

HG-InsightLog: Context Prioritization and Reduction for Question Answering with Non-Natural Language Construct Log Data

Supriya Bajpai* Athira Gopal Chandrakant Harjpal Niraj Kumar

Fujitsu Research India, Bengaluru, India

supriya.bajpai@fujitsu.com, athira.gopal@fujitsu.com,
chandrakant.harjpal@fujitsu.com, niraj.s.kumar@fujitsu.com

Abstract

Modern IT systems generate vast amounts of log data, which pose challenges for Large Language Models (LLMs) due to their large size, irrelevant entries, and non-Natural Language (non-NL) construct (e.g., domain-specific jargon, error codes, file paths, and abbreviations). Traditional methods like Retrieval-Augmented Generation (RAG) and GraphRAG fail to preserve temporal sequences, handle non-NL for context and entities extraction, and dynamically prioritize query-relevant context. To address these limitations, we propose HG-InsightLog, a novel framework that constructs a multi-entity temporal hypergraph representing log attribute-value pair as nodes and connecting them with hyperedges, capturing critical connections in the data. HG-InsightLog introduces a multi-step query personalization mechanism enhancing the Personalized PageRank algorithm to rank hyperedges based on query relevance and contextual centrality to prioritize critical connections. Top ranked hyperedges are extracted and converted back into log formats preserving temporal order and reducing context. Experimental results across multiple datasets demonstrate its superiority over existing methods, enhancing factual, causal, and analytical reasoning. Our approach enables smaller LLMs like LLaMA-8B to perform effective log-based QA. Being model-agnostic and training-free, it scales with evolving open-source LLMs without relying on proprietary systems.

1 Introduction

Log files are crucial for monitoring, diagnostics, and root cause analysis (RCA) in IT systems, capturing events, operations, and errors (He et al., 2022). However, their sheer volume—sometimes reaching up to 50 GB per hour—makes manual analysis overwhelming and renders traditional machine learning methods ineffective for efficient question answering (QA) (Chen et al., 2024;

Ahmed et al., 2023).

LLMs like GPT-3.5, GPT-4 (Achiam et al., 2023), and Llama (Touvron et al., 2023) excel in QA tasks with Natural Language (NL) data (Lewis et al., 2020; Erkan and Radev, 2004; Mihalcea and Tarau, 2004) but face *two major challenges (long temporal context (see Table 7) and non-NL construct data)* when applied to complex log analysis QA. For example, consider the query: *Given journal logs having a software error (e.g., SWERR), explain it's root cause.*

To find the root cause, LLMs need to process all preceding logs sequentially, requiring a long-context capability while preserving the sequence of events over time (Ahmed et al., 2023) (A brief excerpt of the logs is shown in Figure 4, and a case study illustrating the problem in more detail is provided in Table 8, both in the Appendix.) Additionally, they must recognize the relationships between various logs or errors, which necessitates structuring the data as a graph to correctly capture the origin of the error. However, LLMs encounter significant challenges when structuring log data as a graph. While they excel at extracting entities and relationships from NL data due to their training on vast unstructured textual datasets with clear linguistic structures, grammar, and context, they struggle with log data, which is semi-structured having non-NL construct. Logs contain numeric values, timestamps, error, device, and software codes, and domain-specific jargon, making relationships implicit rather than explicit (see Figure 4). Additionally, LLMs are not trained on system logs or execution traces, their ability to generalize in this domain is limited (Chen et al., 2023). Several LLM-based approaches exist, but they face significant limitations for QA with long context non-NL construct data. Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) and Long-context RAG (LC-RAG) (Jin et al., 2024) loose context as well as temporal sequences due to chunking,

*Corresponding author: supriya.bajpai@fujitsu.com

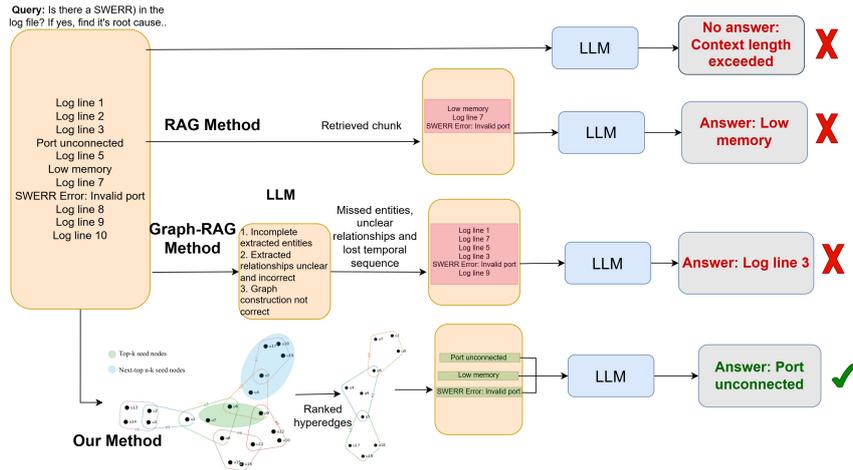


Figure 1: **HG-InsightLog QA Case Study:** This figure presents a complex log analysis-type question from Journal log dataset, used to compare our approach against the Baseline LLM, RAG and Graph-RAG methods. It demonstrates the accurate prediction of our method in identifying "Port unconnected" as the root cause of SWERR.

which is essential for task such as RCA. They also perform poorly on global queries where vital information is spread across large number of multiple chunks. GraphRAG (Edge et al., 2024) and HOLMES (Panda et al., 2024) are designed for global and multihop answering respectively, but are resource-intensive, requiring all data to be passed to the LLM for entity and relationship extraction, increasing token usage and processing time. It also demands extensive prompt engineering to convert data into a graph. Additionally, GraphRAG struggles with non-NL data, as NL-trained LLMs fail to extract accurate entities and relationships, leading to incomplete or incorrect knowledge graphs. It also lacks the ability to preserve temporal sequences, which is critical for tasks like RCA. Other approaches, such as LexRank (Erkan and Radev, 2004) and Personalized PageRank (PPR) (Brin, 1998) identifies query focused central sentences but fails with complex relationships and temporal sequences in logs. Both depend on graph construction, which is challenging due to unclear event interactions (entities-relationships) in log data.

To overcome these challenges, we introduce HG-InsightLog, a novel framework for QA with semi-structured non-NL construct log data that transforms log data into a multi-entity hypergraph representation, encoding all the log attribute-value pairs as nodes and connecting them with hyperedges to capture the multi-dimensional complex relationships. Unlike traditional GraphRAG, HG-InsightLog addresses key challenges effectively. Additionally, the framework prioritizes

query-specific context through an enhanced multi-step Personalized PageRank algorithm which enhances flexibility and adaptability of the algorithm. The algorithm dynamically ranks hyperedges based on multi-step query relevance and structural centrality to focus on critical information. HG-InsightLog also preserves temporal sequence of critical hyperedges ensuring causality is retained, which is critical for RCA. Additionally, we demonstrate these improvements using the smaller Llama3.1 8B model, showcasing its efficiency in complex reasoning and QA tasks without larger LLMs. A case study on the Journal log dataset is provided in Figure 1 (see Table 8 and Figure 4 in Appendix for details). To the best of our knowledge, our work is the first to construct a hyper-KG for non-natural constructs and semi-structured log data and utilize it for global and reasoning-based QA tasks.

The main contributions of this paper are as follows:

- **Scalable QA framework for non-NL construct data:** A novel QA framework for log data with non-NL constructs, employing a novel approach for constructing multi-entity temporal hypergraphs. It handles large size log files efficiently, making it suitable for real-world log data analysis.
- **Temporal sequence preservation:** HG-InsightLog preserves the temporal order of the hyperedges, essential for RCA
- **Multi-step query-specific context prioritization:** A novel mechanism to dynamically rank and prioritize the most critical connections in

the data by enhancing Personalized PageRank algorithm with multi-step query personalization.

By leveraging the efficiency and capabilities of Llama3.1 8B, this work sets a benchmark for resource-efficient, accurate, and query-adaptive log data analysis.

2 Related Work

LLMs for log analysis - LLMs have been explored for log analysis, particularly for parsing and anomaly detection. LILAC (Jiang et al., 2024) applies LLMs for log parsing, converting unstructured logs into structured formats. LogBERT (Guo et al., 2021) employs self-supervised learning to predict masked keys in normal log sequences, aiding anomaly detection. RCA-focused methods Chen et al. (Chen et al., 2023) use LLMs to summarize cloud incidents and predict root causes, though they rely on incident-level data rather than raw logs.

Retrieval-Augmented Generation (RAG) and Limitations: To answer questions from log files, LLMs must retrieve information from log files, address global questions on large log files beyond their context length and maintain temporal sequence. RAG (Lewis et al., 2020) and Long-context RAG (Jin et al., 2024) allows LLMs to fetch and use external information by segmenting it into chunks, presenting only relevant ones based on the query. However, this disrupts the global context and temporal structure crucial in log files.

Graph-Based Methods and Temporal Constraints: Graph-RAG (Edge et al., 2024), HOLMES (Panda et al., 2024), HippoRAG (Gutierrez et al., 2024) and Arigraph (Anokhin et al., 2024), LightRAG (Guo et al., 2024) are graph-based methods that aim to solve the challenges of RAG. Graph-RAG addresses global questions by combining community-level summaries, leveraging the LLM’s ability to extract an entity knowledge graph. However, iterating over community summaries for global question answering further amplifies the resource demands. HOLMES, on the other hand, is designed for multi-hop question answering, utilizing a context-aware knowledge graph distilled to contain only query-relevant information. HippoRAG leverages a graph-based memory structure where stored knowledge is organized as a dynamically growing graph, and retrieval is enhanced using Personalized PageRank to prioritize relevant context for LLMs. In all the above

graph-based approaches constructing examples for entities and relationship extraction is challenging for log data and requiring extensive prompt engineering and often resulting in the omission of information not explicitly covered in the prompts. Furthermore, all these approaches except LightRAG are resource-intensive, as it necessitates passing all data to the LLM for entity and relationship extraction, leading to increased token usage and processing time. LightRAG, GraphRAG, HippoRAG and HOLMES does not preserve the temporal context, which is crucial for causality analysis.

Extractive Summarization and Ranking Algorithms: Graph-based ranking models like PageRank (Brin and Page, 1998), LexRank (Erkan and Radev, 2004), and TextRank (Erkan and Radev, 2004) are effective for summarization but lack query-specific adaptability. They rely on global textual similarities rather than dynamically prioritizing context based on the query, making them unsuitable for RCA tasks requiring causal and temporal consistency.

Gap in Existing Work: Despite these advances, no existing method fully addresses the challenge of complex log-based QA while preserving global context, prioritizing query-relevant information, and maintaining temporal integrity. Our proposed approach bridges this gap by leveraging a hypergraph-based structure to dynamically rank and retrieve log segments, enabling efficient, training-free, zero-shot RCA.

3 Methodology

The key idea of our method is to create a hypergraph representation of the entire large log file and rank the hyperedges based on the user query and node centrality. We then extract the top-p hyperedges while preserving the temporal sequence and convert them back into the original log structure. This approach reduces the context size while preserving the temporal sequence of the logs and prioritizing critical information. Figure 2 provides an overview of our approach. We begin with a brief overview of the problem in Section 3.1, followed by a detailed explanation of our method in Sections 3.2 - 3.6.

3.1 Problem Statement

In this work, we address the challenge of question answering (QA) with large log files in a zero-shot setting. This approach assumes no prior domain-

specific labeled data, making the problem both complex and practically valuable. Furthermore, we operate in a training-free environment, relying on the reasoning abilities of LLMs. The goal is to extract an answer A_q to a NL query q from a given log file L where A_q requires deducing the answer from a vast, non-NL log file.

3.2 Proposed Method (HG-InsightLog)

The overview of HG-InsightLog is provided in Figure 2. First, we construct a multi-entity relationship hypergraph from the log data, representing each log attribute as a distinct node and connecting all nodes of a log entry via a hyperedge. Nodes can be shared across multiple hyperedges. Next, the hyperedges are ranked for query and context based prioritization using a multi-step personalization mechanism that dynamically integrates importance vectors from two subsets of seed nodes, weighted by a tunable parameter for flexibility. Finally, we reduce context size by selecting the top-k hyperedges and converting them back into the original log format while preserving temporal order. This serves as context for LLM-based answer generation. The methodology consists of: (i) Hypergraph Construction, (ii) Seed Node Extraction, (iii) Hyperedge Ranking, and (iv) Ranked Context & Prompt Creation for LLM.

3.3 Hypergraph Construction

LLMs often struggle to extract meaningful entities and relationships from complex, non-natural language (non-NL) log data because they are primarily built for processing natural language — they lack grammar, structure, clear meanings, named entities, and obvious relationships. Instead, they often include vague or unclear connections between elements.

To overcome this limitation, we propose a method that constructs a **multi-dimensional hypergraph** specifically designed for log data. Our approach ensures that no entity or relationship is missed by considering every individual piece of information as a node/entity in log entries—even if they span multiple lines. All the nodes/entities of a log entry is connected by a hyperedge.

We illustrate this with two example log entries from the journal dataset (provided in Figure 4):

```
Oct 30 00:35:36 interface:***systemd[1]
Starting Journal Socket
Oct 30 00:35:36 interface:***systemd[329]
```

```
systemd-journald.socket: Failed to
connect . . .
```

From these log entries, we extract structured pieces of information such as *date*, *time*, *component/interface*, and *message*. Each piece is treated as an **attribute-value pair**, which becomes a node in the hypergraph.

For example, from the first log line we extract:

- Date: Oct 30
- Time: 00:35:36
- Interface: *****systemd[1]
- Message: Starting Journal Socket

Each of these is a node/entity. The combination of nodes extracted from a single log entry becomes a hyperedge. So, the first log line forms the hyperedge:

```
(Date: Oct 30, Time: 00:35:36,
Interface: *****systemd[1], Message:
Starting Journal Socket)
```

The second log line forms another hyperedge:

```
(Date: Oct 30, Time: 00:35:36,
Interface: *****systemd[329], Message:
systemd-journald.socket: Failed to
connect . . .)
```

Here, the dimensions (also called *attributes*) include **Date**, **Time**, **Interface/Component**, and **Message**. Each node is attribute-value pair, and each log entry connecting the nodes, forms a hyperedge. For more details see Appendix A.4 and Figure 5.

Let $V = \{v_1, v_2, \dots, v_N\}$ be the set of all unique nodes, where each node represents a distinct attribute-value pair extracted from the whole log file. These nodes encapsulate structured information such as timestamps, components, and message content. The total number of unique nodes is denoted by $N = |V|$. For a log file containing T entries, each log entry ℓ_i , where $i \in \{1, 2, \dots, T\}$, is transformed into a hyperedge $e_i \subseteq V$ that connects all attribute-value pairs present in that entry. The complete set of hyperedges is represented as $E = \{e_1, e_2, \dots, e_T\}$, where each hyperedge corresponds to one log entry. The resulting hypergraph is denoted by $H = (V, E)$, where V is the set of nodes and E is the set of hyperedges. Particular node may appear in multiple hyperedges, reflecting shared nodes across different log entries. The size of each hyperedge $|e_i|$ corresponds to the number of attributes present in the associated log entry.

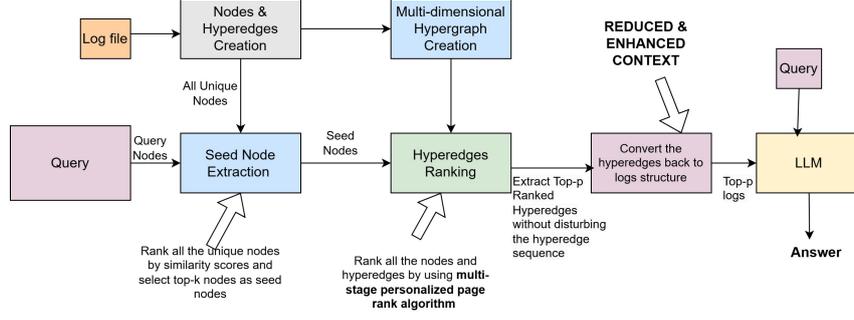


Figure 2: **Model Overview:** The figure provides an overview of the HG-InsightLog methodology. (i) Each attribute of a log is represented as a dimension and each attribute (e.g. 'Date') and its corresponding value (e.g. 'Oct 30') is made a node and all the nodes in a log is connected by a hyperedge. Example node: 'Date: Oct 30' and hyperedge: ('Date: Oct 30', 'Time: 00:35:36', 'interface:*****systemd[1]', 'message: Starting Journal Socket') (ii) The above process is repeated for all the logs in the file to create a big multi-dimensional hypergraph (iii) The hypergraphs are ranked with multi-step PPR algorithm utilizing seed nodes and hypergraph structure. (iv) top-p hyperedges are selected and converted back to original log format. (v) These logs are send to LLM as improved and reduced context for answer generation.

This formulation ensures that the hypergraph comprehensively captures the structural and semantic relationships inherent in the log data.

3.4 Extracting seed nodes from the query

Seed nodes, represented by V_q , where $V_q \subset V$, are extracted using the set of unique nodes V and query nodes V_Q . Top-n nodes are extracted using semantic similarity. This is done by using a pre-trained model (sentence-transformers/all-mpnet-base-v2) to encode both the query and the set of unique nodes into dense vector embeddings. These embeddings represent the semantic meaning of the text. Once the query and unique nodes descriptions are encoded, cosine similarity is computed between the query vector and each node vector to measure how closely related they are in terms of meaning. The function then sorts the nodes based on their similarity scores in descending order. The top-n most similar nodes (V_q) are retrieved.

3.5 Multi-step Query and centrality dependent hyperedge ranking

We extend the existing Personalized PageRank (PPR) (Brin, 1998; Haveliwala, 2002) by introducing a multi-step personalization mechanism that dynamically integrates importance vectors from two subsets of seed nodes, weighted by a tunable parameter. Unlike standard PPR, which typically uses a single personalized vector, this method leverages similarity-based seed node selection to dynamically integrate top k and next top (n-k) similar nodes, en-

hancing flexibility and adaptability. This extended PPR is used to rank the nodes which is then used to rank hyperedges.

The multi-step personalized PageRank (PPR) for nodes within the hypergraph is computed as follows:

$$P_{\text{rank}}^{(t)} = \alpha \tilde{A} P_{\text{rank}}^{(t-1)} + (1 - \alpha) P_{\text{vector}}, \quad (1)$$

where, $P_{\text{rank}}^{(t)} \in \mathbb{R}^{N \times 1}$ represents the multi-step Personalized PageRank vector at iteration t . It denotes the probability distribution over the nodes after t steps of the random walk. The term $P_{\text{vector}} \in \mathbb{R}^{N \times 1}$ is the personalization vector, which remains fixed throughout the computation and biases the random walk towards specific nodes. The iterative update combines two components: a random walk weighted by $\alpha \in [0, 1]$ and a teleportation step weighted by $1 - \alpha$, ensuring that the algorithm converges to a steady-state probability distribution influenced by both the graph structure and the personalization vector. $\tilde{A}[i, j]$ is the transition probability matrix, which represents the probability of transitioning from node v_i to node v_j , accounting for their co-occurrence in the hypergraph.

The node-specific personalization P_{vector} is designed to reflect the personalization preferences of the nodes, and is refined to incorporate multi-steps of seed nodes.

$$P_{\text{vector}} = \beta P_{\text{top-k}} + (1 - \beta) P_{\text{next-top}}, \quad (2)$$

where, $\beta P_{\text{top-k}}$ and $(1 - \beta) P_{\text{next-top}}$ are the personalization vectors derived from the top-k seed

nodes, and the next $n - k$ seed nodes such that, $P_{\text{top-k}} \in \mathbb{R}^{N \times 1}$ and $P_{\text{next-top}} \in \mathbb{R}^{N \times 1}$. $\beta \in [0, 1]$ serves as a tunable parameter that modulates the relative influence of top-k seed nodes personalization and next-top n-k seed nodes personalization. The u^{th} node of vector $P_{\text{top-k}}$ can be expressed as:

$$P_{\text{top-k}}(u) = \begin{cases} 1/k, & \text{if } u \in V_{q,k} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where $V_{q,k} \subset V_q$ is the set of top-k seed nodes.

$$P_{\text{next-top}}(u) = \begin{cases} 1/(n - k), & \text{if } u \in V_{q,n-k} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where $V_{q,n-k} \subset V_q$ is the set of the next $n - k$ seed nodes. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ for the hypergraph H is constructed, where, each entry $A[i, j]$ represents the weighted relationship between node v_i and node v_j based on their co-occurrence in the hyperedges. Specifically, two nodes v_i and v_j are adjacent if they both belong to at least one hyperedge $e_k \in E$. The weight of $A[i, j]$ is defined as the sum of co-occurrences between nodes v_i and v_j across all hyperedges, normalized by the size of the hyperedge:

$$A[i, j] = \sum_{e_k \in E} \frac{\delta(v_i \in e_k) \cdot \delta(v_j \in e_k)}{|e_k|}$$

where $\delta(v_i \in e_k)$ is 1 if node v_i is in hyperedge e_k , otherwise it is 0, and $|e_k|$ is the size (cardinality) of hyperedge e_k . For the diagonal entries of the matrix, representing a node's self-contribution across all hyperedges, we define:

$$A[i, i] = \sum_{e_k \in E} \frac{\delta(v_i \in e_k)}{|e_k|}.$$

The adjacency matrix A is sparse, as nodes are only connected if they co-occur in at least one hyperedge. A is normalized to create transition probability matrix $\tilde{A}[i, j]$ defined as:

$$\tilde{A}[i, j] = \frac{A[i, j]}{\sum_{k=1}^n A[i, k]},$$

which represents the probability of transitioning from node v_i to node v_j , accounting for their co-occurrence in the hypergraph.

The P_{rank} is estimated by iterating until convergence is achieved, as determined by the condition:

$$\|P_{\text{rank}}^{(t)} - P_{\text{rank}}^{(t-1)}\| < \epsilon, \quad (5)$$

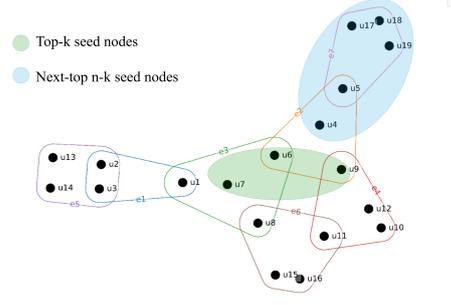


Figure 3: The figure illustrates a multi-entity hypergraph where nodes, labeled as "u", are connected by hyperedges, denoted as "e". Top-k and Next-top n-k seed nodes are shown in blue and green respectively.

where ϵ is a predefined threshold for convergence, and $\|\cdot\|$ denotes the norm.

Hyperedge Weight Calculation and Ranking:

The final step computes the hyperedge weights $w(e_i)$ based on the multi-step Personalized PageRank scores of the constituent nodes. Each hyperedge weight is the sum of the weights of its nodes $u \in e_i$, expressed as:

$$w(e_i) = \sum_{u \in e_i} P_{\text{rank}}(u), \quad (6)$$

where $P_{\text{rank}}(u)$ represents the multi-step PPR of node u , which captures the node's significance within the local structure of the hypergraph. Finally, the top-p hyperedges are extracted out of T hyperedges and used for context creation.

3.6 Creating Prompt with Ranked Context for LLM

To create the prompt for the LLM, the top-p hyperedges are extracted while preserving the original sequence. These hyperedges are then converted back into the original log format, as LLMs struggle with structured lists or tuples. This process ensures optimal information retrieval by retaining only the critical logs, removing irrelevant ones, and preserving the temporal sequence.

4 Experimental Setup

4.1 Evaluation Details

Datasets: We conduct QA experiments on two public log datasets: OpenSSH dataset (Zhu et al., 2023), Zookeeper dataset (Zhu et al., 2023) and one private Journal log dataset. For more details, refer to Appendix A.1.

QA Dataset Generation: Due to the lack of a

Table 1: Performance comparison between Our method (HG-InsightLog) with other State-of-the-art methods. P_{BERT} , R_{BERT} and F_{BERT} represent precision, Recall and F1-score respectively

Model	Question Category	RAG (Llama3.1-8B)			LC-RAG (Llama3.1-8B)			HOLMES (GPT-4o)			GraphRAG (GPT-4o)			Our (Llama3.1-8B)		
		P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}
Zookeeper	Fact-based	0.7168	0.8443	0.7733	0.6906	0.8230	0.7490	0.3978	0.4487	0.4217	0.4714	0.3801	0.4209	0.8873	0.9294	0.9072
	Analytical Reasoning	0.7961	0.8466	0.8185	0.8165	0.8453	0.8290	0.3501	0.3246	0.3369	0.3854	0.3167	0.3477	0.8764	0.8506	0.8628
	Causal Reasoning	0.7518	0.8187	0.7813	0.7712	0.8271	0.7982	0.3764	0.4035	0.3895	0.3369	0.3018	0.3183	0.8653	0.8415	0.8527
	Decision-based	0.6996	0.8461	0.7624	0.7076	0.8482	0.7715	0.4521	0.4604	0.4562	0.3645	0.3243	0.3432	0.8434	0.8817	0.8613
	Summarization	0.7147	0.8274	0.7656	0.7091	0.8532	0.7745	0.3008	0.3377	0.3182	0.3340	0.3091	0.3211	0.8252	0.8505	0.8370
	Temporal Reasoning	0.7188	0.8714	0.7852	0.7408	0.8586	0.7930	0.3219	0.3334	0.3275	0.3517	0.3324	0.3418	0.8255	0.8889	0.8555
	Anomaly Detection	0.7297	0.8619	0.7876	0.7422	0.8544	0.7923	0.4207	0.3195	0.3632	0.3922	0.3471	0.3683	0.8225	0.8809	0.8503
OpenSSH	Fact-based	0.6666	0.7835	0.7184	0.6532	0.8121	0.7240	0.3821	0.4253	0.4025	0.4499	0.3689	0.4054	0.8099	0.8547	0.8332
	Analytical Reasoning	0.8033	0.8105	0.8033	0.8146	0.8253	0.8199	0.3552	0.3355	0.3451	0.3738	0.3126	0.3405	0.8731	0.8572	0.8643
	Causal Reasoning	0.7291	0.8220	0.7702	0.7462	0.8331	0.7873	0.3658	0.3902	0.3776	0.3461	0.3172	0.3310	0.8683	0.8550	0.8729
	Decision-based	0.6879	0.7993	0.7372	0.6961	0.8224	0.7540	0.4673	0.4751	0.4712	0.3741	0.3462	0.3596	0.8579	0.8823	0.8693
	Summarization	0.7060	0.8084	0.7530	0.6899	0.8486	0.7611	0.2932	0.3254	0.3085	0.3212	0.2932	0.3066	0.8058	0.8637	0.8331
	Temporal Reasoning	0.7818	0.8216	0.7960	0.7748	0.8337	0.8032	0.3353	0.3423	0.3388	0.3643	0.3452	0.3545	0.8417	0.8625	0.8510
	Anomaly Detection	0.6911	0.7965	0.7376	0.7061	0.8249	0.7609	0.4118	0.3152	0.3571	0.3822	0.3354	0.3573	0.8504	0.8671	0.8577

Table 2: Performance comparison of our method using GPT-4o and Mistral-7B, with RAG (GPT-4o) as a baseline.

extbfModel	extbfQuestion Category	RAG (GPT-4o)			Our (GPT-4o)			Our (Mistral-7B)		
		P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}	P_{BERT}	R_{BERT}	F_{BERT}
Zookeeper	Fact-based	0.9009	0.9234	0.9113	0.9249	0.9498	0.9375	0.9016	0.9379	0.9190
	Analytical Reasoning	0.8733	0.8493	0.8608	0.8753	0.8462	0.8607	0.8709	0.8543	0.8620
	Causal Reasoning	0.8723	0.8465	0.8589	0.8812	0.8529	0.8676	0.8846	0.8526	0.8679
	Decision-based	0.8456	0.8943	0.8686	0.8624	0.9121	0.8871	0.8601	0.8901	0.8738
	Summarization	0.8735	0.8270	0.8493	0.8798	0.8367	0.8579	0.8714	0.8229	0.8460
	Temporal Reasoning	0.8285	0.8840	0.8549	0.8357	0.9210	0.8772	0.8435	0.8976	0.8694
	Anomaly Detection	0.8210	0.8825	0.8502	0.8432	0.9086	0.8753	0.8339	0.8914	0.8613
OpenSSH	Fact-based	0.8456	0.8487	0.8457	0.8447	0.8728	0.8578	0.8304	0.8607	0.8446
	Analytical Reasoning	0.8733	0.8429	0.8575	0.8818	0.8574	0.8690	0.8478	0.8592	0.8530
	Causal Reasoning	0.8591	0.8401	0.8492	0.8626	0.8549	0.8584	0.8301	0.8573	0.8431
	Decision-based	0.8631	0.8869	0.8742	0.8752	0.8834	0.8784	0.8351	0.8734	0.8534
	Summarization	0.8810	0.8077	0.8422	0.9055	0.8298	0.8656	0.8774	0.8552	0.8657
	Temporal Reasoning	0.8297	0.8577	0.8427	0.8360	0.8634	0.8489	0.8160	0.8550	0.8347
	Anomaly Detection	0.8411	0.8629	0.8508	0.8772	0.8760	0.8759	0.8316	0.8649	0.8473

Table 3: Performance comparison of RAG, LC-RAG, and our method with Llama3.1-8B and GPT-4o as LLM on three datasets using complex log analysis questions and human evaluation(see Appendix A.3).

Methods	Journal	OpenSSH	Zookeeper
RAG (Llama)	0.7024	0.68	0.475
LC-RAG (Llama)	0.7362	0.7481	0.5123
Our (Llama)	0.9848	0.9771	0.95
RAG (GPT-4o)	0.7532	0.7856	0.6328
Our (GPT-4o)	0.9908	0.9898	0.9811

publicly available question-answering benchmark for log data, we generated 8427 QA pairs using GPT-4o across three datasets: OpenSSH (7317 pairs), Zookeeper (1,080 pairs), and Journal (30 pairs). Unlike (Chen et al., 2024), which summarizes cloud incidents, our approach processes raw logs directly. Unlike (Edge et al., 2024), which focuses on multi-hop reasoning in graph-based summarization, our method ensures temporal, relational, and causal consistency for RCA.

QA Categorization: The questions were categorized in two broad categories: (i) Local chunk dependent (ii) Complex log analysis questions (which require a comprehensive view (global) of the data. The chunk-dependent QA pairs are categorized in seven categories (i) factual-based (ii) Analytical

Table 4: Comparison of average input tokens to LLM and time (sec) required for single query.

Method	RAG	LC-RAG	Graph-RAG	HOLMES	Ours
Tokens	55.8K	11.2K	2.51M	2.51M	11.2K
Time	165.0	95.2	4437.5	3942.5	75.6

Reasoning (iii) Causal Reasoning (iv) Decision-based (v) Summarization (vi) Temporal Reasoning (vii) Anomaly Detection. Factual, decision-based, temporal, and anomaly detection questions have single-word answers, while others have up to 30 words. Complex log questions allow up to 50-word answers with 50-100 word justifications. See Appendix for details.

Baselines & Metrics: We compare against (i) RAG (ii) LC-RAG (iii) HOLMES (iv) Graph-RAG, and three LLMs: (i) Llama3.1-8B (ii) Mistral-7B (iii) GPT-4o. Accuracy is measured using (P_{BERT}), Recall (R_{BERT}), F1-Score (F_{BERT}) (Zhang et al., 2019), and human evaluation (based on relevance, clarity, accuracy, completeness, depth, and conciseness) (see Appendix A.3).

4.2 Implementation Details

All experiments were conducted on a computing system with the following specifications: In-

Table 5: **Ablation Study:** F1 Scores (F_{BERT}) for different question categories on the Zookeeper dataset using Our(Llama3.1-8B). ✓ = present, ✗ = ablated.

Configuration (Graph / Rank / Multi-step)	Fact-based	Analytical	Causal	Decision	Summarization	Temporal	Anomaly
✗ / ✗ / ✗	0.7710	0.8052	0.7631	0.7736	0.7468	0.7237	0.7781
✗ / ✓ / ✓	0.8040	0.8493	0.8164	0.8228	0.7988	0.8137	0.8208
✓ / ✓ / ✗	0.8811	0.8447	0.8485	0.8539	0.8265	0.8449	0.8441
✓ / ✓ / ✓	0.9072	0.8628	0.8527	0.8613	0.8370	0.8555	0.8503

Table 6: **Sensitivity Analysis:** F1 Scores (F_{BERT}) reported by varying seed nodes (Top-k and Next-top) on the Zookeeper dataset using Our(Llama3.1-8B).

Seed nodes (Top-k / Next-top)	Fact-based	Analytical	Causal	Decision	Summarization	Temporal	Anomaly
Top-20 / Next-0	0.8720	0.8557	0.8456	0.8522	0.8370	0.8521	0.8495
Top-15 / Next-15	0.8870	0.8597	0.8321	0.8561	0.8418	0.8504	0.8540
Top-10 / Next-15	0.8970	0.8559	0.8376	0.8572	0.8433	0.8522	0.8510
Top-5 / Next-0	0.8811	0.8447	0.8485	0.8539	0.8265	0.8449	0.8441
Top-5 / Next-15	0.9072	0.8628	0.8527	0.8613	0.8451	0.8555	0.8503
Top-5 / Next-25	0.9086	0.8620	0.8474	0.8553	0.8387	0.8572	0.8465
Top-5 / Next-35	0.9067	0.8621	0.8423	0.8532	0.8428	0.8591	0.8483

tel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz, 256GB RAM, Nvidia A30 GPU, Python 3.10, and Ubuntu OS 22.04.3. The hyperparameters used are: $\alpha = 0.85$, $\beta = 0.6$, $k = 5$, $n = 20$, top-p = 200 hyperedges, and $\epsilon = 10^{-6}$, with initial $P_{\text{rank}}^{t=0}(u)$ uniformly set to $1/N$. The OpenSSH and Zookeeper logs contain 638,947 and 74,273 entries, respectively. Logs were split into chunks of $T = 50,000$ lines for experiments, with all questions based on these chunked files. The embedding model used is sentence-transformers/all-mpnet-base-v2. For Journal logs, full files were used, with chunk sizes varying from 20,000 to 80,000 logs/lines. RAG baselines used $T = 50,000$ logs per file, chunked into 200-line segments. LC-RAG, Graph-RAG, and HOLMES used 50-line chunks. Retrieved top-5 chunks. RAG(GPT-4o), Our(GPT-4o), and Our(Mistral) experiments were conducted on 700-question subsets for Zookeeper and OpenSSH.

5 Results and Analysis

We evaluate HG-InsightLog’s performance across key aspects of log-based QA: (1) Questioning answering capability (a) w.r.t automatic metrics (b) w.r.t human & semantic metrics (2) Performance w.r.t. different levels of question complexity (3) Input token count and time taken

Performance analysis and comparison: We as-

sess performance across diverse question types and reasoning categories. Tables 1 and Table 2 compare our method against baselines using Precision (P_{BERT}), Recall (R_{BERT}), and F1 score (F_{BERT}). HG-InsightLog consistently outperforms all baselines across question complexities, achieving significant F1-score improvements. GraphRAG and HOLMES perform poorly because they rely on LLMs to extract entities and relationships for building a knowledge graph (KG). LLMs struggle to accurately extract this information and construct a complete and precise graph from non-NL (Non-NL) data. Since the KG serves as the input for generating answers, any inaccuracies or missing information in the graph prevent the LLMs from providing correct responses. Graph-RAG and HOLMES were unable to identify entities or relationships when using less advanced LLMs like Llama 3.1-8B, due to hallucinations, often returning empty lists, making graph construction impossible and terminating further steps. Therefore, graph creation was not possible with Llama 8B as LLM, so we used GPT-4o as base LLM for Graph-RAG and HOLMES experiments.

RAG and LC-RAG also show lower performance compared to our method. For fact-based QA, HG-InsightLog significantly improves F1 scores by leveraging query-specific personalization and semantic similarity, particularly for straightforward factual queries. Decision-based QA benefits from

its ability to maintain temporal order and incorporate key indicators, enabling precise log segment extraction for decision-making. For causal and temporal reasoning tasks, our hypergraph structure effectively preserves event relationships via hyperedges, enhancing contextual understanding. In summarization, HG-InsightLog prioritizes crucial nodes and hyperedges while maintaining temporal consistency, reducing context size without losing key information.

To evaluate generalizability, we tested our method using different LLMs (GPT-4o, Llama, and Mistral). There is a significant performance gap observed between RAG(GPT-4o), and RAG(Llama8B), as GPT-4o, has strong reasoning abilities, which can perform reasoning to arrive at the correct answer, given the correct retrieved chunks which may contain additional irrelevant information. However, if the same retrieved information is given to an LLM with weaker reasoning skills, like Llama, it may fail to reason, leading to incorrect or misleading results. This is where the InsightLog approach improves accuracy. Unlike standard RAG, InsightLog retrieves information by establishing causal, temporal and structural relationships beforehand. This allows even models with limited reasoning capabilities, like Llama, to correctly produce accurate responses. As a result, the accuracy of Our(Llama) becomes comparable to Our(GPT-4o), which is significantly higher than RAG(Llama). Our approach demonstrates how higher-quality structured context enables a smaller model (LLaMA-8B) to compete with or even surpass larger models like GPT-4o. HG-InsightLog is a robust, flexible, and model-agnostic framework that strengthens community-driven models like LLaMA3.1-8B, allowing them to handle complex tasks without depending on proprietary solutions. As open-source LLMs continue to advance, HG-InsightLog is designed to evolve alongside them, ensuring long-term adaptability and relevance.

Performance comparison on complex log analysis questions: HG-InsightLog outperforms other methods on complex log analysis tasks (Table 3) by constructing a multi-entity hypergraph and extracting query-relevant logs. This approach enables a global dataset view by linking attributes across timesteps, capturing intricate relationships. The multi-step personalization mechanism enhances reasoning by dynamically prioritizing hyperedges based on query relevance, ensuring retrieval of

only the most relevant information. The results show substantial performance gains over RAG and LC-RAG on human evaluation metrics (see Appendix A.3). P, R, and F-scores are excluded due to long responses and their inability to assess relevance, clarity, accuracy, completeness, depth, and conciseness, which are crucial for evaluating complex log analysis responses.

Input token count and time: In QA with LLMs like GPT-4, irrelevant information raises costs, while in models like Llama, it increases inference time. Efficient data filtering enhances performance, speed, and cost-effectiveness. Table 4 shows a significant token count reduction compared to baselines.

Sensitivity and Ablation study: Table 5 presents our ablation study results. Since the graph is integral to our ranking and PageRank algorithms, therefore, ranking cannot function without it. We conduct experiments under various best possible settings: (i) No Graph & No Ranking: Embed each line, perform a similarity search, extract one line, and pass it to the LLM. (ii) No Graph with Ranking: Embed each line, perform a similarity search, extract the top 200 lines (ranked by their similarity), and pass them to the LLM. (iii) Graph without Multi-Step Ranking: Construct the graph, rank it without the multi-step component, and extract 200 lines in temporal order. (iv) Full Method (Ours): Includes all components.

Table 6 presents the results of the sensitivity analysis, which analyzes the impact of varying top-k and next-top nodes.

6 Conclusion

We present HG-InsightLog, a novel hypergraph-based framework for efficient LLM-driven QA on large, non-natural log data. By constructing a multi-entity hypergraph and leveraging an enhanced hyperedge ranking mechanism, our method effectively reduces context size, preserves temporal structure, and prioritizes query-relevant information. Experimental results across multiple datasets demonstrate superior performance over baselines in factual, causal, and analytical reasoning tasks. Our approach ensures scalable, training-free QA while maintaining causal coherence, making it a robust solution for log-based analysis. Future work will explore integrating domain-specific retrieval and adaptive fine-tuning to enhance performance on a wide varieties of proprietary log datasets.

Limitations

The limitations of our method are:

1. Dependence on LLM pretrained knowledge:

Our method relies on the pretrained knowledge of the underlying LLM to interpret the provided ranked log text using its internal knowledge. Future work may lie in connecting private documentation to the method or fine-tuning LLM to address this limitations.

2. Scalability with very large continuous log files:

A limitation of our approach is scalability with very large continuous log files. While hypergraph construction is linear in complexity—each log line is processed once to extract attribute-value pairs and form a hyperedge—the resulting graph can grow too large for a single machine to handle. As the graph size exceeds a certain threshold, efficient scaling of hypergraph construction across large log datasets necessitates the use of distributed or parallel processing and computing resources.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1737–1749. IEEE.
- Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Mikhail Burtsev, and Evgeny Burnaev. 2024. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*.
- Sergey Brin. 1998. The pagerank citation ranking: bringing order to the web. *Proceedings of ASIS, 1998*, 98:161–172.
- Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117.
- Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2023. Empowering practical root cause analysis by large language models for cloud incidents. *arXiv preprint arXiv:2305.15778*.
- Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 674–688.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.
- Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479.
- Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.
- Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation.
- Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526.
- Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. 2022. An empirical study of log analysis at microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1465–1476.
- Shaohan Huang, Yi Liu, Carol Fung, Jiaying Qi, Hailong Yang, and Zhongzhi Luan. 2023. Logqa: Question answering in unstructured logs. *arXiv preprint arXiv:2303.11715*.
- Shaohan Huang, Yi Liu, Jiaying Qi, Jing Shang, Zhiwen Xiao, Carol Fung, Zhihui Wu, Hailong Yang, Zhongzhi Luan, and Depei Qian. 2024. Gloss: Guiding large language models to answer questions from system logs. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 91–101. IEEE.
- Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2024. Lilac: Log parsing using llms with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 1(FSE):137–160.
- Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O Arik. 2024. Long-context llms meet rag: Overcoming challenges for long inputs in rag. *arXiv preprint arXiv:2410.05983*.

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*, 33:9459–9474.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411.

Pranoy Panda, Ankush Agarwal, Chaitanya Devaguptapu, Manohar Kaul, et al. 2024. Holmes: Hyper-relational knowledge graphs for multi-hop question answering using llms. *arXiv preprint arXiv:2406.06027*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

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)*, pages 355–366. IEEE.

A Appendix

A.1 Dataset details:

The datasets used in our experiments are two public log datasets: OpenSSH dataset (Zhu et al., 2023), Zookeeper dataset (Zhu et al., 2023) and one private Journal logs dataset.

1. **OpenSSH Dataset** (Zhu et al., 2023): OpenSSH is a widely used tool for secure remote login via the SSH protocol. The dataset used in our experiments was collected from an OpenSSH server set up in our lab. Over a period of 28 days, logs were gathered to form the dataset.
2. **ZooKeeper** (Zhu et al., 2023): It is a centralized service that manages configuration data, naming, distributed synchronization, and group services. The log dataset used in this study was gathered by collecting logs from the ZooKeeper service in our lab environment, which consists of 32 machines and spans a period of 26.7 days.

Table 7: Statistics of a single journal log file.

Statistic	Value
Total log lines	74349
Total tokens	4079723
Average lines per second	37.32
Average tokens per second	2048
Average Lines per minute	2239.42
Average Tokens per minute	122883

3. **Journal log Dataset:** Journal files are system logs used to record events and messages generated by the system, kernel, and various services on Linux-based systems. Journal log files were collected from company’s internal systems which is private due to security reasons. The dataset consists of 6 big size log files collected on different days with log duration ranging from 12 minutes to 33 minutes. The statistics from one of the file is given in Table 7.

A.2 Case study

This case study analyzes a log file containing a Software Error (SWERR) (Fig. 4) to identify its root cause using three different methods: RAG, Graph-RAG, and HG-InsightLog (Table 8). Sensitive information in both the log snippet and responses is masked as ***** due to dataset confidentiality. The SWERR error includes the failure message: "Application Info: Invalid Port, port=25." To determine its root cause, it is necessary to identify related errors and failures within the log file. However, a standard LLM struggles with large log files due to context length limitations.

RAG Method: This approach splits the log file into smaller chunks and provides the chunk containing the SWERR error to the LLM. However, since logs are generated rapidly (thousands of lines per minute), the root cause may appear minutes before the SWERR error, making it inaccessible within a single chunk or not correctly retrieved across multiple chunks. In this case, the root cause occurs at 00:35:36, but RAG is limited to logs from 00:35:38, preventing accurate identification.

Graph-RAG Method: This method extracts relationships from the LLM using predefined examples in the prompt. However, due to LLMs being primarily trained on natural language, they fail to accurately capture relationships in log files—especially those not explicitly mentioned in

the prompt. Furthermore, Graph-RAG only establishes connections between closely occurring log lines and does not link distant events. For example, extracted relationships like (SWERR, contains, Invalid port, Port=25) and (SWERR, contains, Ks_SWERR_only) lack connections to earlier logs. During querying, Graph-RAG searches for related nodes but fails to establish a link with the actual root cause, leading to an inconclusive response: "Can't determine the root cause with given information."

HG-InsightLog Method: Our method scans the entire log file to identify related errors and failures associated with the SWERR error. It successfully retrieves logs from the large size having a global view beyond the chunk size limit of RAG and the context length restriction of LLMs. For example, the failure message "systemd-journald.socket: Failed to connect stdout to the journal socket, ignoring: No such file or directory" is crucial for root cause analysis but remains inaccessible to the RAG and Graph-RAG methods. HG-InsightLog, however, identifies it by creating a temporal and structural (via nodes and hyperedges) connection in error nodes, port issues, and socket failures. By providing both the SWERR error and related failure logs to the LLM, HG-InsightLog enables accurate root cause identification, proving to be the most effective method for this analysis.

Case study 2: Let's consider another question: *A system failure occurred at 03:15:42. Identify the earliest sign of this failure from any component in the system logs, even if it occurred hours or days earlier, and explain how it contributed to the final failure.*

Why All Other Algorithms Will Fail:

1. Traditional RAG Will Fail:

Reason: RAG retrieves information based on chunked retrieval, but since the root cause could be far apart from the failure event, RAG won't retrieve relevant logs unless they exist in the same chunk. Failure Mode: Will only retrieve logs immediately before or after 03:15:42, missing critical early warning signs.

2. GraphRAG Will Fail:

Reason: GraphRAG builds entity-relationship graphs, but logs often contain no explicit entity relations connecting long-term system-wide dependencies.

Failure Mode: The early warning signs may be in a different subsystem or involve indirectly related attributes, which GraphRAG won't link properly.

3. Personalized PageRank and LexRank Will Fail:

Reason: These algorithms rank logs based on similarity and importance, but they lack the ability to model temporal causality. Failure Mode: They will rank recent logs higher, overlooking the earliest indicators of failure.

4. Standard Log Summarization Will Fail:

Reason: Summarization techniques focus on the most frequent or central events rather than tracing low-frequency anomalies. Failure Mode: The earliest weak signal may be a rare, infrequent log entry that standard summarization will discard.

Why HG-InsightLog Succeeds:

Temporal Sequence is Preserved: Unlike RAG, it does not disrupt log order and allows retrieval of distant historical logs that contributed to failure.

Multi-Step Query-Based Ranking: Unlike GraphRAG, it prioritizes logs based on query-specific causal relevance, not just static entity connections.

Hypergraph Captures Complex Relations: Unlike PageRank, it identifies weak signals even if they are buried in logs far apart in time.

Anomaly Detection & Contextual Reduction: Unlike summarization methods, it preserves rare but critical logs instead of discarding them due to low frequency.

Few more example questions for complex log analysis:

1. *What trends can be identified from the frequency of different types of login errors (e.g., permission denied, command not found)?*
2. *How did multiple independent warnings in different system components correlate to the final system failure?*

A.3 Evaluation by Domain Experts

In the evaluation of the complex log analysis category questions, human evaluators, who were domain experts played a key role since the dataset lacked predefined questions and correct answers. They were responsible for creating questions and assessing the accuracy of the system's predicted answers. The evaluation process worked as follows:

Expert evaluators were given access to the log files, the questions, and the predicted answers. Their main task was to evaluate the predicted answer and provide scores. The evaluators assessed the responses to the question based on six key matrices: relevance, clarity, accuracy, completeness, conciseness and analytical depth. Each met-

```

Oct 30 00:35:36 ***** systemd[1]: Starting Journal Socket...
Oct 30 00:35:36 ***** systemd[329]: systemd-journald.socket: Failed to connect stdout to the journal socket, ignoring: No such file or directory
Oct 30 00:35:36 ***** systemd[1]: Listening on udev Control Socket.
Oct 30 00:35:36 ***** systemd[1]: Listening on udev Kernel Socket.
Oct 30 00:35:36 ***** systemd[1]: Listening on User Database Manager Socket.
Oct 30 00:35:36 ***** systemd[1]: Listening on Journal Socket.
Oct 30 00:35:36 ***** systemd[1]: Huge Pages File System was skipped because of a failed condition check (ConditionPathExists=/sys/kernel/mm/hugepages).
Oct 30 00:35:36 ***** systemd[1]: Mounting POSIX Message Queue File System...
Oct 30 00:35:36 ***** systemd[1]: Mounting Kernel Debug File System...
Oct 30 00:35:36 ***** systemd[1]: Mounting Kernel Trace File System...
Oct 30 00:35:36 ***** systemd[1]: Starting Create List of Static Device Nodes...
Oct 30 00:35:36 ***** systemd[1]: Starting Load Kernel Module configs...
Oct 30 00:35:36 ***** systemd[1]: Starting Load Kernel Module drm...
Oct 30 00:35:36 ***** systemd[1]: Starting Load Kernel Module fuse...
Oct 30 00:35:36 ***** systemd[1]: Starting RPC Bind...

```

(a)

```

Oct 30 00:35:38 ***** esalbase[450]: portType=OSC2 no rate
Oct 30 00:35:38 ***** esalbase[450]: GetDuplex:101
Oct 30 00:35:38 ***** esalbase[450]: portType=OSC2 no duplex
Oct 30 00:35:38 ***** esalbase[450]: portType=OSC2 defaultVlan=4000
Oct 30 00:35:38 ***** esalbase[450]: portType=OSC2 unknown port access
Oct 30 00:35:38 ***** esalbase[450]: SettingNotFoundException:MAIN.portType.OSC2.memberVlansnot found
Oct 30 00:35:38 ***** systemd[1]: Starting Pkt Handler App startup service file...
Oct 30 00:35:38 ***** SWERR[***]: =====SWERR Start=====
Sequence Number: **
Level: KS_SWERR_ONLY
TID: esalbase(0)
PID: esalbase(0x1ca)
PPID: systemd(0x1)
Core: 0
File: *****
Stack Trace:
/usr/lib/libswerr.so.0(_ZN12SwerrContext10init_earlyEv+0x90)[0x14b0f4eb8550]
/usr/lib/libswerr.so.0(_ZN12SwerrContext4initEv+0x9)[0x14b0f4ebbbf9]
/usr/lib/libswerr.so.0
(_ZN5SwerrCIENS_10SwerrLevelERKNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEIS8_PK12SwerrContext+0x16f)[0x14b0f4ebbd7f]
/usr/bin/esalbase(_ZN18EsalBoardFramework11ISPortValIdEj+0x1e3)[0x556f1d73ab73]
/usr/bin/esalbase(_ZN18EsalBoardFramework14HandlePortProvE9InterFaceRK15PortProvMessage+0x3c4)
[0x556f1d74e3d4]
/usr/bin/esalbase(_ZN13EsalStaticCfg14ConfigurePortsEv+0x9b)[0x556f1d6e977b]
/usr/bin/esalbase(_ZN18EsalBoardFramework9BoardInitEv+0x45)[0x556f1d73a355]
/usr/bin/esalbase(_ZN12EsalBaseMain+0x11c)[0x556f1d6cc39c]
/usr/bin/esalbase(main+0x9)[0x556f1d6c7129]
/lib/libc.so.6(+0x2d57b)[0x14b0f451657b]
/lib/libc.so.6(__libc_start_main+0x80)[0x14b0f4516630]
/usr/bin/esalbase(_start+0x25)[0x556f1dec7185]
Application Info:
Invalid Port, port=**
=====SWERR End=====
Oct 30 00:35:38 ***** pktHandler[498]: Libconfig::openAndRead status open path /usr/local/fnc/pkthandler/02_00_00/pkthandler.cfg
Oct 30 00:35:38 ***** pktHandler[498]: Libconfig::openAndRead /usr/local/fnc/pkthandler/02_00_00/pkthandler.cfg
Oct 30 00:35:38 ***** pktHandler[498]: GetRxFilters:116 mac=FF:FF:FF:FF:FF:FF macMask=FF:FF:FF:FF:FF:FF vlan=2003 vlanMask=4095

```

(b)

Figure 4: Excerpt from the journal log file showing: (a) log entries related to the root cause of the *SWERR*, and (b) log entries containing the *SWERR* event itself. A large number of intermediate log lines between (a) and (b) have been omitted for brevity. The snippet includes only the information necessary to support the case study analysis.

ric was scored on a scale of 0 to 1, with the aim of providing a comprehensive evaluation of the answers. The evaluators carefully considered whether the response directly addressed the question (relevance), whether the answer was easy to understand and well-organized (clarity), the correctness of the claims made (accuracy), whether the answer fully explored the topic (completeness), the reasoning behind the answer (analytical depth), and whether the answer was concise (succinct and free of unnecessary information while still being complete). For each response, a final score out of 6 was assigned, and the final score for each response was calculated by the dividing the total score of a response by 6. Finally, the average score of all the responses was reported.

A.4 Example multi-dimensional hypergraph

To understand how we represent the hyperedges in the hypergraph, we demonstrate it by using an example taken from the journal dataset. Using Figure 4 and Figure 5, we demonstrate how the dimensions, Nodes, hyperedges in a hypergraph looks like. Dimensions are date, time, in-

terface/components, and message. For example, first hyperedge is represented as ('Date: Oct 30', 'Time: 00:35:36', 'interface:*****systemd[1]', 'message: Starting Journal Socket'). Similarly, second hyperedge is represented as ('Date: Oct 30', 'Time: 00:35:36', 'interface:*****systemd[329]', 'message: systemd-journald.socket: Failed to connect') and so on. Here, 'Date: Oct 30', 'Time: 00:35:36', 'interface:*****systemd[1]', 'message: Starting Journal Socket' are nodes. All the hyperedges collectively form a multi-dimensional hypergraph.

A.5 Dataset Generation

Question-answer pair generation using log files was done by LogAQ(Huang et al., 2023) but the questions type were restricted to only data extraction from chunks. (Huang et al., 2024) gave an algorithm to generate questions from log files in three stages - generate questions by giving raw log file to LLM and asking to generate 5 questions, second stage generate answers and in third stage questions are filtered based on pre-determined rules like answer length, unclear answer((e.g., "I don't know"

Table 8: **Case study:** Comparison of response provided by the RAG Method, Graph-RAG and Our Method on complex log analysis question on Journal dataset.

<p>Query: From the Given log with software errors (e.g., SWERR) in the journal log, explain the root cause of this error.</p>
<p>Prompt used: Analyze the provided journal log file containing SWERR ie, software error logs. For the first occurrence of an SWERR log in the file (1) Determine the root cause of this SWERR log (i.e., explain what caused it to occur). (2) Review all preceding logs in sequential order and assess whether this SWERR is caused or influenced by any prior errors. Also, pinpoint the first error in the log file that could have resulted in this SWERR.</p>
<p>RAG Method:</p> <p>1. The root cause of this SWERR appears to be related to a failure in the 'IsPortValid()' function within the 'EsalBoardFramework'. The stack trace suggests that during the initialization sequence of the board framework, a validation check on ports was performed, which did not pass, triggering the SWERR logging mechanism.</p> <p>The core issue could be due to:</p> <ul style="list-style-type: none"> - An invalid port configuration or status at the time of invocation. - Software logic errors in the function responsible for validating ports. <p>2. Reviewing Preceding Logs for Influences on SWERR:</p> <p>Examining the logs preceding the earliest SWERR log indicated: Oct 30 00:35:38 ***** systemd-logind[444]: New seat seat0.</p> <p>This log does not indicate any errors or issues that could influence the SWERR as it simply states that a new seat (session) has been created.</p> <p>Additionally, no other errors were present in the logs prior to the SWERR, such as hardware failures, port failures, or configuration issues. The log entries leading up to the SWERR involve the configuration and initialization of various components, indicating a standard operation rather than a problematic event.</p> <p>Conclusion:</p> <p>The first error in the log file that could potentially have resulted in the SWERR is tied to the port validation function 'IsPortValid()', which indicates that there may have been issues with the hardware or software configuration of the ports. However, no explicit prior errors are noted to directly influence this SWERR event, suggesting the issue is self-contained within the port validation logic.</p>
<p>Graph-RAG method: -</p> <p>Entities and relationship extracted examples: (SWERR, contains, Invalid port, Port=25) ,(SWERR, contains,Ks_SWERR_only) , (Esalbase[458], references, Porttype=Osc2 Unknown Port Access) , (Pkthandler[498], references, Libconfig)</p> <p>Response: I'm sorry, but with the provided information, I can't determine the root cause of the software error in the journal log. Please provide more specific details or context about the error entries in the log.</p>
<p>Our Method:</p> <p>1. The root cause of the first SWERR log is that the systemd-journal.socket failed to connect stdout to the journal socket due to the journal socket being absent (error message: "No such file or directory").</p> <p>2. Prior to this SWERR, several logs were focused on standard output types being updated from syslog to journal and setting up various slices, watching sockets, and listening units by systemd, along with starting some services and mounting filesystems. But none of these specifically caused or influenced the SWERR related to systemd-journal.socket directly.</p> <p>The first error in the log file with time information that could have resulted in this SWERR is: Date: Oct 30 Time: 00:35:36 interface: ***** Message: systemd[329]: systemd-journal.socket: Failed to connect stdout to the journal socket, ignoring: No such file or directory</p>

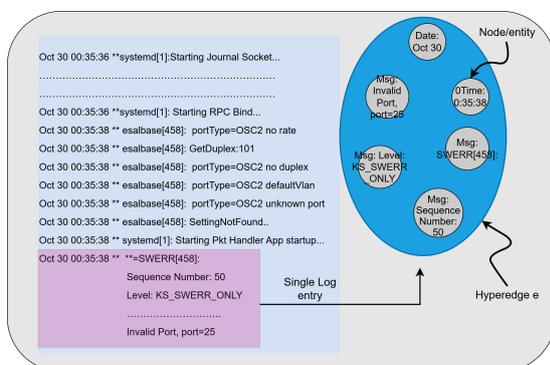


Figure 5: This figure presents a segment of system log entries on the left, while the right side illustrates a corresponding log entry using a node and hyperedge representation.

or “no information”). This approach doesn't have control on kind of questions created as well as restricting answer length to 7 excludes question categories like analytical reasoning and summarization. We generated two sets of questions for Zookeeper and OpenSSH datasets - Local chunk dependent questions and Complex log analysis questions. For Journal dataset we only generated Complex log based questions.

A.5.1 Chunk based questions

We used GPT-4o to generate question and answers for different categories and complexity levels, each requiring distinct analysis to arrive at answers. The categories of question generated are as follows with

their descriptions and challenges:

1. **Fact-based:** Questions that extract key information data such as events, values, or occurrences. These questions are challenging due to inconsistent log entries, huge log size.
2. **Analytical Reasoning:** Questions that require interpreting patterns or relationships in the log data to derive insights or conclusions. Answers are restricted to 30 words. These questions are challenging due to irrelevant and repetitive entries in log data, sparse connections present difficulties in finding patterns.
3. **Causal Reasoning:** Questions that identify potential causes or triggers for specific events or trends found in the logs. Answer are maximum 30 words. These questions are challenging as it requires inferring context information for events to answer such questions.
4. **Decision-based:** Questions on decisions that requires judgment and inference from the log data. These questions are challenging as it requires combining information from multiple log entries spread across time.
5. **Summarization:** Questions on summarizing information from the logs. Answer is generated on less than 50 words. Challenges - Important events are sparsely distributed.
6. **Temporal Reasoning:** Questions based on Temporal Reasoning task to Understand the sequence and timing of events and patterns found in the logs. These questions are challenging as it requires understanding gaps or inconsistencies in the sequence of events.
7. **Anomaly Detection:** Questions that Detect outliers or unusual behaviors found in the logs. These questions are challenging as it requires understanding of normal behavior to detect rare events.

We created chunks of log file based on timestamps from the log files. For both Zookeeper and OpenSSH we took 1 hour time duration as chunks. For zookeeper 150 chunks were used and for OpenSSH 50 chunks were used. For each chunk 3 questions were created for each of the 7 category. We manually verified the question-answer pairs correctness from the chunks and re-created the questions if pair was incorrect.

A.5.2 Complex log analysis questions

To analyze the global and multi-hop question answering capabilities of our method on whole log file we created questions spanning whole document. Human experts created 30 complex questions for each of three dataset. The questions were created such that it's of global nature where answer should combine information from different places having a global view of the file.

A.5.3 Prompt used for data generation

prompt = f"""You are provided with a segment of data containing multiple logs from a log file. Your objective is to perform the following tasks of Question-Answer pair generation while adhering strictly to the given requirements:

Requirements:

1. Ensure that all questions target local details in the log file.
2. Ensure the questions focus on patterns, trends, or key decision points observable in the logs.
3. For each question, provide answer that reflects insights or conclusions derived from the log data.

Task 1: Formulate 3 questions answer based on summarization task found in the logs. For each question, provide a concise (maximum 50 words) answer.

Example Question:

Q1. Summarize all major events from the logs between 2:15 AM to 2:30 AM.

Task 2: Generate 3 question-answer pairs that require analytical reasoning based on patterns observed in the log file. Formulate questions that demand analytical thinking and inference. Each answer should be concise (maximum 30 words). The questions should be based on key observations related to user behavior, security events, or notable system activities.

Example Question:

Q1. What recurring patterns of invalid login attempts can be identified across the logs?

Q2: Which is the most severe error in the data?

Task 3: Formulate 3 question-answers pairs based on Causal Reasoning task to Identify potential causes or triggers for specific events or trends found in the logs. For each question, provide a concise (maximum 30 words) answer.

Example Question:

Q1. Why did the SSH daemon disconnect the connection from 75.138.16.23 after multiple failed attempts?

Table 9: Below is a comparative table highlighting how GraphRAG, HOLMES, and our algorithm (HG-InsightLog) perform on three challenging RCA questions derived from log data. Each row shows the complex question, the true root cause (based on multi-step, time-ordered logs), how GraphRAG and HOLMES respond (and why they fail), and how HG-InsightLog succeeds by maintaining complete attribute-value relationships and proper temporal sequences. All facts from the original explanation are preserved.

Question & Context	Ground-Truth Root Cause	GraphRAG (Wrong)	HOLMES (Wrong)	HG-InsightLog (Correct)
Q1: SWERR at Oct 30 00:35:38 Invalid Port errors (25-27) appear just before SWERR.	Invalid/unconnected ports initialized due to config errors; caused cascading SWERRs.	Partial mention of "invalid port," no full causal chain. Misses earlier logs.	Mentions SWERR but omits port misconfig. Triple-based parse lacks context chaining.	Traces SWERR to earlier port config errors (25-27). Captures order and all value pairs.
Q2: Why did mslm_chassis terminate? Log shows missing config file then exit.	Missing mslmShelf.cfg caused PID 508 exit; ZMQ context terminated as a result.	Mentions file error, but fails to link it to process exit or ZMQ issue.	Mentions exit and ZMQ error, not file cause. Misses non-triple logs.	Chains missing file → service exit → ZMQ error. Hypergraph links all by PID and time.
Q3: Why was lmp00 brought down? Peer (port 2) disabled in kernel log, then user log says lmp00 down.	Peer (lmp00-peer, port 2) went offline; system shut lmp00 as safeguard.	Cannot match "port 2" to lmp00. Graph too fragmented.	Sees two events, but fails to connect without explicit attribute matching.	Links peer port to lmp00, capturing full cause. Attributes retained across logs.

Task 4: Create 3 question-answers pairs based on Temporal Reasoning task to Understand the sequence and timing of events and patterns found in the logs. For each question, provide a concise, one word answer.

Example Question:

Q1. What event occurred first in the log sequence after event XXXX?

Task 5: Create 3 decision-based question-answer pairs based on decisions that requires judgment and inference from the log data. For each question, provide a concise, one-word answer. The questions should capture key observations related to user behavior, security events, or notable system activities.

Example Question:

Q1. Which IP made the highest number of failed login attempts?

Task 6: Generate 3 question-answers pairs based on Anomaly Detection task that Detect outliers or unusual behaviors found in the logs. For each question, provide a concise, one word answer. Example Question: Q1. What is the most unusual activity recorded in the logs?

Task 7: Create 3 simple, clear data extraction based question-answer pairs and provide corresponding one-word answer based on the key information in the log file. Requirements: Local Scope: Each question should target unique, local details in the log file, ensuring that there is only one correct answer across the entire dataset for each question.

Unambiguous: The questions must be designed in a way that eliminates ambiguity. Each question should have a clear, definitive answer based on the log content.

One-Word Answers: Ensure that the answer to

each question is a single word derived directly from the information within the log file.

File-chunk-specific: The questions should be specific to the details of the current log file chunk. This means the same question cannot be applicable to multiple log files; the question must rely on information unique to this specific file.

Example Question:

Q1: What is the default value for initLimit?

Note: Avoid general questions that could apply to multiple log entries, and also don't use timestamps.

For each task, respond in the following format: [Category]

Q1: [Question] A1: [Answer]

Q2: [Question] A2: [Answer]

Q3: [Question] A3: [Answer]

Now, generate questions and answers for the following segment of data or log file: document"""

A.6 Addressing Limitations Related to Pretrained LLM Knowledge

As noted in the paper, the framework leverages the pretrained knowledge of large language models (LLMs) to interpret reduced-context logs. This reliance can limit performance in domains where the LLM lacks prior exposure—such as proprietary or highly specialized systems.

To mitigate this limitation without modifying the framework architecture or retraining the LLM, we propose two complementary strategies:

Fallback via Generalized Node Representations: When domain-specific terms are not recognized by the LLM, our multi-entity hypergraph representation groups attribute-value pairs (or entire log

lines) into higher-level semantic units. This process is embedded in the node construction logic and does not introduce a new algorithmic component. Instead, each node acts as a generalized representation, enabling the model to reason over structural relationships and anomalies even in the absence of domain-specific understanding.

Minimal Supplementary Documentation: Although the LLM is not retrained, brief supporting documentation—such as glossaries or concise system descriptions—can be appended to the prompt. This lightweight augmentation provides sufficient context for the model to interpret domain-specific terminology or abbreviations, enhancing comprehension while maintaining a training-free setup.

These strategies preserve the framework’s flexibility and zero-shot capability while extending its applicability to domains beyond the LLM’s original training distribution.