depending on the system or application that generates them. Typically, they consist of unstructured text produced by logging statements in the source code, such as `logging.info()`, `printf()` [34] and other language-specific functions. Due to the diverse design goals of these systems, the information recorded in different log files is often difficult to cross-reference or consolidate, posing significant challenges for system administrators and engineers who need a unified view of system performance.

The growing volume of log data and the rapid expansion of log information make manual processing and analysis increasingly difficult. While logs offer valuable insights into the behavior and performance of complex computer systems, manual analysis is time-consuming, labor-intensive, and prone to errors. The need for quick troubleshooting, combined with the repetitive nature of log analysis, can lead to operator fatigue, burnout, and inconsistent interpretations, contributing to inefficiencies and inaccuracies in system diagnostics. Furthermore, logs often fail to provide real-time insights, making it harder to identify emerging issues or anomalies promptly.

We propose a unified framework, LogBabylon, to overcome these challenges. It adopts the semantic approach for log integration and leverages Large Language Models with the Retrieval Augmented Generation (RAG) [21]. RAG enhances LLMs by enabling them to

retrieve relevant, external information from databases or document repositories and generate contextually accurate responses. Combining internal and external resources, LLMs can significantly reduce the "hallucination" and enhance the response accuracy.

LLMs have demonstrated strong capabilities in different tasks like code translation [17],unit test generation [13], in understanding unstructured data [14, 15], including logs, and can process vast amounts of information more efficiently than traditional semantic methods. However, LLMs alone are insufficient for real-time analysis, as they rely on pre-trained knowledge that may not account for the dynamic and evolving nature of security events. By integrating RAG, we create a system capable of retrieving relevant log data in real time and generating accurate, context-aware insights. These systems can be even extended with the help of confidential execution of these LLMs [16].

LogBabylon leverages RAG and LLMs to overcome traditional semantic systems' scalability and contextual limitations. This enables more sophisticated and adaptable analysis of system activities and security events. LogBabylon significantly enhances situational awareness, improves the detection of complex attacks, and reduces the manual workload for system administrators and security analysts. As a result, it provides a more efficient and effective solution for modern security management, enhancing the capabilities of system administrators, engineers, and security analysts involved in log analysis and system diagnostics.

This research paper presents several key contributions to the field of log analysis, prompting a re-evaluation of current methodologies:

- **Unified Framework:** How might LogBabylon's comprehensive approach to log analysis reshape our understanding of the classification, consolidation, and interpretation of diverse log formats within a single, cohesive system?
- **LLM-Powered Parsing:** In what ways does LogBabylon's utilization of Large Language Models (LLMs) for the accurate extraction of log templates minimize our reliance on manual intervention and domain-specific expertise?
- **RAG-Enhanced Analysis:** How does the integration of Retrieval Augmented Generation (RAG) technology allow LogBabylon to leverage a vast knowledge base of log examples, ultimately facilitating deeper insights and more accurate anomaly detection?
- **Variable-Aware Prompting and In-Context Learning:** Can incorporating advanced prompting techniques and in-context learning significantly enhance the LLM's understanding of log data and improve template extraction accuracy?
- **Human-Readable Insights:** How does LogBabylon's ability to generate clear, concise, and human-readable analyses empower users to understand their log data and make informed decisions?

The structure of the paper will be as follows: Section 1 will be the introduction, setting the context for the study. Section 2 will cover related work, providing an overview of previous research in the field. Section 3 describes the methods and implementation used in the study, followed by Section 4, which focuses on the experiments conducted and result. Finally, Section 5 will conclude and summarize the key findings and the implications.

2 Related Work

Different approaches have been proposed to integrate log files. We examine them from three directions.

2.1 Semantic Approach of Log Integration

Traditional semantic approaches to log integration face significant challenges, primarily because they must handle log data from many different sources, each with its own format and structure [1, 5]. These methods often are not scalable enough to keep up with the massive amounts of log data generated by modern systems. Additionally, log events are highly specific to the context in which they occur, making it difficult to automate their interpretation. As a result, these limitations prevent semantic analysis from capturing key insights, especially when dealing with modern cyber threats and complex systems.

SEPSES is a semantic log analysis framework that offers a platform for semantic-based security monitoring, auditing, and forensic investigations [4]. It uses JSON-LD (JSON for Linking Data) to consolidate fragmented log information and extract and interlink related to security information. In this system, the log vocabulary stack is one of the key components that enables transforming the raw log data into a uniform RDF (Resource Description Framework) representation [23]. The log vocabulary stack consists of two kinds of terms. One is the core vocabulary, `slog:core`, the foundation of the basic terms to describe log messages independent from their sources. The other one is the source-specific terms. To provide event specifications, they collect background knowledge, such as syslog events, Apache events, etc.

The significant drawback of SPESES is that the log vocabulary and background knowledge are manually collected and added to the system. This limits the system's ability to interpret logs and link events dynamically. Thus, the scalability is a big concern for this system. As the volume of log event data surges, the semantic processing system must scale accordingly, which can be demanding for a system with a static design. Cyber threats constantly evolve, and semantic processing systems need continual updates to understand new patterns and threats. Keeping the system up-to-date requires ongoing effort and adaptation, which can be resource-intensive.

2.2 AI Approach of Log Integration

AI and machine learning have become increasingly central to integrating log data across different systems, offering enormous improvements in monitoring, security, and operational management efficiency and effectiveness. AI can automatically parse logs from different sources and normalize the data into a consistent format [10]. Machine learning models can learn from historical log data to understand normal system behavior. These model can detect anomalies or deviations in new log data, which could indicate potential issues like security breaches or system failures [2]. AI can analyze trends within log data to predict future system behaviors and potential problems. This proactive approach enables preemptive maintenance and security measures, minimizing downtime and preventing breaches before they occur [9].

There are numerous commercial and opensource tools for automatically analyzing log files, such as Semtext Logs [8], SolarWinds Loggly [29], Splunk [25], Rapid7 InsightOps [24], and Sumo

Logic [26]. These tools offer centralized log management, allowing users to collect, store, and analyze logs from various sources, with most providing cloud-based solutions. There are approaches which look at LLM based solutions as well [7]. They include search functionality and analytics features to help users gain insights from the data, as well as alerting systems to notify users of important events or anomalies. Additionally, they offer data visualization through charts, dashboards, or other graphical representations.

However, these tools often have drawbacks, such as being overly complex, offering limited log management capabilities, and performing poorly when handling large volumes of data or long-term retention. These difficulties highlight the trade-offs in log management tools, which are influenced by specific needs, technical expertise, and budget constraints.

2.3 Log Analysis

The traditional approach to log analysis starts with log parsing, an essential step for preparing log data for further analysis. A notable method in this area is "Drain", an online log parsing technique designed to speed up this process [10]. After parsing, the next step is log-based anomaly detection, which includes extracting relevant features and then using these features in a machine-learning model to identify and predict anomalous events [2]. These steps represent a foundational framework in the current landscape of log analysis technology.

However, traditional methods primarily focus on detecting anomalies. "LogGPT" [9] is a novel approach in which a model is trained to predict the next log entry based on previous sequences. If an observed log key does not appear in the top 50% of the model's prediction list, it is labeled as an anomaly. While this method utilizes large language models (LLMs) for anomaly detection, it does not address the system insight analysis.

By contrast, Xu et al. combined log parsing with text mining to analyze console logs and focused their research on detecting program bugs and runtime anomalies [9]. This approach, while innovative, remains limited to specific types of log analysis and does not encompass broader applications such as summarization or comprehensive system diagnostics. The log analysis landscape is rapidly evolving, with AI and ML playing a central role in transforming how organizations handle and derive insights from their log data. The focus is on automation, real-time analysis, and making log data more accessible and actionable across different organizational roles. This actually gets emphasized by [18] which shows the limitations on some of the approaches.

3 Detailed Design of LogBabylon

To overcome the complexities of log parsing, we introduce LogBabylon, an innovative approach powered by Large Language Models (LLMs). LogBabylon excels in accurately extracting log templates due to the LLMs' advanced comprehension abilities. It strategically utilizes LLMs to optimize performance and minimize resource consumption. Designed for versatility, LogBabylon adapts to various log formats and domains while minimizing human intervention in tasks, such as data labeling and parameter tuning. Furthermore, it incorporates a dynamic feedback loop to refine the parsing granularity based on human input, ensuring accurate and efficient log analysis.

The LogBabylon framework consists of a sequence of progressive steps: classification, consolidation, and interpretation. Figure 1 demonstrates high-level architecture design of LogBabylon.

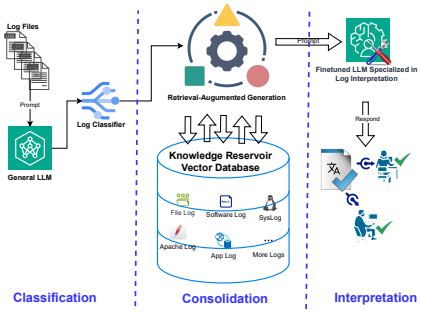

Figure 1: LogBabylon Architecture

3.1 Classification

3.1.1 Overview. LogBabylon employs a streamlined pre-processing approach that minimizes the need for extensive domain expertise. Unlike methods that rely heavily on regular expressions to replace specific variables, such as IP addresses [11], LogBabylon retains the original log message, allowing the LLM to fully grasp its context. This approach, which uses plain space-based tokenization and leverages the LLM's native tokenizer [6, 32], simplifies preprocessing while maintaining high log parsing efficiency.

3.1.2 Algorithm. LogBabylon's core algorithm centers on a prefix parse tree [11]. This tree structure efficiently matches incoming logs with existing clusters, thereby facilitating the rapid identification of similar entries. The algorithm intelligently determines when to invoke the LLM for template extraction, optimizing resource usage. As the LLM identifies new templates, the tree dynamically updates to incorporate these findings, ensuring an accurate and evolving representation of the log data. This sophisticated approach enables LogBabylon to effectively process and analyze logs by combining the strengths of the prefix parse tree and the LLM.

Three primary data structures form the backbone of our approach: a set of log clusters, a template pool, and a prefix parse tree. Figure 2 visually represents this organizational structure. The subsequent discussion delineates their respective functionalities.

Log Cluster: A log cluster is a collection of logs with the same template. It tracks the individual log IDs and stores a log embedding created by an LLM encoder for future use. Each cluster is characterized by its log template, extracted via LLM, and possibly multiple syntax templates aiding the prefix tree in its traversal and template matching processes. While syntax templates correspond directly to the tokens of the raw logs, identifying static and variable parts, the log templates from the LLM may represent several tokens with a single placeholder. These syntax templates are stored in a dictionary, utilizing the token counts as keys and the corresponding template lists as values.

Template Pool: The template pool establishes a linkage, mapping log templates to their respective log clusters.

Prefix Parse Tree: In this tree structure, each node, except the root, represents a token. The wildcard token "<*>" is a universal matcher for any token. Importantly, the leaf nodes and nearly all nodes (except the root) can have pointers to log clusters that match the token sequence, starting from the root. A notable feature is that a single log cluster may be accessible from multiple nodes due to the possibility of having different syntax template variations for the same log cluster.

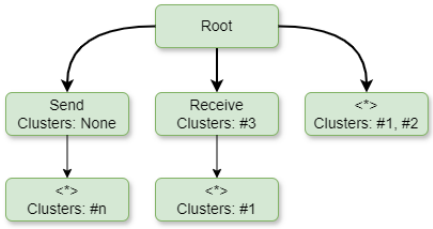

Figure 2: An example of the data structures in our method

When a new log arrives, LogBabylon first breaks it into individual tokens. These tokens are then sequentially matched against the nodes in the prefix parse tree, gradually narrowing down the potential log clusters. The process continues until all tokens are matched or no more matches are found.

Unlike traditional methods that rely on similarity metrics and thresholds, LogBabylon employs a more definitive matching process, categorizing matches as strict, loose, or non-existent. First, it checks whether the number of tokens in the incoming log aligns with the syntax templates of the cluster. If the counts differ, a match is ruled out immediately. Next, a "loose match" is attempted, where tokens from the syntax template and the log are aligned, allowing wildcard tokens ("<*>") to match any log token. If a loose match is found, regular expressions are used to ensure strict alignment with the non-wildcard elements of the syntax template. A complete token alignment signifies a strict match.

The log is readily added to the corresponding cluster if a strict match is identified. However, if no strict match is found, LogBabylon calls upon its LLM-powered template extractor to generate a new template. The data structures are then updated accordingly, ensuring that the system continuously learns and adapts to the new log patterns. This dynamic approach allows LogBabylon to categorize logs accurately even when encountering previously unseen formats.

LogBabylon's accuracy in log parsing stems from the inherent capabilities of LLMs, eliminating the tedious task of fine-tuning the hyperparameters for different log sources. LogBabylon simplifies the tuning process and minimizes the need for frequent LLM calls by leveraging the ability of LLMs to generate semantically accurate templates. Theoretically, if the LLM consistently produces templates that perfectly reflect the actual structure of the logs, the number of LLM calls would be limited to the total number of distinct syntax templates. This roughly equals the total number of log templates, making LogBabylon a highly scalable solution.

Algorithm 1 LogBabylon

Input: *logs*, *root_node* **Result:** *log_clusters*, *tree*

log_clusters ← {}
template_pool ← {}
partial_match_cache ← {} // *Cache for loose matches*
for *log* **in** *logs* **do**
  matched_clusters ← search(*tree*, *log*)
  if strict_match **then**
    strict_matched_cluster.add(*log*)
    added ← True
  else if loose_match **or** no_match **then**
    template ← get_llm_template(*log*, *log_clusters*)
    if *template* **in** *template_pool* **then**
      update_tree(*tree*, *log*, *template_pool*[*template*])
      added ← True
    else
      loose_matched_clusters ← *partial_match_cache*.get(*log*, [])
      if loose_matched_clusters = {} **then**
        loose_matched_clusters ← find_loose_matches(*log*, *log_clusters*)
        partial_match_cache[*log*] ← *loose_matched_clusters*
      end if
      for *cluster* **in** *loose_matched_clusters* **do**
        check_merge(*log*, *cluster*)
        if merge **then**
          cluster.update(*merged_template*)
          cluster.add(*log*)
          template_pool[*merged_template*] = *cluster*
          break
          added ← True
        end if
      end for
    end if
  end if
  if not *added* **then**
    new_cluster ← create_cluster(*log*, *template*)
    update_tree(*tree*, *log*, *new_cluster*)
    template_pool[*template*] = *new_cluster*
  end if
end for

The LogBabylon algorithm shown in Algorithm 1 efficiently parses logs by combining template matching with the power of LLMs. It starts with a hierarchical tree structure and two key data structures: log_clusters to store groups of similar logs, and template_pool to store templates for quick retrieval.

For each new log, LogBabylon searches for the tree for matching clusters. The log is immediately added to the cluster if a perfect ("strict") match is found. Otherwise, the LLM generates a template for the log.

LogBabylon then checks if this template already exists in template_pool. If so, the log is added to the corresponding cluster, and the tree structure is updated. If the template is new, then the algorithm attempts to merge the log with the most similar cluster. If the

merge is successful, the cluster is updated with the new template and the log is added. The template_pool is also updated.

If no match or successful merge is found, a new cluster is created for the log and the tree is updated to include this new cluster. This process ensures the efficient grouping of logs based on both exact and approximate matches, leveraging the LLM's ability to extract meaningful templates and enhance clustering accuracy.

3.2 Consolidation

LogBabylon utilizes Retrieval Augmented Generation (RAG) [20] to enhance its log analysis capabilities. This approach ensures that it does not solely rely on internal knowledge but also leverages a vast collection of "normal" log entries to interpret new logs effectively.

Consider this collection of a comprehensive library of log examples that is efficiently stored in a vector database. When a new log entry arrives, LogBabylon attempts to match it with existing templates and queries this database to find similar past entries.

The LLM then conducts a detailed semantic analysis by comparing the new log entry with the retrieved examples. This process allows LogBabylon to:

- **Identify subtle patterns and anomalies:** By comparing the new log with known examples, LogBabylon can detect even minor deviations that may indicate unusual activity.
- **Understand the log context:** The retrieved examples offer context, helping LogBabylon grasp the significance of the log.
- **Generate more accurate interpretations:** The combination of internal knowledge with external examples enables LogBabylon to provide more accurate and insightful interpretations.

This RAG-based approach renders LogBabylon a versatile and user-friendly tool for log analysis. Unlike traditional methods that require complex, multi-stage processing pipelines, LogBabylon offers an end-to-end solution that adapts seamlessly to any log source with a minimal configuration. By combining the power of LLMs with a rich knowledge base of log examples, LogBabylon simplifies log analysis and unlocks deeper insights from the data.

LogBabylon's application of the Retrieval Augmented Generation (RAG) model can be divided into three steps:

3.2.1 Finding Relevant Information. When LogBabylon receives a new log entry, denoted by x, it first seeks contextually relevant log entries from its knowledge base (stored in a vector database) to better understand the new entry. This step leverages dense-vector retrieval.

- **Turning Logs into Vectors:** LogBabylon employs a pre-trained embedding model to convert each log entry into a dense vector representation [19]. These vectors encapsulate the semantic meaning of the logs, enabling comparisons based on meaning rather than on literal words.
- **Calculating Similarity:** To identify relevant logs, LogBabylon compares the vector of the new log entry, x, with the stored vectors in the database. This comparison uses the inner product, a mathematical operation that measures similarity between two vectors.
- **Retrieving the Best Matches:** The database returns the log entries whose vectors have the highest similarity scores to x. This set of relevant log entries is denoted as z.

3.2.2 Generating a Response. With the relevant context z retrieved for the new log entry x, LogBabylon proceeds to generate a meaningful response, leveraging the Large Language Model (LLM), represented as f.

- **Combining Input and Context:** The LLM takes both the new log entry x and the retrieved relevant logs z as input.
- **Generating Output:** Based on this combined input, the LLM generates a response, y. This response can be an interpretation of the log, an anomaly detection flag, or another task-specific output.

3.2.3 LLM Semantic Analysis in LogBabylon. Once LogBabylon retrieves relevant log entries from its vector database, it must analyze and compare them with the new log entry. This is where the LLM's semantic analysis capabilities come into play. Figure 3 demonstrates this process.

Decoding the Vectors. The retrieved vectors are first transformed back into their original log entry format. This is achieved using the decoder component of the embedding model, which reverses the encoding process, converting the abstract vector representation into human-readable log text.

Framing the Anomaly Detection Task. LogBabylon frames the anomaly detection task as a question-answering problem. This approach leverages the natural ability of LLMs to interpret questions and provide human-like responses.

Crafting the Prompt. A specialized prompt template is used to provide structured input to the LLM. The template includes:

- **The new log entry:** The log that LogBabylon is analyzing.
- **The retrieved normal log entries:** These serve as context, offering a baseline for comparison.
- **The question:** The LLM is asked, *"Is the new log entry normal or abnormal, given the provided examples of normal logs?"*

LLM Analysis. The LLM processes this structured prompt by leveraging its language understanding and contextual reasoning abilities. It determines whether the new log entry deviates from the norm, producing two outputs. One is **a simple classification** and another is **a detailed explanation including the anomalies and it's significance.**

By framing anomaly detection as a question-answering task and harnessing the semantic capabilities of the LLM, LogBabylon can effectively identify unusual log entries, even those with subtle deviations from the expected behavior.

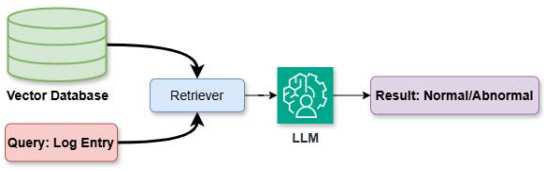

Figure 3: LLM Semantic Analysis in LogBabylon

3.3 Interpretation

LogBabylon's final step involves refining the insights gained from the previous stages and presenting them in a clear, human-readable format. This is where a specialized LLM, fine-tuned specifically for log interpretation, takes the center stage.

3.3.1 Enhancing LLM Template Extraction. While LogBabylon's base algorithm effectively matches log clusters, there is always room for improvement, particularly in the accuracy of the LLM template extractor. LogBabylon incorporates two key strategies to enhance this crucial component: variable aware prompting and in-context learning.

3.3.2 Variable-Aware Prompting. Inspired by prior research [22], LogBabylon employs variable-aware prompting to identify and classify variables in logs, such as timestamps, IP addresses, and error codes. This approach, similar to "chain of thought" reasoning [30], guides the LLM toward a deeper understanding of log structure and component meanings.

3.3.3 In-Context Learning with K-Shot Demonstrations. In-context learning (ICL) is a powerful technique that allows LLMs to learn new tasks without extensive fine-tuning [3]. LogBabylon leverages ICL by providing the LLM with a few examples of log entries and their corresponding templates. These examples serve as "demonstrations" that guide the LLM towards generating accurate templates for new log entries.

3.3.4 Combining the Power of Both. LogBabylon seamlessly integrates variable-aware prompting and ICL to enhance template extraction. Each time the LLM is called upon to extract a template, it receives a prompt containing:

- **Instructions:** A clear description of the template extraction task.
- **Demonstrations:** A small set of examples ($k = 3$) of log entries and their corresponding templates. These examples are carefully selected from a pool of previously extracted templates based on their similarity to the new log entry.
- **Seed Examples:** Ten examples representing different types of log parameters are included as initial seeds to guide the LLM in variable classification.
- **Query:** The log entry for which the LLM generates a template.

By combining clear instructions, relevant demonstrations, and seed examples, LogBabylon guides LLM to generate accurate and informative templates. This approach not only improves the accuracy of template extraction but also enhances the LLM's overall understanding of log data.

3.3.5 Beyond Variable-Aware Prompting and ICL. While variable-aware prompting and ICL significantly enhance LogBabylon's template extraction capabilities, there are other potential avenues for further improvement, including:

- **Utilizing more powerful LLMs:** As the field of language models advances, LogBabylon can readily incorporate newer, more capable LLMs to further boost its performance.
- **Supervised fine-tuning:** Training the LLM on a labeled dataset of log entries and their corresponding templates can lead to even greater accuracy in template extraction.

3.3.6 Human-Readable Output for Various Purposes. The final output produced by LogBabylon is designed to be easily understood by humans. It can take various forms, depending on the specific needs of the user, including:

- **Concise summaries of log events:** This provides a quick overview of what happened in the system.
- **Detailed explanations of anomalies:** This helps identify the root cause of problems and potential security threats.
- **Actionable insights for troubleshooting:** This guides users towards resolving issues and improving system performance.

This versatile output can be used for various purposes such as troubleshooting system errors, monitoring performance, detecting security breaches, and gaining a deeper understanding of user behavior. By providing precise, human-readable analyses, LogBabylon empowers users to make informed decisions and take appropriate actions based on their log data.

3.4 Metrics for Evaluation

We use existing metrics [33] for our evaluation.

Grouping Accuracy (GA) or Clustering Accuracy (CA): This metric measures the ratio of log messages that are correctly grouped.

Parsing Accuracy (PA): This metric evaluates the ability of the technique to extract templates accurately, a crucial aspect for tasks like anomaly detection.

F1 score of Grouping Accuracy (FGA): This is a template-level metric that evaluates the fraction of correctly grouped templates. It uses the true number of templates (N_g), parsed templates (N_p), and correctly parsed templates (N_c) to calculate Precision ($P_{GA} = \frac{N_c}{N_p}$) and Recall ($R_{GA} = \frac{N_c}{N_g}$) of Grouping Accuracy. The F1 score is the harmonic mean of these two values.

F1 score over template accuracy(FTA): FTA is the harmonic mean of Recall of Template Accuracy (RTA) and Precision of Template Accuracy (PTA). Similar to FGA, FTA evaluates correct template identification at the template level. A template is correct if log messages with the same parsed template share the same ground-truth template and the parsed template matches the ground-truth template exactly.

Precision Template Accuracy (PTA) is a template-level metrics to evaluate the quality of parsing. PTA measures the ratio of correctly identified templates to the total number of identified templates.

Recall Template Accuracy (RTA) measures the ratio of correctly identified templates to the total number of ground-truth templates.

4 Experiments

To rigorously evaluate LogBabylon's performance, we utilize two datasets, loghub-2k [35] and logPub [12], employing standard metrics like parsing accuracy alongside a novel "granularity distance" metric to assess the precision of template extraction. Our analysis of loghub-2k focuses on illustrating the contribution of each design component, such as variable-aware prompting and in-context learning. Meanwhile, our evaluation with logPub aims to demonstrate the effectiveness and efficiency of our approach when applied to large-scale, real-world datasets.

Table 1: Comparison with existing methods in loghub

	# of labeled logs	Clustering Accuracy	Parsing Accuracy
Eval of chatgpt	0	72.1	54.3
	4	76.1	79.0
DivLog[31]	200	92.8	98.1
LogBabylon	0	88.9	79.1
	4	82.2	63.9
	200	97.2	94.0

Table 2: Comparison on the logPub dataset

	Drain						Uniparser						LogBabylon					
	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD	GA	PA	FGA	FTA	GGD	PGD
Proxifier	69.2	68.8	20.6	17.6	4	14	50.9	63.4	28.6	45.7	5	10	44.9	55.8	35.2	46.9	4	8
Linux	**68.6**	11.1	77.8	25.9	30	432	28.5	16.4	45.1	23.2	108	274	23.8	14.3	70.5	41.0	16	71
Apache	**100.0**	72.7	100.0	51.7	**0**	21	94.8	94.2	68.7	26.9	11	31	88.0	75.4	88.0	57.6	0	7
Zookeeper	99.4	84.3	90.4	61.4	2	30	98.8	**98.8**	66.1	51.0	14	31	86.9	72.1	75.9	63.7	2	17
Hadoop	92.1	54.1	78.5	38.4	18	210	69.1	88.9	62.8	47.6	38	119	82.5	59.5	76.8	48.4	12	95
HealthApp	86.2	31.2	1.0	0.4	11	138	46.1	81.7	74.5	46.2	16	60	87.8	51.2	84.1	72.0	4	4
OpenStack	75.2	2.9	0.7	0.2	6618	23**	**100.0**	51.6	96.9	28.9	1	7	88.0	43.6	88.0	69.7	0	10
HPC	79.3	72.1	30.9	15.2	10	178	77.7	94.1	66.0	35.1	10	58	76.0	82.9	66.9	63.9	5	158
Mac	76.1	35.7	22.9	6.9	102	1347	73.7	68.8	69.9	28.3	73	624	78.9	26.7	74.5	31.9	37	391
OpenSSH	70.7	58.6	87.2	48.7	3	33	27.5	28.9	0.9	0.5	15	26	68.6	60.7	84.6	77.7	1	8
Spark	88.8	39.4	86.1	41.2	18	239	85.4	79.5	2.0	1.2	62	186	85.9	70.6	75.0	40.7	14	130
Thunderbird	83.1	21.6	23.7	7.1	137	2043	57.9	65.4	68.2	29.0	194	976	64.2	50.2	70.4	49.3	92	583
BGL	91.9	40.7	62.4	19.3	48	434	91.8	94.9	62.4	21.9	43	209	82.5	71.3	69.4	44.0	30	136
HDFS	99.9	62.1	93.5	60.9	2	6	**100.0**	94.8	96.8	58.1	1	1	88.0	83.4	65.7	50.9	4	23

Table 3: LogBabylon with different LLM

	Avg. # of	Avg. Time(s)			Avg. Metrics					
	LLM Calls	Per Infer.	Base	Total	GA	PA	FGA	FTA	GGD	PGD
LogBabylon w/ GPT-3.5-turbo	621.5	0.57	564.7	893.3	88.6	69.7	87.3	62.4	29.4	220.8
LogBabylon with GPT-4	468.8	4.62	498.3	2461.6	98.2	74.5	93.4	69.4	19.3	146.4
LogBabylon w/ fine-tuned Gemma2-9b(32shot)	7282.3	2.79	683.5	2499.9	85.6	78.5	72.8	54.4	50.5	213.4

4.1 Testbed

4.1.1 Datasets. We assess LogBabylon's performance using two datasets: Loghub-2k [35] and LogPub [12]. Loghub-2k, a benchmark in log parsing research, comprises 2,000 annotated log messages from 16 systems across diverse computing environments, including distributed systems, supercomputers, operating systems, mobile platforms, servers, and standalone software.

4.1.2 Experimental Environment. Experiments were conducted on an Ubuntu 20.04.3 LTS server with 512GB RAM. Template extraction utilized ChatGPT (gpt-3.5-turbo-0301) and GPT-4 (gpt-4-0613) via the OpenAI API, while log embedding employed the text-embedding-ada-002 model [28].

The LLM models used in the experiment are configured in the following way:

- **Template Extraction:** gpt-3.5-turbo-0301, gpt-4-0613
- **Log Embedding:** text-embedding-ada-002
- **Fine-tuning:** Gemma2-9b [27]
- **Temperature:** 0.

4.1.3 In-context Learning. For in-context learning, 32 log-template pairs were uniformly sampled from the first 10% of each dataset based on token length, serving as candidate logs and fine-tuning examples.

$$\begin{aligned} &\text{Sample Size} = 32 \text{ log-template pairs} \\ &\text{Sampling Method} = \text{Uniform from first 10\% of dataset} \end{aligned} \tag{1}$$

This experimental setup ensures a consistent and reproducible environment for evaluating log parsing techniques using state-of-the-art language models and embedding approaches.

4.2 Results

4.2.1 Loghub-2k. Our experiments with the Loghub-2k dataset focused on two objectives: evaluating the contributions of LogBabylon's core components and refining the system. This involved optimizing LLM-driven template extraction prompts, fine-tuning cluster merging criteria, and improving the verification process.

4.2.2 Comparison with Existing Parsers based on LLM. Evaluating existing LLM-based log parsers, which typically process logs line-by-line, poses significant challenges when applied to the extensive LogPub dataset due to the substantial computational costs associated with numerous LLM API calls. To address this issue, we conducted a comparative analysis using the more manageable Loghub-2k dataset. However, it is essential to acknowledge the limitations of this smaller dataset; its restricted scope means that a carefully selected set of labeled samples could potentially cover a large proportion of the log templates, which may lead to overly optimistic performance metrics that do not generalize well to larger, more diverse log collections. Despite these constraints, our method, LogBabylon, achieves competitive or superior performance compared with existing approaches when utilizing an equivalent number of labeled logs, as illustrated in Table 1. This outcome highlights LogBabylon's effective use of Large Language Models for log parsing tasks within the confines of a smaller dataset. It suggests that our approach offers tangible benefits in accuracy and efficiency. Furthermore, the strong performance indicates promise for future applications to larger and more complex log datasets, providing valuable insights into the efficacy of our LLM-based log parsing strategy while laying the groundwork for more extensive assessments down the line.

4.2.3 LogPub. The results from our evaluation of the extensive logPub dataset in Table 2 demonstrate LogBabylon's strong performance. Even without any fine-tuning for specific log formats, LogBabylon significantly outperforms all other methods in key metrics like Grouping Accuracy (GA), Full Accuracy (FGA), and Parsing Accuracy (PA).

While LogBabylon's overall accuracy (PA) is slightly lower, this is mainly due to the complexities of perfectly matching the varying levels of detail in different log formats. LogBabylon shows consistent performance across all 14 datasets within logPub without needing any adjustments for each specific log type. This highlights its adaptability and generalizability.

When we incorporate in-context learning (ICL) to calibrate LogBabylon for specific log formats (LogBabylon-C), we see a further improvement, particularly in template parsing metrics like PA and FTA. This demonstrates how ICL helps bridge the gap between general log parsing and the nuances of individual log structures.

4.2.4 Different LLMs. LogBabylon is designed to work with a variety of language models, allowing for flexibility and adaptability. This evaluation specifically aimed to explore how different LLMs impact its performance and efficiency. Our findings are summarized in Table 3.

5 Conclusion

This paper introduces LogBabylon, a unified framework for comprehensive log analysis that leverages Large Language Models (LLMs)

and Retrieval Augmented Generation (RAG) to address challenges in integrating and interpreting diverse log formats. LogBabylon combines accurate log template extraction using LLMs, reducing manual effort and domain expertise requirements, with RAG technology to utilize extensive log knowledge bases for enhanced anomaly detection and insights. Advanced prompting techniques and in-context learning further improve analysis accuracy and understanding. Rigorous evaluation on the loghub-2k and logPub datasets demonstrates LogBabylon's superior adaptability, accuracy, and efficiency compared to existing methods, making it a valuable tool for troubleshooting, performance monitoring, and anomaly detection. Future work will focus on integrating more advanced LLMs, expanding knowledge bases, and enabling interactive user feedback for refined and user-friendly analysis.

Acknowledgments

This work was supported by the US National Security Agency (H98230-22-1-0323) and part of the pipeline creation was supported by cloud credits from Google Developer Expert program.

The authors would like to thank the anonymous reviewers for their helpful comments and feedback, which greatly strengthened the final version of the paper.

References

[1] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2005. A comprehensive semantic framework for data integration systems. *Journal of Applied Logic* 3, 2 (2005), 308–328.

[2] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R. Lyu. 2022. Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection. arXiv:2107.05908 [cs.SE] https://arxiv.org/abs/2107.05908

[3] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[4] Andreas Ekelhart, Elmar Kiesling, and Kabul Kurniawan. 2018. Taming the logs - Vocabularies for semantic security analysis. *Procedia Computer Science* 137 (2018), 109–119. https://doi.org/10.1016/j.procs.2018.09.011 Proceedings of the 14th International Conference on Semantic Systems 10th – 13th of September 2018 Vienna, Austria.

[5] Andreas Ekelhart, Elmar Kiesling, and Kabul Kurniawan. 2018. Taming the logs-Vocabularies for semantic security analysis. *Procedia Computer Science* 137 (2018), 109–119.

[6] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. 2022. Investigating and improving log parsing in practice. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1566–1577.

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[8] L. Giamattei, A. Guerriero, R. Pietrantuono, S. Russo, I. Malavolta, T. Islam, M. Dînga, A. Koziolek, S. Singh, M. Armbruster, J.M. Gutierrez-Martinez, S. Caro-Alvaro, D. Rodriguez, S. Weber, J. Henss, E. Fernandez Vogelin, and F. Simon Panojo. 2024. Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software* 208 (2024), 111906. https://doi.org/10.1016/j.jss.2023.111906

[9] X. Han, S. Yuan, and M. Trabelsi. 2023. LogGPT: Log Anomaly Detection via GPT. In *2023 IEEE International Conference on Big Data (BigData)*. IEEE Computer Society, Los Alamitos, CA, USA, 1117–1122. https://doi.org/10.1109/BigData59044.2023.10386543

[10] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. 33–40. https://doi.org/10.1109/ICWS.2017.13

[11] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[12] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. 2024. 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*. 223–234.

[13] Rabimba Karanjai, Aftab Hussain, Md Rafiqul Islam Rabin, Lei Xu, Weidong Shi, and Mohammad Amin Alipour. 2024. Harnessing the Power of LLMs: Automating Unit Test Generation for High-Performance Computing. *arXiv preprint arXiv:2407.05202* (2024).

[14] Rabimba Karanjai, Edward Li, Lei Xu, and Weidong Shi. 2023. Who is smarter? an empirical study of ai-based smart contract creation. In *2023 5th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 1–8.

[15] Rabimba Karanjai and Weidong Shi. 2024. LookALike: Human Mimicry based collaborative decision making. *arXiv preprint arXiv:2403.10824* (2024).

[16] Rabimba Karanjai and Weidong Shi. 2024. Trusted LLM Inference on the Edge with Smart Contracts. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–7.

[17] Rabimba Karanjai, Lei Xu, and Weidong Shi. 2024. SolMover: Smart Contract Code Translation Based on Concepts. In *Proceedings of the 1st ACM International Conference on AI-Powered Software* (Porto de Galinhas, Brazil) *(AIware 2024)*. Association for Computing Machinery, New York, NY, USA, 112–121. https://doi.org/10.1145/3664646.3664771

[18] Egil Karlsen, Xiao Luo, Nur Zincir-Heywood, and Malcolm Heywood. 2024. Benchmarking Large Language Models for Log Analysis, Security, and Interpretation. *Journal of Network and Systems Management* 32, 3 (2024), 59.

[19] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300* (2019).

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[22] Zhenhao Li, Chuan Luo, Tse-Hsun Chen, Weiyi Shang, Shilin He, Qingwei Lin, and Dongmei Zhang. 2023. Did we miss something important? studying and exploring variable-aware log abstraction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 830–842.

[23] Jeff Z Pan. 2009. Resource description framework. In *Handbook on ontologies*. Springer, 71–90.

[24] Rapid7 InsightOps [n. d.]. https://docs.rapid7.com/insightops/

[25] Splunk. 2021. Splunk for the MSSP Technical Architecture. https://www.splunk.com/en_us/pdfs/white-paper/splunk-for-managed-security-service-providers-technical-architecture.pdf

[26] Sumo Logic 2024. https://help.sumologic.com/

[27] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118* (2024).

[28] text-embedding-ada-002 [n. d.]. https://openai.com/index/new-and-improved-embedding-model/ New and improved embedding model.

[29] Dave Wagner and Pravesh Ramachandran. 2021. SolarWinds Loggly Playbook. https://www.loggly.com/wp-content/uploads/2021/01/2012_loggly_ebook_LogglyPlaybook.pdf

[30] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[31] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2023. Prompting for automatic log template extraction. *arXiv preprint arXiv:2307.09950* (2023).

[32] Siyu Yu, Pinjia He, Ningjiang Chen, and Yifan Wu. 2023. Brain: Log parsing with bidirectional parallel tree. *IEEE Transactions on Services Computing* 16, 5 (2023), 3224–3237.

[33] Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. 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*. 4559–4570.

[34] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. 2023. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. arXiv:2008.06448 [cs.SE] https://arxiv.org/abs/2008.06448

[35] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. 2023. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 355–366.