2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE) | 979-8-3503-9302-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISSRE66568.2025.00019

AetherLog: Log-based Root Cause Analysis by Integrating Large Language Models with Knowledge Graphs

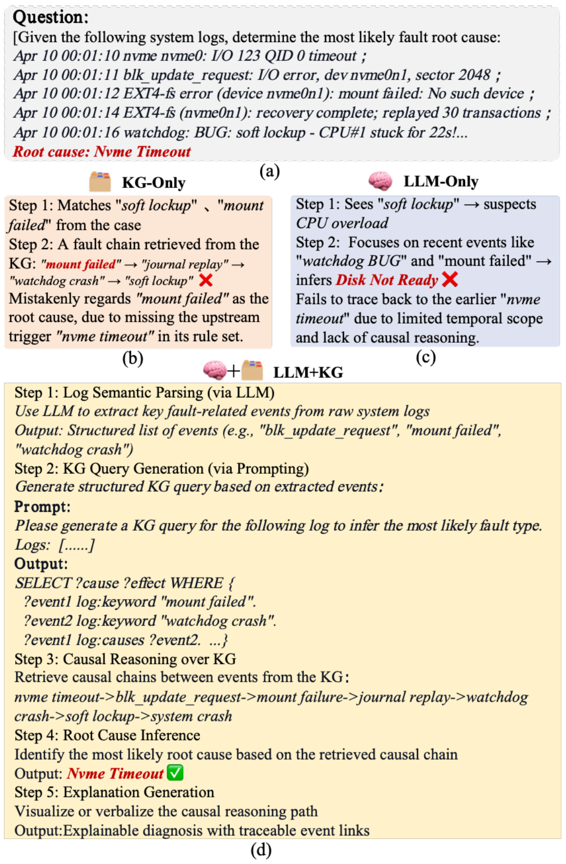
Tianyu Cui[†], Ruowei Fu[†], Changchang Liu[†], Yuhe Ji[†], Wenwei Gu[†], Shenglin Zhang[†§*], Yongqian Sun[†¶], Dan Pei[‡]
[†]Nankai University, Tianjin, China [‡]Tsinghua University, Beijing, China
[§]Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin, China
[¶]Tianjin Key Laboratory of Software Experience and Human Computer Interaction, Tianjin, China
Emails: {cuitianyu, furuowei, 2113411, yuheji}@mail.nankai.edu.cn,
gwwdaxue@gmail.com, {zhangsl, sunyongqian}@nankai.edu.cn, peidan@tsinghua.edu.cn

Abstract*—Log-based fault root cause analysis (RCA) is paramount for ensuring the reliability of large-scale software systems. While small language model (SLM)-based methods offer efficiency and ease of deployment, their limited generalization across diverse fault scenarios often hinders their effectiveness. Conversely, large language model (LLM)-based methods demonstrate strong semantic understanding but can suffer from inaccuracies and hallucinations due to a lack of domain-specific knowledge. To overcome these limitations, we present *AetherLog*, a novel RCA framework synergistically integrating LLMs with knowledge graphs (KGs). In an offline phase, *AetherLog* employs LLMs to extract fault-relevant entities and relations, constructing a compact and semantically aligned KG through embedding-based clustering and normalization. During online analysis, the framework leverages an LLM to summarize fault logs and extract pertinent entities. Subsequently, it retrieves semantically similar entities from the KG to enrich the context and formulates context-enhanced prompts, leading to more accurate RCA. Extensive experiments conducted on two real-world datasets demonstrate that *AetherLog* consistently surpasses state-of-the-art baselines, achieving F1-scores of 0.93 and 0.97. These results represent significant improvements of 6% and 8% over the best existing methods, respectively, demonstrating *AetherLog*'s effectiveness and generalizability in log-based fault RCA.

***Index Terms*—Knowledge Graph, Large Language Model, Root Cause Analysis**

I. Introduction

In today's large-scale software systems, increasing complexity and intricate interdependencies among components have made system faults more frequent and challenging to diagnose [1]. In this context, Root Cause Analysis (RCA) refers to the process of analyzing and identifying the underlying causes of system faults, enabling precise localization, effective repair, and efficient system recovery [2]–[8]. System logs, as the most direct and granular record of system behavior, are a critical data source for RCA. Modern systems continuously generate vast amounts of logs, rich with temporal and semantic information. Through careful analysis of these logs, it is possible to reveal fault origins and dependencies [9]–[12]. Consequently, log-based RCA has become a central research topic in both academia and industry.



Question:
[Given the following system logs, determine the most likely fault root cause:
```
Apr 10 00:01:10 nvme nvme0: I/O 123 QID 0 timeout ;
Apr 10 00:01:11 blk_update_request: I/O error, dev nvme0n1, sector 2048 ;
Apr 10 00:01:12 EXT4-fs error (device nvme0n1): mount failed: No such device ;
Apr 10 00:01:14 EXT4-fs (nvme0n1): recovery complete; replayed 30 transactions ;
Apr 10 00:01:16 watchdog: BUG: soft lockup - CPU#1 stuck for 22s!...
```
Root cause: Nvme Timeout

(a)

KG-Only

Step 1: Matches "*soft lockup*" 、"*mount failed*" from the case
Step 2: A fault chain retrieved from the KG: *"**mount failed**" → "journal replay" → "watchdog crash" → "soft lockup"* ✗
Mistakenly regards *"mount failed"* as the root cause, due to missing the upstream trigger *"nvme timeout"* in its rule set.

(b)

LLM-Only

Step 1: Sees "*soft lockup*" → suspects *CPU overload*
Step 2: Focuses on recent events like "*watchdog BUG*" and "mount failed" → infers ***Disk Not Ready*** ✗
Fails to trace back to the earlier "*nvme timeout*" due to limited temporal scope and lack of causal reasoning.

(c)

LLM+KG

Step 1: Log Semantic Parsing (via LLM)
Use LLM to extract key fault-related events from raw system logs
Output: Structured list of events (e.g., "blk_update_request", "mount failed", "watchdog crash")
Step 2: KG Query Generation (via Prompting)
Generate structured KG query based on extracted events:
Prompt:
Please generate a KG query for the following log to infer the most likely fault type.
Logs: [......]
Output:
```
SELECT ?cause ?effect WHERE {
  ?event1 log:keyword "mount failed".
  ?event2 log:keyword "watchdog crash".
  ?event1 log:causes ?event2.  ...}
```
Step 3: Causal Reasoning over KG
Retrieve causal chains between events from the KG:
nvme timeout->blk_update_request->mount failure->journal replay->watchdog crash->soft lockup->system crash
Step 4: Root Cause Inference
Identify the most likely root cause based on the retrieved causal chain
Output: ***Nvme Timeout*** ✅
Step 5: Explanation Generation
Visualize or verbalize the causal reasoning path
Output:Explainable diagnosis with traceable event links

(d)

Fig. 1: Comparison of using LLMs, KGs, and their combination for log-based fault RCA.

Early approaches to log-based RCA can be broadly grouped into two categories. (1) **Rule-based methods** employ hand-

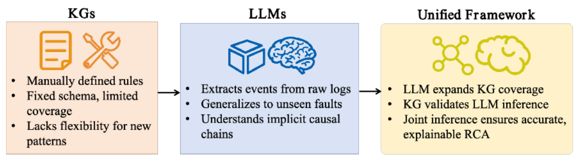
* Shenglin Zhang is the corresponding author.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

crafted extraction rules to identify important events and causal paths, as seen in Ladra [13] and LogRule [14]. (2) **Machine learning-based methods** utilize classifiers or regressors to perform fault localization, such as Log3C [15] and LogRCA [7]. However, both paradigms depend on shallow features or manually crafted patterns, limiting their ability to generalize to complex and ever-evolving log data. To address these limitations, LogKG [16] introduces knowledge graphs (KGs), providing a more structured and semantically rich representation of entities and their relationships. However, KGs that rely on fixed extraction rules face two main challenges: (1) *incomplete coverage*, where previously unseen patterns or log templates are ignored, and (2) *high maintenance costs*, as frequent manual updates are required to keep pace with changes in system behavior. Figure 1(a) illustrates log events during a representative fault in a storage subsystem. The fault sequence starts with an *nvme timeout* (the true root cause), inducing severe I/O latency and triggering downstream faults—first a block device I/O error, then a *mount fault* due to the missing device. Despite an attempted recovery via *journal replay*, the system's condition deteriorates, culminating in a *watchdog crash* and a *soft lockup*, together indicating a system collapse. As shown in Figure 1(b), the KG-based RCA method identifies events like *"soft lockup"* and *"mount failed"* by matching them to KG entries and reconstructs a partial fault chain: *"mount failed"* $\rightarrow$ *"journal replay"* $\rightarrow$ *"watchdog crash"* $\rightarrow$ *"soft lockup"*. However, this approach incorrectly attributes *"mount failed"* as the root cause, overlooking the actual trigger—*"nvme timeout"*—which is absent from the KG due to incomplete rule coverage.

Recent advances have investigated the application of large language models (LLMs) to a range of log analysis tasks, including anomaly detection and log parsing [18], [19], [21]–[28]. Approaches leveraging LLMs can be broadly categorized into three representative paradigms: (1) **Direct generation**, where LLMs produce analysis results directly from raw log sequences without requiring additional context or fine-tuning, as seen in LogPrompt [25] and LogGPT [28]; (2) **Fine-tuning**, which specializes pre-trained LLMs on system-specific log data through supervised training [18]; (3) **Retrieval-Augmented Generation (RAG)**, which supplements LLMs with external knowledge—such as historical logs or structured information—during inference, as exemplified by LogRAG [29]. While these paradigms demonstrate considerable promise, each faces inherent limitations: direct generation methods may hallucinate plausible but incorrect outputs; fine-tuning approaches require substantial labeled data; and RAG-based techniques depend critically on the quality and relevance of retrieved context. For instance, Figure 1(c) illustrates RCA under the direct generation paradigm. Here, the LLM-based approach correctly identifies recent log events (e.g., *"soft lockup"*, *"watchdog BUG"*, *"mount failed"*) and infers a symptom-level root cause such as "disk not ready." However, it fails to trace the fault chain back to the earlier *"nvme timeout"* event, which is the actual root cause. This limitation arises from the LLM's restricted temporal focus—giving preference to recent log entries—and its lack of explicit causal reasoning capabilities, hindering comprehensive fault diagnosis.

These findings highlight the potential of integrating KGs with LLMs for robust RCA. As demonstrated in Figure 1(d), the combination of LLMs and KGs enables accurate and explainable RCA. In this approach, the LLM first semantically parses raw logs to extract key fault-related events (e.g., *"blk_update_request"*, *"mount failed"*, *"watchdog crash"*), which are then formulated as structured KG queries via prompt engineering. These queries retrieve causal chains from the KG, capturing the fault's complete progression—from the initial *"nvme timeout"* to the final *"soft lockup"*. Leveraging the causal graph, the system accurately infers *nvme timeout* as the root cause and produces a step-by-step rationale linking each event. This synergy combines the adaptive event understanding of LLMs with the structured and traceable reasoning of KGs, thereby reducing hallucinations and improving explainability. As summarized in Figure 2, this unified paradigm effectively addresses the shortcomings of both KG-only and LLM-only approaches. To this end, **we propose a unified framework that integrates KGs and LLMs for log-based RCA,** aiming to harness the semantic reasoning ability of LLMs alongside the structured domain knowledge encapsulated in KGs.


KGs
Manually defined rules
Fixed schema, limited coverage
Lacks flexibility for new patterns
LLMs
Extracts events from raw logs
Generalizes to unseen faults
Understands implicit causal chains
Unified Framework
LLM expands KG coverage
KG validates LLM inference
Joint inference ensures accurate, explainable RCA
Fig. 2: Enhancing Log-Based Fault RCA via LLM and KG

Despite the promise of LLM-KG integration, two key challenges remain in practical deployment: (1) *Semantic redundancy among extracted entities:* LLMs may identify diverse surface expressions for the same underlying concept, resulting in redundant nodes within the KG and undermining effective reasoning; (2) *Missing or irrelevant information during unsupervised inference:* Direct extraction can overlook critical fault entities or introduce noise, diminishing analysis quality.

Our main contributions are as follows:

- We propose a *semantic entity aggregation mechanism* to address challenge (1). This approach combines LLM-based extraction, domain-specific embeddings, and density-based clustering to identify and merge semantically equivalent entities, reducing node redundancy and enhancing KG consistency.
- We introduce a *context-aware entity recall mechanism* targeting challenge (2). By integrating LLM-based summarization, entity extraction, and embedding-based matching, this mechanism retrieves the most relevant KG nodes, significantly improving recall and boosting both the coverage and accuracy of downstream RCA.
- We present *AetherLog*, the first framework to integrate LLMs and KGs specifically for log analysis. We validate its effectiveness with comprehensive experiments on two

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Summary of Representative Works

Proposal	Objective	Baseline	Reason
SLM-Based Log Analysis			
Log3C [15]	Cascading clustering and correlation analysis for RCA	✗	Relies on system-wide service metrics for correlation, which becomes infeasible when such metrics are unavailable or unreliable.
LADRA [13]	Log-based abnormal task detection and RCA in Spark	✗	Designed for Spark environments with only four fixed fault types; lacks generalizability to more diverse root causes.
LogKG [16]	Graph-based RCA using knowledge graphs	✓	Utilizes KG structures for log-level fault reasoning, similar to our approach, making it a suitable baseline.
LogRule [14]	Rule-based log template mining and RCA	✓	Produces interpretable rules through pattern mining, aligning with our goal of explainable RCA.
LogRCA [7]	Semi-supervised RCA using PU learning and log ranking	✗	Focuses on log ranking with PU learning; outputs Top-n logs instead of root cause categories, making it incompatible with our task.
LLM-Based Log Analysis			
LLMParser [23]	Few-shot log parsing with LLMs	✗	Focuses on log parsing and structure extraction, not RCA.
UniLog [19]	In-context learning for log generation and prompt creation	✗	Designed for development-time log generation rather than runtime RCA.
SelfLog [27]	Self-evolving log parsing via N-gram grouping	✗	Targets adaptive parsing without annotations; goal and output are unrelated to RCA.
LogPrompt [25]	Prompt-based zero-shot log parsing and anomaly detection with in-context learning	✓	Its prompt-based framework supports flexible adaptation to RCA tasks, making it a suitable baseline through prompt modification.
LogGPT [28]	Zero and few-shot anomaly detection using ChatGPT	✗	Shares similar prompting techniques with LogPrompt, but lacks explicit CoT reasoning. We choose LogPrompt as the representative baseline.
LogRAG [29]	Retrieval-augmented anomaly detection with LLMs	✓	Introduces semantic retrieval of historical logs as external context, consistent with our KG-enhanced RCA strategy.

large industrial datasets from Alibaba and China Mobile, containing 2,671 and 93 fault cases, respectively. AetherLog achieves F1-scores of 0.93 and 0.97, outperforming the best baselines by up to 6% and 8%.

The source code and datasets for AetherLog are publicly available[1].

II. Related Work

To better understand recent progress in log-based RCA, we categorize existing related works into two main research lines: **(1) SLM-Based Log Analysis** and **(2) LLM-Based Log Analysis**. Table I summarizes representative works in both categories, highlighting their objectives, baseline selection, and rationale.

SLM-Based Log Analysis has long been a core area in system reliability and observability. In recent years, several scholars have proposed various methods. Log3C [15] uses cascading clustering and correlation analysis for RCA but relies on system-wide service metrics for correlation. This becomes less effective when service-level metrics are unavailable. LADRA [13] introduces a method to correlate log sequences with fault types, excelling in anomaly detection and diagnosis within Spark by focusing on four specific root causes: CPU, memory, network, and disk issues. However, its support for only these four specific root causes imposes significant limitations when dealing with other types of faults. LogKG [16] utilizes KGs to capture relationships between logs by integrating multi-field information through KG structures for fault diagnosis. Given that our method similarly employs KG-based techniques, LogKG serves as an appropriate baseline for comparison. LogRule [14] applies optimized association rule mining (ARM) on structured logs, using item aggregation and disjunctive support to extract frequent co-occurrence patterns. It generates interpretable rules for efficient RCA. This method aligns well with our objective, making it suitable for comparison. LogRCA [7] employs semi-supervised positive-unlabeled (PU) learning with Transformer-based embeddings to identify minimal sets of root-cause lines from imbalanced datasets. It ranks log lines by fault relevance, enabling tunable recall through top-n candidate selection. However, since it outputs a ranked set of potentially relevant log lines instead of an explicit root cause label or explanation, it is not directly applicable to our case-level RCA task.

LLM-Based Log Analysis has recently emerged as a promising direction for log-related tasks. LLMParser [23] proposes a few-shot log parsing framework leveraging GPT-style LLMs via in-context learning (ICL) [20]. It formulates the task as a text-to-structure generation problem. Despite achieving high parsing accuracy, it does not address RCA, and thus falls beyond the scope of our study. UniLog [19] presents an end-to-end automatic logging framework based on ICL with LLMs. It predicts logging positions, verbosity levels, and log messages in source code without fine-tuning. It focuses on log instrumentation and thus falls outside the scope of our RCA-oriented study. SelfLog [27] proposes a self-evolving log parsing system based on a self-supervised, group-wise learning framework. It employs N-gram [30] tokenization and pattern matching to group semantically similar log lines, enabling template extraction without manual annotations. While effective for adaptive parsing, it is not designed for RCA and thus unsuitable as a baseline. LogPrompt [25] advances LLM-based log parsing and anomaly detection by introducing three specialized prompting strategies—Self-prompt, Chain-of-Thought (CoT) [31], and ICL—to facilitate zero-shot and interpretable reasoning. Similarly, LogGPT [28] employs task-specific prompts enhanced with human knowledge for anomaly detection, and adopts structured JSON outputs to support practical few-shot learning. While both approaches leverage prompt engineering, LogPrompt explicitly integrates CoT reasoning to improve step-wise inference. Given its superior interpretability, we adopt LogPrompt as our representative

[1]https://github.com/ISSRE25-Submission-56/AetherLog

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

LLM-based baseline. LogRAG [29] introduces a retrieval-augmented generation (RAG) framework that enhances LLM-based log anomaly detection. It retrieves semantically similar historical log templates as external context and feeds them into an LLM to refine anomaly detection decisions. This form of external knowledge augmentation is conceptually aligned with our KG-based entity injection strategy, making LogRAG a suitable and comparable baseline.

Based on this analysis, we select the following methods as baselines in our evaluation: **LogKG** [16], **LogRule** [14], **LogPrompt** [25], and **LogRAG** [29].

III. Design Of AetherLog

As shown in Figure 3, AetherLog consists of two main pipelines: an offline pipeline that leverages LLMs to construct and refine a domain-specific KG, and an online pipeline that utilizes LLMs to retrieve and reason over the KG for accurate RCA. In this section, we provide a detailed explanation of each component with technical depth.

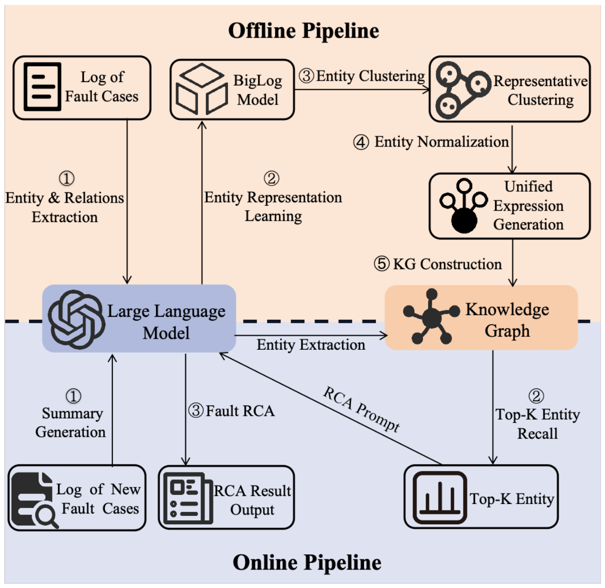


Fig. 3: The framework of AetherLog

A. *Offline Pipeline: Semantic Entity Aggregation Mechanism*

To build a reliable and semantically coherent KG from raw log data, our offline pipeline consists of five key steps: entity and relation extraction via LLM, entity representation learning, entity clustering, entity normalization, and redundancy-free KG construction. Each step is designed to progressively refine the extracted knowledge, ensuring both accuracy and compactness.

*1) **Step 1: Entity and Relation Extraction via LLM**:* In this step, we extract fault-relevant entities and relations from raw logs to improve the precision and relevance of the knowledge used for subsequent KG construction.

We introduce a label-aware prompting technique, where root cause labels function as weak supervision during the LLM prompting process. For each root cause, we empirically set the number of examples to $case\ number = 3$. A detailed analysis of different values of $case\ number$ and their impact on performance can be found in Sec. IV-E5. We adopt three distinct prompting strategies to enhance the robustness of entity and relation extraction, as illustrated in Figure 4.

a) Zero-shot Prompting: In zero-shot prompting, the LLM is directly instructed to extract entities and their semantic relations based on the given log case and root cause label, without any example guidance. This method relies on the LLM's prior knowledge and generalization capability.

Zero-shot Prompting

```
Instruction: The following log case is
     associated with the root cause "[
     RootCauseLabel]".
Extract key entities and their semantic
     relations, including components,
     operations, states, and critical
     parameters.

Log: [Log content here]
Output:
Entities: [Entity1, Entity2, ...]
Relations: [(EntityA, Relation, EntityB),
      ...]
```

Few-shot Prompting

```
Instruction: The following are annotated
     examples for extracting entities and
     semantic relations based on root
     cause labels.

Example 1:
Root Cause: "Database Connection fault"
Log: "[17:22:03] Database connection
     failed due to timeout. Application
     retries the connection after 5
     seconds."
Entities: [Database, connection, timeout,
      application, retries]
Relations: [(Database, causes, connection
      fault), (connection, blocked by,
     timeout), (application, retries,
     connection)]

Example 2:
...

Now extract from the following:
Root Cause: "[NewRootCauseLabel]"
Log: [New log here]
Output:
Entities: [...]
Relations: [...]
```

CoT Prompting

```
Instruction: The following log case is
     associated with the root cause "[
     RootCauseLabel]".
First, reason through the fault step-by-
     step to identify cause-effect
     relationships.
Then extract relevant entities and their
     semantic relations accordingly.

Log: [Log content here]
Output:
Entities: [Entity1, Entity2, ...]
Relations: [(EntityA, Relation, EntityB),
      ...]
Reasoning: [Step-by-step explanation]
```

Fig. 4: Illustration of different prompting strategies.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

b) Few-shot Prompting: Few-shot prompting augments the input with a small number of annotated log–label pairs, enabling the LLM to learn mapping patterns from raw logs to structured triples. In our approach, for each root cause, we provide a single representative fault case as the few-shot example. This targeted strategy helps the model focus on the most relevant components, operations, and conditions associated with the root cause.

c) CoT Prompting: CoT prompting encourages the LLM to perform step-by-step reasoning before producing structured outputs. In our setting, each prompt explicitly instructs the model to first analyze the causal logic within the log case and then extract relevant entities and relations accordingly. This approach improves both the interpretability of the extraction process and the accuracy of the resulting structured knowledge.

*2) **Step 2: Entity Representation Learning**:* In this step, we embed the extracted entities into a semantically meaningful vector space to facilitate downstream clustering and matching tasks.

Each extracted entity e_i is encoded into a dense semantic embedding v_i using a pre-trained model tailored for log data. While general-purpose language models like BERT [32] excel in feature extraction for various text analysis tasks, they may not fully capture the domain-specific nuances inherent in log data. Several studies have leveraged large-scale log data to pre-train language models [10], [33]–[37]. Here, we utilize BigLog [36], which has been specifically fine-tuned on large-scale log datasets, to capture more accurate semantic representations for log-related entities. Formally, the representation is computed as $v_i = \text{BigLog}(e_i)$, producing high-dimensional vectors that reflect both linguistic structure and log-specific characteristics. This enhances the effectiveness of subsequent RCA tasks.

*3) **Step 3: Entity Clustering**:* In this step, we perform clustering on semantically similar entities to reduce redundancy and improve the quality of the KG.

Although KMeans [38] is a widely used clustering algorithm, it assumes spherical cluster shapes and requires the number of clusters to be specified in advance. These assumptions do not hold in our task. In our setting, log-derived entity embeddings often exhibit irregular distributions, and the number of semantic groups is unknown beforehand. Therefore, we adopt DBSCAN [39], a density-based clustering method better suited to the characteristics of log data. DBSCAN does not require the number of clusters to be predefined and can identify clusters of arbitrary shapes. It relies on two key parameters: ε (neighborhood radius) and MinPts (minimum number of points required to form a dense region).

- **MinPts**: To determine the optimal value of MinPts, we conducted a series of experiments over a range of values (from 2 to 10), with ε held constant. We evaluated each configuration using metrics such as intra-cluster density, noise ratio, and silhouette score. Experimental results showed that setting MinPts $= 3$ yielded the best trade-off between identifying meaningful clusters and suppressing noise. This ensures that only sufficiently dense regions are considered valid clusters, while maintaining robustness to outliers.
- ε: The neighborhood radius ε was determined using the k-distance graph method. We computed the distance from each point to its MinPts-th nearest neighbor, sorted these distances in ascending order, and identified the "elbow" point. Based on this analysis, we selected $\varepsilon = 0.5$, which effectively captures dense semantic regions while minimizing the impact of noise.

This density-based clustering strategy ensures accurate grouping of fault-related entities, providing a solid foundation for subsequent normalization and KG construction.

*4) **Step 4: Entity Normalization**:* In this step, we canonicalize clustered entities to eliminate semantic redundancy and ensure consistent representations within the KG.

Given the clusters $\{c_1, c_2, \ldots, c_m\}$ generated in the previous step, we select a representative canonical entity $\hat{e}_j$ for each cluster c_j. Specifically, we adopt a centroid-based selection strategy, where the entity closest to the geometric center of the cluster in the embedding space is chosen:

$$\hat{e}_j = \arg\min_{e \in c_j} \|\mathbf{v}_e - \mu_j\|_2 , \quad \mu_j = \frac{1}{|c_j|} \sum_{e \in c_j} \mathbf{v}_e$$

This approach ensures that the selected entity is semantically central, preserving the representative meaning of each cluster while eliminating duplication and surface-level variation. After normalization, all entity mentions within the same cluster are mapped to their canonical form. This not only reduces node-level redundancy but also improves log semantic alignment and interpretability.

*5) **Step 5: Redundancy-Free KG Construction**:* In this step, we construct a compact and semantically aligned KG by integrating normalized entities and filtered semantic relations. This graph serves as a foundational structure for downstream RCA tasks.

Each canonical entity $\hat{e}_j$ corresponds to a node in the KG, and directed edges represent semantic relations (e.g., `causes`, `involves`, `leads_to`) extracted from logs using LLMs. To ensure semantic fidelity and structural clarity, we remap each original relation triple (e_i, r, e_j) to its normalized form $(\hat{e}_i, r, \hat{e}_j)$, where $\hat{e}_i$ and $\hat{e}_j$ denote the canonical forms of the original entities e_i and e_j, respectively. Since multiple entities may be mapped to the same canonical node, this projection process can lead to duplicate or conflicting edges. To address this, we consolidate candidate edges by selecting the most frequently occurring relation type among duplicates. In case of ties, we retain the first-observed relation. For example, consider two relations extracted from logs: `(Serving Gateway, causes, Timeout)` appears three times, while `(SGW, triggers, Timeout)` appears once. After normalization, both `Serving Gateway` and `SGW` are mapped to the canonical entity `SGW`. Since `causes` is the most frequent relation, the final consolidated edge becomes `(SGW, causes, Timeout)`. This scenario illustrates a common case where

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

synonymous entities and semantically overlapping relations are unified to preserve semantic coherence and reduce ambiguity in the KG structure.

To support efficient storage and querying, the constructed KG is persisted in a graph database (Neo4j [40]), which enables Cypher-based retrieval and allows for attaching rich attributes to both nodes and edges. This storage backend not only facilitates structured reasoning in downstream RCA tasks but also provides a flexible interface for exploring the semantic relations between fault-relevant entities.

B. *Online Pipeline: Context-aware Entity Recall Mechanism*

To support accurate and context-aware RCA, our online pipeline consists of three key steps: log summary generation, entity extraction and Top-K entity recall from KG, and RCA via LLM prompting. Each step is designed to efficiently process new fault cases by retrieving relevant context from the KG and guiding the LLM to infer causes or expand the graph if necessary.

1) ***Step 1: Log Summary Generation****:* In this step, we compress noisy and verbose fault cases into concise summaries that preserve causal signals for subsequent semantic matching.

In the online phase, the absence of label information makes it challenging to directly identify key fault-related content. To address this issue, we leverage the language understanding capabilities of LLMs to condense and interpret raw log sequences into concise and informative summaries suitable for downstream entity recall and RCA. We denote the semantic summary as $S_i = f_{\text{LLM}}(\text{Logs}_{F_i})$, where F_i is a given fault case and f_{LLM} is a summary function powered by GPT-4 [42].

Figure 5 illustrates the zero-shot prompt used for summary generation. The LLM is instructed to extract root cause indicators and relevant fault conditions from the log sequence, while omitting irrelevant implementation details. In contrast to the entity extraction stage, which involves multiple prompt engineering strategies (see Figure 4), this summarization step adopts a fixed zero-shot prompt. The objective here is not to perform reasoning or classification, but to produce a concise and structured abstraction of verbose logs that can serve as input for entity extraction and recall.

```
Instruction: The following log contains a
      sequence of events related to a
     fault. Please summarize the key fault
      information, focusing on the root
     cause and any relevant fault
     conditions, without including
     unnecessary details.
Log: [Log content here]
Output:
Summary: [Concise description of the
     fault cause and related events.]
```

Fig. 5: Log Summary Generation Prompt

2) ***Step 2: Entity Extraction and Top-K Entity Recall from KG****:* In this step, we extract semantic entities from the fault summary and match them to relevant nodes in the KG via embedding similarity.

After generating the semantic summary S_i in Step 1, we perform entity and relation extraction using the same LLM-based prompting strategies as employed in the offline stage. Let the extracted entity set from S_i be $\mathcal{E}_{S_i} = \{e_1, e_2, \dots\}$. Each entity $e_k \in \mathcal{E}_{S_i}$ is encoded into a vector representation $\mathbf{v}_{e_k}$ using the same embedding model (BigLog).

To retrieve the most relevant entities from the KG, we compute the cosine similarity between each extracted entity vector $\mathbf{v}_{e_k}$ and all KG entity vectors $\mathbf{v}_j$. The final recalled entity set $\mathcal{E}_{\text{top}}$ consists of the top-K most similar KG entities across all comparisons: $\mathcal{E}_{\text{top}} = \text{Top-K}_{e_j \in \text{KG}} \left(\max_{e_k \in \mathcal{E}_{S_i}} \frac{\mathbf{v}_{e_k} \cdot \mathbf{v}_j}{\|\mathbf{v}_{e_k}\| \cdot \|\mathbf{v}_j\|} \right)$, where the similarity is defined as cosine similarity. Through empirical validation, we found that $K = 3$ yields the best recall performance. A detailed analysis of different values of K and their impact on performance is provided in Section IV-E5.

3) ***Step 3: RCA via LLM Prompting****:* In this step, we construct a structured prompt that integrates the fault summary and the retrieved entities to guide the LLM in performing RCA and suggesting knowledge expansion.

Once the top-3 relevant entities are identified, we formulate a prompt to query the LLM for RCA. The structure of the RCA prompt is illustrated in Figure 6. It consists of two key components: (1) the fault summary, which provides a concise description of the current log case, and (2) the related entities, which include the top-3 matched entities along with their corresponding relations and cosine similarity scores from the KG. The LLM is instructed to utilize these similarity scores in its reasoning process: if one or more entities have high similarity scores, the LLM should consider root causes previously associated with them. If the similarity scores are low, the LLM is prompted to infer a novel root cause by reasoning over the fault context and inter-entity relationships.

```
Instruction: Based on the following fault
      case and associated entities,
     identify the root cause.
Fault Summary: [summary of new log]
Related Entities: [Top-3 entities]
Similarity Scores: [Cosine similarity
     scores of the top-3 entities]
Guidance:
- If any entity has a high similarity
     score, consider selecting its
     associated root cause.
- If all scores are relatively low,
     reason based on inter-entity
     relationships and fault context to
     infer a new root cause.
- If a novel root cause is inferred, also
      suggest a new entity and its
     potential relationship to the
     existing KG.
- Then, extract key entities and
     relationships in the form of (Head
     Entity, Relation, Tail Entity) based
     on the inferred root cause and fault
     context.
Output:
Root Cause: [Inferred or selected root
     cause]
Knowledge Triples:
(Head Entity 1, Relation 1, Tail Entity
     1)
(Head Entity 2, Relation 2, Tail Entity
     2)
...
```

Fig. 6: Prompt Construction for RCA

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

This approach enables the model to generalize beyond previously seen patterns and facilitates dynamic KG expansion by suggesting new entities and causal relationships. Specifically, when a novel root cause is inferred, the LLM is further prompted to extract structured triples in the form of `(Head Entity, Relation, Tail Entity)`, based on both the inferred cause and contextual log semantics. These newly generated triples are validated and incorporated into the KG, allowing the knowledge base to evolve continuously.

IV. Experiments

In this section, we evaluate AetherLog by addressing the following research questions (RQs):

- **RQ1:** How does AetherLog perform in log-based fault RCA compared to state-of-the-art methods?
- **RQ2:** How do different LLMs affect AetherLog's performance on the RCA task?
- **RQ3:** How do different prompting strategies influence AetherLog's performance on the RCA task?
- **RQ4:** How does each component contribute to the overall RCA performance of AetherLog?
- **RQ5:** How do key hyperparameters impact AetherLog's performance on the RCA task?

A. Datasets

We conduct experiments on two real-world datasets:

- **Dataset 1**[2]: Sourced from Alibaba Cloud's Tianchi platform, which collects data from servers deployed in production environments. The dataset comprises 2,671 fault cases recorded over several weeks. Each fault case is mapped to one of the following three root causes: *Processor Internal fault*, *Memory Access Violation*, and *Peripheral Hardware Fault*.
- **Dataset 2**[3]: Sourced from an OpenStack-based [41] system deployed by China Mobile, which collects data from a large-scale 4G/5G core network infrastructure. The dataset comprises 93 fault cases recorded over 24 days. Each fault case is mapped to one of the following six root causes: *Message Queue Unavailability*, *Database Connection fault*, *Compute Node Outage*, *Resource Configuration Conflict*, *Linuxbridge Agent fault*, and *Controller Service Disconnection*.

B. Baseline Methods

As detailed in Table I, we compare AetherLog against several representative baseline methods. Although some of them were not originally designed for RCA tasks, we adapt them accordingly to ensure a fair and meaningful comparison. Below, we describe each baseline and explain how it is modified or integrated into our experiments.

- **LogKG** [16]: A semi-supervised KG-based method designed for log-based fault diagnosis. In our setup, we reuse the KG constructed by AetherLog to ensure consistency in both knowledge content and structure.
- **LogRule** [14]: An ARM method that generates interpretable rules from structured logs for automated RCA. We convert the offline KG into a structured log format and use it as input to LogRule's rule generation pipeline, enabling both methods to operate on the same underlying knowledge base.
- **LogPrompt** [25]: A prompt-based zero-shot anomaly analysis method that leverages ICL and CoT prompting. We adapt LogPrompt for RCA by modifying its prompts to focus on root cause identification and by injecting top-K retrieved KG entities and relations to enrich contextual understanding.
- **LogRAG** [29]: A semi-supervised RAG-based framework that improves LLM-based log analysis by retrieving relevant historical logs as external context. To adapt it for RCA, we replace its DeepSVDD module with a multi-class classifier trained on LLM-generated log summaries. We also substitute its retrieval source with our KG and feed the top-K retrieved entities into a CoT-style prompt, enabling the LLM to refine the root cause prediction.

For each baseline, we report the best-performing configuration across multiple runs. All methods are evaluated on the same knowledge base to ensure a fair and consistent comparison with AetherLog.

C. Evaluation Metrics

Given that each fault case in our dataset is annotated with a definitive root cause from multiple possible classes, we formulate the RCA task as a multi-class classification problem. Accordingly, we adopt three widely used metrics—**Precision**, **Recall**, and **F1-Score (macro-averaged)**—to evaluate predictive performance. Additionally, we report **Inference Time** to assess model efficiency and its suitability for online RCA scenarios.

Precision (P) measures the proportion of correctly predicted root causes among all predicted instances for each class, calculated as Precision $= \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i+FP_i}$, where C is the number of root cause classes, and TP_i and FP_i denote the true positives and false positives for class i.

Recall (R) measures the proportion of actual root cause instances that are correctly identified by the model for each class, defined as Recall $= \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i+FN_i}$, where FN_i is the number of false negatives for class i.

F1-Score (F) is the harmonic mean of precision and recall, computed per class and then macro-averaged across all classes: F1-Score $= \frac{1}{C}\sum_{i=1}^{C} 2 \times \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$.

Inference Time (T) measures the average time (in seconds) required by the model to complete RCA for a single fault case, reflecting the model's efficiency and applicability to online RCA scenarios.

D. Implementation Details and Environment

AetherLog was implemented in Python 3.8 using the PyTorch framework for KG construction and related inference tasks. All experiments were conducted on a server equipped with 16 GB of RAM and 10 NVIDIA A6000 GPUs. For

[2]https://tianchi.aliyun.com/competition/entrance/531947/information
[3]https://github.com/SycIsDD/LogKG

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

the LLM component, we utilized a diverse set of pre-trained models, including GPT-4 [42], DeepSeek-V3 [43], LLaMA 3 [44], and Mistral 7B [45]. GPT-4 and DeepSeek-V3 models were accessed via API calls (GPT-4 through OpenAI's API and DeepSeek-V3 through the DeepSeek interface, while LLaMA 3 and Mistral 7B were run locally using model weights downloaded from Hugging Face[4][5].

E. Results and Analysis

1) RQ1: How does AetherLog perform in log-based fault RCA compared to state-of-the-art methods?: To evaluate the effectiveness of our proposed method, we compare **AetherLog** against four representative RCA baselines, as shown in Table II. Among them, **LogRule** and **LogKG** are SLM-based methods, while **LogPrompt** and **LogRAG** are LLM-based methods.

TABLE II: Performance Comparison of Baseline Methods

Method	**Dataset 1**				**Dataset 2**			
	P	**R**	**F**	**T**	**P**	**R**	**F**	**T**
LogRule	0.33	0.30	0.31	0.02	0.26	0.22	0.23	0.02
LogKG	0.50	0.47	0.48	2.18	0.52	0.59	0.50	2.20
LogPrompt	0.84	0.88	0.85	3.64	0.77	0.78	0.77	3.35
LogRAG	0.89	0.86	0.87	4.42	0.90	0.89	0.89	4.45
AetherLog	**0.94**	**0.93**	**0.93**	4.91	**0.97**	**0.96**	**0.97**	4.83

Comparison with SLM-based methods. LogRule exhibits the lowest overall performance, with precision scores of 0.33 and 0.26, recall scores of 0.30 and 0.22, and F1-scores of 0.31 and 0.23 on Dataset 1 and Dataset 2, respectively. This underperformance is primarily due to its reliance on manually defined, static rules. While LogRule demonstrates high efficiency with an inference time of only 0.02s per case, the substantial trade-off in accuracy limits its practical applicability. LogKG achieves precision scores of 0.50 and 0.52, recall scores of 0.47 and 0.59, and F1-scores of 0.48 and 0.50, with inference times of 2.18s and 2.20s. The performance improvement is attributed to its use of structured knowledge representations.

Comparison with LLM-based methods. LogPrompt achieves precision scores of 0.84 and 0.77, recall scores of 0.88 and 0.78, and F1-scores of 0.85 and 0.77 on the two datasets, with inference times of 3.64s and 3.35s, respectively. These results highlight the LLM's capability for semantic reasoning and generalization. LogRAG further improves performance, achieving precision scores of 0.89 and 0.90, recall scores of 0.86 and 0.89, and F1-scores of 0.87 and 0.89. However, these gains are accompanied by longer inference times—4.42s and 4.45s—due to the additional retrieval stage.

AetherLog consistently outperforms all baseline methods. It achieves the highest precision scores of 0.94 and 0.97, recall scores of 0.93 and 0.96, and F1-scores of 0.93 and 0.97 on the two datasets. Compared to LogRAG—the strongest baseline—AetherLog improves the F1-score by 6% on Dataset 1 and 8% on Dataset 2, while incurring only a modest increase in inference time (4.91s vs. 4.42s on Dataset 1, and 4.83s vs. 4.45s on Dataset 2).

[4]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[5]https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1

2) RQ2: How do different LLMs affect AetherLog's performance on the RCA task?: To evaluate the effect of LLM capacity on RCA performance, we compare AetherLog using four representative models: **GPT-4**, **DeepSeek-V3**, **LLaMA 3**, and **Mistral 7B**. As shown in Figure 7, we report precision, recall, F1-score, and inference time on both datasets. These models are categorized into two groups based on parameter size: **large-scale models** (GPT-4 and DeepSeek-V3, with over 100B parameters) and **lightweight models** (LLaMA 3 and Mistral 7B, with approximately 7–8B parameters).

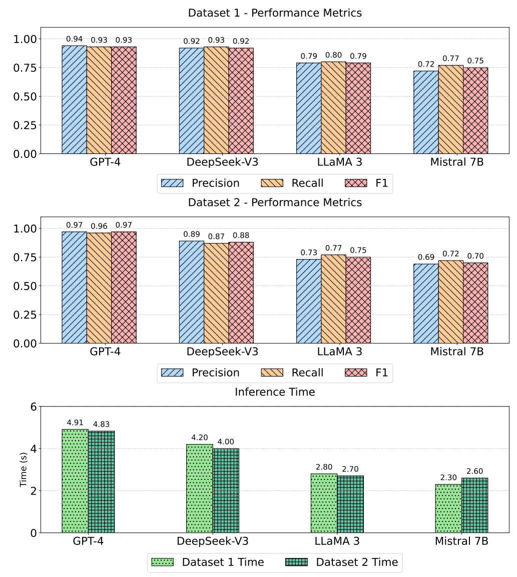


Fig. 7: Performance comparison of different LLMs.

Large-scale models. GPT-4 achieves precision scores of 0.94 and 0.97, recall scores of 0.93 and 0.96, and F1-scores of 0.93 and 0.97 on the two datasets, with inference times of 4.91s and 4.83s, respectively. These results demonstrate GPT-4's strong reasoning ability and superior performance. DeepSeek-V3 achieves precision scores of 0.92 and 0.89, recall scores of 0.93 and 0.87, and F1-scores of 0.92 and 0.88, with shorter inference times of 4.2s and 4s. While it slightly trails GPT-4 in accuracy, its reduced latency makes it a viable alternative in time-sensitive deployments.

Lightweight models. LLaMA 3 achieves precision scores of 0.79 and 0.73, recall scores of 0.80 and 0.77, and F1-scores of 0.79 and 0.75, with inference times of 2.8s and 2.7s. These results indicate that LLaMA 3 offers a favorable trade-off between accuracy and efficiency, making it suitable for deployment in resource-constrained environments. Mistral 7B achieves precision scores of 0.72 and 0.69, recall scores of 0.77 and 0.72, and F1-scores of 0.75 and 0.70, with the shortest inference times of 2.3s and 2.6s. However, its overall lower performance suggests that lightweight models with smaller parameter counts may struggle to capture the

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

complex semantics required for accurate RCA compared to large-scale models.

These results confirm AetherLog's flexibility in integrating LLMs of varying capacity based on task requirements. Given its consistent superiority across all metrics, GPT-4 is chosen as the default backbone in the following experiments.

3) RQ3: How do different prompting strategies influence AetherLog's performance on the RCA task?: To investigate the impact of prompting strategies on AetherLog's performance, we evaluate three commonly used techniques: **Zero-shot**, **Few-shot**, and **CoT** prompting. Table III summarizes the results in terms of precision, recall, F1-score, and inference time across both datasets.

TABLE III: Effect of Different Prompt Strategies

Prompt Type	Dataset 1				Dataset 2			
	P	R	F	T	P	R	F	T
Zero-shot Prompting	0.79	0.81	0.79	2.74	0.76	0.75	0.75	2.69
Few-shot Prompting	0.86	0.84	0.85	3.28	0.82	0.83	0.82	3.17
CoT Prompting	**0.94**	**0.93**	**0.93**	4.91	**0.97**	**0.96**	**0.97**	4.83

Zero-shot Prompting achieves precision scores of 0.79 and 0.76, recall scores of 0.81 and 0.75, and F1-scores of 0.79 and 0.75 on the two datasets, with inference times of 2.74s and 2.69s, respectively. These results suggest that, while zero-shot prompting enables basic fault extraction without additional examples, its lack of contextual guidance limits its ability to capture complex or domain-specific patterns. The low latency, however, makes it efficient for quick preliminary analysis.

Few-shot Prompting improves performance across all metrics, achieving precision scores of 0.86 and 0.82, recall scores of 0.84 and 0.83, and F1-scores of 0.85 and 0.82, with inference times of 3.28s and 3.17s. This demonstrates the benefit of example-driven ICL, which helps the model better generalize across diverse fault cases, though at a moderate computational cost.

CoT Prompting yields the best overall performance, with precision scores of 0.94 and 0.97, recall scores of 0.93 and 0.96, and F1-scores of 0.93 and 0.97. These gains come at the cost of increased inference times—4.91s and 4.83s—due to the additional reasoning steps. Nevertheless, the improved accuracy clearly demonstrates the advantage of CoT prompting in enabling deeper semantic understanding of complex fault scenarios.

In summary, CoT prompting significantly outperforms other prompting strategies in terms of accuracy, achieving the highest precision, recall, and F1-scores across both datasets. While it introduces longer inference time, the overhead remains within an acceptable range. This is because RCA is typically conducted as an offline or near-real-time diagnostic task, where accuracy and interpretability are prioritized over strict latency constraints. Therefore, we adopt CoT prompting as the default strategy in AetherLog.

4) RQ4: How does each component contribute to the overall RCA performance of AetherLog?: To evaluate the importance of individual components in AetherLog, we conduct an ablation study by removing three key modules: (1) LLM-based entity extraction, (2) embedding-based entity alignment, and (3) LLM-based log summarization. For each variant, the removed module is replaced with a simpler alternative. We then evaluate model performance in terms of precision, recall, F1-score, and online inference time. The results are shown in Figure 8.

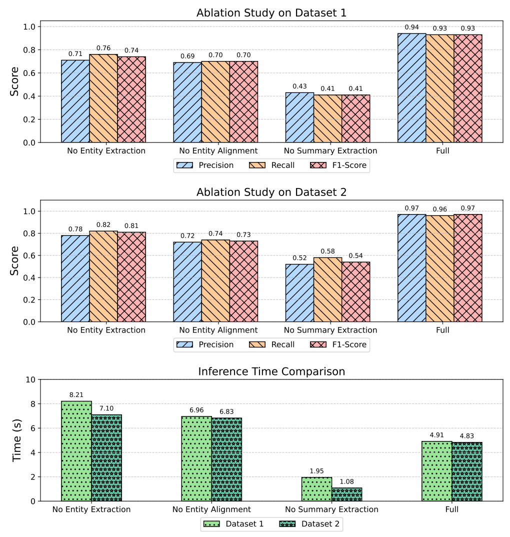


Fig. 8: Ablation Study of AetherLog's Components

AetherLog (Full) achieves the best overall performance, with precision scores of 0.94 and 0.97, recall scores of 0.93 and 0.96, and F1-scores of 0.93 and 0.97 on Dataset 1 and Dataset 2, respectively. The average online inference time per case is 4.91s and 4.83s, representing the cost of summary generation, semantic matching, and RCA via LLM prompting.

No Entity Extraction. This variant replaces the LLM-based entity extraction step with a rule-based pipeline combining Stanford CoreNLP [46] and OpenIE [47]. Although this reduces reliance on LLMs, it causes performance to degrade to precision scores of 0.71 and 0.78, recall scores of 0.76 and 0.82, and F1-scores of 0.74 and 0.81. Notably, the online inference time increases significantly to 8.21s and 7.10s. This is because rule-based extraction introduces many noisy or redundant entities into the KG, enlarging the search space and slowing down semantic matching.

No Entity Alignment. In this variant, the alignment step that merges semantically similar entities via clustering is removed, and all extracted entities are preserved in their raw form. This leads to a moderate drop in performance, with precision scores of 0.69 and 0.72, recall scores of 0.70 and 0.74, and F1-scores of 0.70 and 0.73. Online inference time slightly increases to 6.96s and 6.83s, as the unaligned KG becomes more redundant, resulting in longer entity recall computations. These results highlight that entity alignment plays a crucial role in enhancing both efficiency and reasoning accuracy by reducing duplication in the KG.

No Summary Extraction. This variant removes the LLM-based summarization module from the online pipeline. Instead, the full raw log is fed directly into the entity extraction step.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

This results in the most severe performance degradation, with F1-scores dropping to 0.41 and 0.54 on the two datasets, and precision scores of 0.43 and 0.52 and recall scores of 0.41 and 0.58. Interestingly, this variant exhibits the lowest inference times—1.95s and 1.08s—since the summarization step is omitted. However, bypassing summarization severely reduces the quality of extracted entities and introduces noise, thus undermining the RCA accuracy.

In summary, each component—LLM-based summarization, entity extraction, and entity alignment—plays a vital role in ensuring accurate and efficient RCA. Removing entity extraction or alignment leads to increased online latency due to KG redundancy, while removing summarization reduces latency at the cost of drastically lower accuracy. These findings strongly support the modular design of AetherLog and highlight the complementary contributions of each component.

5) RQ5: How do key hyperparameters impact AetherLog's performance on the RCA task?: To understand how different design choices impact AetherLog's effectiveness, we conduct studies over two critical hyperparameters: (1) the number of representative fault cases used to construct the KG (**Case Number**), and (2) the number of candidate entities retrieved during the semantic matching phase (**Top-K**). Table IV summarizes the results in terms of precision, recall, and F1-score across both datasets.

TABLE IV: Performance of Different Hyperparameter Settings

Case No.	K	Dataset 1			Dataset 2		
		P	R	F	P	R	F
1	1	0.50	0.55	0.52	0.58	0.53	0.55
	3	0.72	0.70	0.71	0.71	0.69	0.70
	5	0.71	0.69	0.70	0.70	0.68	0.69
3	1	0.69	0.66	0.67	0.65	0.63	0.64
	3	**0.94**	**0.93**	**0.93**	**0.97**	**0.96**	**0.97**
	5	0.95	0.93	0.93	0.95	0.96	0.96
5	1	0.61	0.68	0.65	0.68	0.66	0.67
	3	0.88	0.87	0.87	0.87	0.86	0.86
	5	0.87	0.86	0.86	0.86	0.85	0.85
7	1	0.63	0.60	0.61	0.60	0.68	0.64
	3	0.84	0.83	0.83	0.87	0.86	0.87
	5	0.83	0.82	0.82	0.86	0.85	0.85

Impact of Case Number. When fixing the Top-K value to $K = 3$, using 3 representative fault cases per root cause consistently yields the best performance. AetherLog achieves precision scores of 0.94 and 0.97, recall scores of 0.93 and 0.96, and F1-scores of 0.93 and 0.97 on Dataset 1 and Dataset 2, respectively. Increasing the number of cases to 5 or 7 does not result in further improvements; for example, with 5 cases, F1-scores drop slightly to 0.87 and 0.86, and with 7 cases, to 0.83 and 0.87. In contrast, using only 1 case per root cause leads to significantly lower performance, with F1-scores of 0.71 and 0.70. These results indicate that using 3 well-chosen examples per class strikes a good balance between accuracy and annotation cost, while larger case sets may introduce redundancy or noise that limits gains.

Impact of Top-K. When fixing the number of cases per root cause to 3, the Top-K setting has a clear impact on performance. AetherLog achieves its best results with $K = 3$, reaching F1-scores of 0.93 and 0.97 on the two datasets. Using $K = 1$ significantly reduces performance, with F1-scores of 0.67 and 0.64, due to insufficient candidate diversity for reasoning. On the other hand, increasing to $K = 5$ yields no additional benefits, with F1-scores of 0.93 and 0.96—comparable to $K = 3$. These results suggest that Top-3 retrieval provides the optimal trade-off between recall sufficiency and entity relevance in RCA prompting.

Based on the observed results, we select three representative fault cases per root cause and set the entity recall number to Top-3 in all experiments.

V. Case Study

In this case study, we analyze how AetherLog improves RCA by addressing two fundamental challenges in log-based reasoning:

- **Type 1: Semantic Entity Aggregation.** Logs may describe the same fault using inconsistent terminology. AetherLog applies LLM-based entity extraction followed by clustering-based alignment to unify these expressions, thereby improving the compactness and semantic consistency of the KG.
- **Type 2: Context-Aware Entity Recall.** Fault cases often contain verbose, interleaved logs from multiple components. AetherLog generates high-level summaries to abstract causal information, then performs entity recall based on the summary, enhancing both precision and coverage.

A. Type 1: Redundant Entity Expressions

To illustrate the benefits of semantic entity aggregation, we present two representative examples. As shown in Figure 9, in Dataset 1, multiple logs describe thermal-related processor faults using different terms: *"Processor error"*, *"CPU fault"*, *"Power cycle"*, and *"System shutdown"*. AetherLog clusters these into unified entities such as *Processor error* and *Thermal shutdown*, enhancing semantic consistency. In Dataset 2, messaging-related failures—e.g., *"Message router"*, *"Messaging dispatcher"*, and *"Message bus"*—are consolidated into a single representative entity: *Message bus*. This alignment reduces redundancy and improves downstream reasoning. In both cases, the number of distinct entities is reduced from 6–8 to 3. Table V further summarizes the global impact of alignment, showing a reduction in KG entity count by 54.32% in Dataset 1 and 57.38% in Dataset 2.

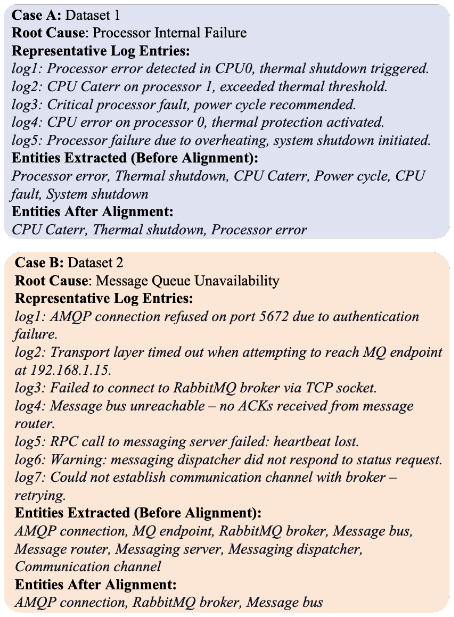
TABLE V: Entity Alignment Statistics

Dataset	Original	Aligned	Reduction
Dataset 1	81	37	54.32%
Dataset 2	122	52	57.38%

B. Type 2: Context-Aware Entity Recall

To evaluate the effectiveness of AetherLog's context-aware entity recall mechanism, we present representative cases from both datasets. As illustrated in Figure 10, logs from Dataset 1

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.



Case A: Dataset 1
Root Cause: Processor Internal Failure
Representative Log Entries:
log1: Processor error detected in CPU0, thermal shutdown triggered.
log2: CPU Caterr on processor 1, exceeded thermal threshold.
log3: Critical processor fault, power cycle recommended.
log4: CPU error on processor 0, thermal protection activated.
log5: Processor failure due to overheating, system shutdown initiated.
Entities Extracted (Before Alignment):
Processor error, Thermal shutdown, CPU Caterr, Power cycle, CPU fault, System shutdown
Entities After Alignment:
CPU Caterr, Thermal shutdown, Processor error

Case B: Dataset 2
Root Cause: Message Queue Unavailability
Representative Log Entries:
log1: AMQP connection refused on port 5672 due to authentication failure.
log2: Transport layer timed out when attempting to reach MQ endpoint at 192.168.1.15.
log3: Failed to connect to RabbitMQ broker via TCP socket.
log4: Message bus unreachable – no ACKs received from message router.
log5: RPC call to messaging server failed: heartbeat lost.
log6: Warning: messaging dispatcher did not respond to status request.
log7: Could not establish communication channel with broker – retrying.
Entities Extracted (Before Alignment):
AMQP connection, MQ endpoint, RabbitMQ broker, Message bus, Message router, Messaging server, Messaging dispatcher, Communication channel
Entities After Alignment:
AMQP connection, RabbitMQ broker, Message bus

Fig. 9: Before and after entity alignment.

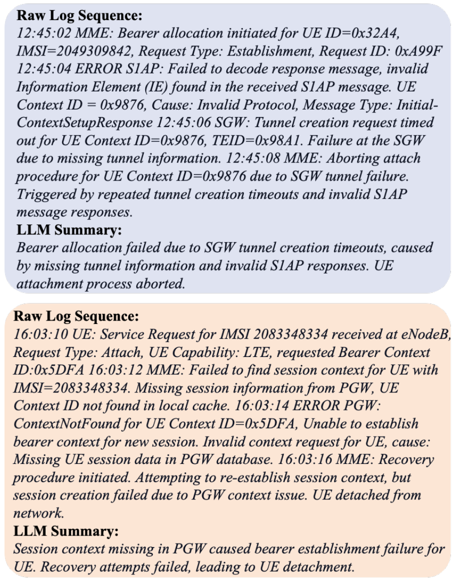
span components such as MME, SGW, and S1AP. Key fault indicators like *tunnel creation timeout* and *invalid S1AP message* are scattered and difficult to extract line by line. AetherLog summarizes the log sequence as: *"Bearer allocation failed due to SGW tunnel creation timeouts, caused by missing tunnel information and invalid S1AP responses. UE attachment process aborted."* From this summary, entities such as *SGW tunnel timeout* and *invalid S1AP response* are extracted—critical for accurate RCA but hard to obtain from raw logs alone. In Dataset 2, the logs involve complex session establishment failures linked to PGW context issues. The summary—*"Session context missing in PGW caused bearer establishment failure for UE. Recovery attempts failed, leading to UE detachment."*—enables accurate entity recall such as *PGW session missing* and *UE detachment*. Table VI quantifies the improvement: the number of recalled relevant entities increased from 21 to 32 in Dataset 1 (52.38% improvement) and from 32 to 47 in Dataset 2 (46.88% improvement).

TABLE VI: Entity Recall Statistics

Dataset	Before	After	Increase
Dataset 1	21	32	52.38%
Dataset 2	32	47	46.88%

VI. Conclusion and Future Work

This paper presents AetherLog, a novel framework for RCA that integrates the strengths of LLMs and KGs. AetherLog operates in two phases: offline, it introduces a semantic entity aggregation mechanism that constructs a clean and de-duplicated KG through LLM-based entity extraction, semantic clustering, and normalization; online, it introduces a context-aware entity recall mechanism that leverages LLM-generated summaries to retrieve relevant KG entities and construct informative prompts for RCA. Experiments on two real-world datasets demonstrate



Raw Log Sequence:
12:45:02 MME: Bearer allocation initiated for UE ID=0x32A4, IMSI=2049309842, Request Type: Establishment, Request ID: 0xA99F 12:45:04 ERROR S1AP: Failed to decode response message, invalid Information Element (IE) found in the received S1AP message. UE Context ID = 0x9876, Cause: Invalid Protocol, Message Type: Initial-ContextSetupResponse 12:45:06 SGW: Tunnel creation request timed out for UE Context ID=0x9876, TEID=0x98A1. Failure at the SGW due to missing tunnel information. 12:45:08 MME: Aborting attach procedure for UE Context ID=0x9876 due to SGW tunnel failure. Triggered by repeated tunnel creation timeouts and invalid S1AP message responses.
LLM Summary:
Bearer allocation failed due to SGW tunnel creation timeouts, caused by missing tunnel information and invalid S1AP responses. UE attachment process aborted.

Raw Log Sequence:
16:03:10 UE: Service Request for IMSI 2083348334 received at eNodeB, Request Type: Attach, UE Capability: LTE, requested Bearer Context ID:0x5DFA 16:03:12 MME: Failed to find session context for UE with IMSI=2083348334. Missing session information from PGW, UE Context ID not found in local cache. 16:03:14 ERROR PGW: ContextNotFound for UE Context ID=0x5DFA, Unable to establish bearer context for new session. Invalid context request for UE, cause: Missing UE session data in PGW database. 16:03:16 MME: Recovery procedure initiated. Attempting to re-establish session context, but session creation failed due to PGW context issue. UE detached from network.
LLM Summary:
Session context missing in PGW caused bearer establishment failure for UE. Recovery attempts failed, leading to UE detachment.

Fig. 10: Before and after LLM-based summary extraction.

that AetherLog consistently outperforms state-of-the-art baselines. Ablation studies confirm the complementary strengths of LLMs and KGs, while prompt engineering experiments reveal that CoT reasoning significantly boosts RCA performance.

For future work, we plan to evaluate the performance of different LLMs within the AetherLog framework and explore more effective strategies for entity recall and prompt construction to further improve RCA accuracy. In addition, we will investigate advanced reasoning techniques such as multi-hop KG traversal and incorporate temporal dynamics from log data to better capture the evolution of system faults over time and across contexts.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (62272249, 62302244), and the Fundamental Research Funds for the Central Universities (XXX-63253249).

References

[1] Yu B, Yao J, Fu Q, et al. Deep learning or classical machine learning? an empirical study on log-based anomaly detection[C]//Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. 2024: 1-13.
[2] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: Multi-modal Causal Structure Learning and Root Cause Analysis for Microservice Systems. In Proceedings of the ACM Web Conference 2024 (WWW '24). Association for Computing Machinery, New York, NY, USA, 4107–4116.
[3] Yu E, Dong H, Ren Y, et al. Hrca: A heterogeneous graph-based adaptive root cause analysis framework[C]//2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2023: 63-68.
[4] Zhang Y, Guan Z, Qian H, et al. CloudRCA: A root cause analysis framework for cloud computing platforms[C]//Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 2021: 4373-4382.
[5] Wang H, Wu Z, Jiang H, et al. Groot: An event-graph-based approach for root cause analysis in industrial settings[C]//2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2021: 419-429.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.

[6] Wu L, Tordsson J, Elmroth E, et al. Microrca: Root cause localization of performance issues in microservices[C]//NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2020: 1-9.

[7] Wittkopp T, Wiesner P, Kao O. LogRCA: Log-based Root Cause Analysis for Distributed Services[C]//European Conference on Parallel Processing. Cham: Springer Nature Switzerland, 2024: 362-376.

[8] Zawawy H, Kontogiannis K, Mylopoulos J. Log filtering and interpretation for root cause analysis[C]//2010 IEEE International Conference on Software Maintenance. IEEE, 2010: 1-5.

[9] He S, He P, Chen Z, et al. A survey on automated log analysis for reliability engineering[J]. ACM computing surveys (CSUR), 2021, 54(6): 1-37.

[10] Le V H, Zhang H. PreLog: A Pre-trained Model for Log Analytics[J]. Proceedings of the ACM on Management of Data, 2024, 2(3): 1-28.

[11] X. Li, P. Chen, L. Jing, Z. He and G. Yu, "SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults," 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 2020, pp. 92-103.

[12] Zhang X, Xu Y, Qin S, et al. Onion: identifying incident-indicating logs for cloud systems[C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021: 1253-1263.

[13] Lu S, Wei X, Rao B, et al. LADRA: Log-based abnormal task detection and root-cause analysis in big data processing with Spark[J]. Future Generation Computer Systems, 2019, 95: 392-403.

[14] Notaro P, Haeri S, Cardoso J, et al. Logrule: Efficient structured log mining for root cause analysis[J]. IEEE Transactions on Network and Service Management, 2023, 20(4): 4231-4243.

[15] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018). Association for Computing Machinery, New York, NY, USA, 60–70.

[16] Sui Y, Zhang Y, Sun J, et al. Logkg: Log failure diagnosis through knowledge graph[J]. IEEE Transactions on Services Computing, 2023, 16(5): 3493-3507.

[17] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[18] Shuang Song, Yifei Zhang, and Neng Gao. 2025. Confront Insider Threat: Precise Anomaly Detection in Behavior Logs Based on LLM Fine-Tuning. In Proceedings of the 31st International Conference on Computational Linguistics, pages 8589–8601, Abu Dhabi, UAE. Association for Computational Linguistics.

[19] Xu J, Cui Z, Zhao Y, et al. Unilog: Automatic logging via llm and in-context learning[C]//Proceedings of the 46th ieee/acm international conference on software engineering. 2024: 1-12.

[20] Dong Q, Li L, Dai D, et al. A survey on in-context learning[J]. arXiv preprint arXiv:2301.00234, 2022.

[21] Zhong A, Mo D, Liu G, et al. Logparser-llm: Advancing efficient log parsing with large language models[C]//Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2024: 4559-4570.

[22] He M, Jia T, Duan C, et al. LLMeLog: An Approach for Anomaly Detection based on LLM-enriched Log Events[C]//2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2024: 132-143.

[23] Ma Z, Chen A R, Kim D J, et al. Llmparser: An exploratory study on using large language models for log parsing[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024: 1-13.

[24] Xu J, Yang R, Huo Y, et al. Divlog: Log parsing with prompt enhanced in-context learning[C]//Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024: 1-12.

[25] Liu Y, Tao S, Meng W, et al. Logprompt: Prompt engineering towards zero-shot and interpretable log analysis[C]//Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. 2024: 364-365.

[26] Jiang Z, Liu J, Chen Z, et al. Lilac: Log parsing using llms with adaptive parsing cache[J]. Proceedings of the ACM on Software Engineering, 2024, 1(FSE): 137-160.

[27] Pei C, Liu Z, Li J, et al. Self-Evolutionary Group-wise Log Parsing Based on Large Language Model[C]//2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2024: 49-60.

[28] Qi J, Huang S, Luan Z, et al. Loggpt: Exploring chatgpt for log-based anomaly detection[C]//2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). IEEE, 2023: 273-280.

[29] Zhang W, Zhang Q, Yu E, et al. Leveraging RAG-Enhanced Large Language Model for Semi-Supervised Log Anomaly Detection[C]//2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2024: 168-179.

[30] Cavnar W B, Trenkle J M. N-gram-based text categorization[C]//Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. 1994, 161175: 14.

[31] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in neural information processing systems, 2022, 35: 24824-24837.

[32] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019). https://arxiv.org/abs/1810.04805.

[33] Chen Z, Gao Q, Moss L S. NeuralLog: Natural language inference with joint neural and logical reasoning[J]. arXiv preprint arXiv:2105.14167, 2021.

[34] Guo H, Yuan S, Wu X. Logbert: Log anomaly detection via bert[C]//2021 international joint conference on neural networks (IJCNN). IEEE, 2021: 1-8.

[35] Zhang M, Chen J, Liu J, et al. Logst: Log semi-supervised anomaly detection based on sentence-bert[C]//2022 7th International Conference on Signal and Image Processing (ICSIP). IEEE, 2022: 356-361.

[36] Tao S, Liu Y, Meng W, et al. Biglog: Unsupervised large-scale pre-training for a unified log representation[C]//2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS). IEEE, 2023: 1-11.

[37] Huo Y, Lee C, Su Y, et al. Evlog: Identifying anomalous logs over software evolution[C]//2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2023: 391-402.

[38] Steinley D. K-means clustering: a half-century synthesis[J]. British Journal of Mathematical and Statistical Psychology, 2006, 59(1): 1-34.

[39] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "Density-based spatial-clustering of applications with noise," in Int.Conf.KnowledgeDiscovery and Data Mining,vol.240, 1996, p.6.

[40] D. Fernandes and J. Bernardino, "Graph databases comparison:Allegrograph, arangodb, infinitegraph, neo4j, and orientdb," in 7thInternational Conference on Data Science, Technology and Applications,2018.

[41] Rosado T, Bernardino J. An overview of openstack architecture[C]//Proceedings of the 18th international database engineering & applications symposium. 2014: 366-367.

[42] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[43] Liu A, Feng B, Xue B, et al. Deepseek-v3 technical report[J]. arXiv preprint arXiv:2412.19437, 2024.

[44] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., et al. (2024). The Llama 3 herd of models. arXiv preprint arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783

[45] Chaplot D S. Albert q. jiang, alexandre sablayrolles, arthur mensch, chris bamford, devendra singh chaplot, diego de las casas, florian bressand, gianna lengyel, guillaume lample, lucile saulnier, lélio renard lavaud, marie-anne lachaux, pierre stock, teven le scao, thibaut lavril, thomas wang, timothée lacroix, william el sayed[J]. arXiv preprint arXiv:2310.06825, 2023.

[46] C.D.Manning, M.Surdeanu, J.Bauer, J.R.Finkel, S.Bethard, and D.McClosky, "The stanford corenlp natural language processing toolkit," in Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014, pp. 55–60.

[47] G.Stanovsky, I.Dagan et al., "Openie as an intermediate structure for semantic tasks," in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), 2015, pp. 303–308.

Authorized licensed use limited to: NANKAI UNIVERSITY. Downloaded on March 08,2026 at 08:14:06 UTC from IEEE Xplore. Restrictions apply.