Root Cause Analysis with Knowledge Graphs and LLMs

Abstract

Background

SRE and operations teams often face fragmented observability tooling and noisy alerts that can show what is broken but not why. This project addresses that gap by fusing multi-modal telemetry (Windows event logs and performance counters) into an explainable knowledge graph that ranks probable root causes and generates human-readable RCA narratives for technical and non-technical stakeholders.

Methods

The system implements a **four-phase causal architecture**.

Phase 1: Data Engineering and Temporal Alignment

- All timestamps are converted to **UTC**.
- Event logs are aligned with performance counters using a **1-minute fuzzy join**.
- Features are engineered to highlight abnormal behavior, including:
    - **Z-score–based anomaly detection** *(Z > 3.0)*
    - **Burst (log-storm) detection** based on events-per-minute patterns

Phase 2: Knowledge Graph Construction

- A directed knowledge graph (**NetworkX DiGraph**) is constructed using a simple ontology:
    - **Nodes:** {System, Component, Event, Metric}
    - **Edges:** {OCCURS_IN, AFFECTS, PRECEDES, CORRELATES_WITH}
- Temporal causal links are modeled using a **60-second PRECEDES heuristic** to capture short-latency event cascades.

Phase 3: Causal Inference and Root Cause Scoring

- Event–metric relationships are strengthened by adding correlation-derived **Event → Metric** edges using windowed correlation analysis.
- Root cause candidates are ranked using a composite graph score:

Score = 0.4(OutDegree) + 0.3(PageRank) + 0.3(Betweenness)

Phase 4: LLM Integration and RCA Narrative Generation

- The knowledge graph is serialized into readable triples and queried using a retrieval step that selects relevant facts for a user question.
- The system supports three LLMs for grounded responses:
    - **google/flan-t5-base (local)** for offline query answering
    - **TinyLlama -1.1B-Chat-v1.0 (local)** as an additional lightweight local generator
    - **OpenAI gpt-5-mini (API)** for higher-quality narrative summaries

Results

- On the **forensicsacl2** incident, the final graph contained **87 nodes** and **828 directed edges**.
- The system computed **4,002** Event–Metric correlations; applying **r > 0.5** added **555** statistically significant correlation-based causal edges.
- Root-cause scoring ranked:
    - **Event 16 (IOMMU Fault)** as the top candidate *(score = 0.7064; out-degree = 31)*
    - **Event 4672** *(score = 0.5652)*
    - **Event 4624** *(score = 0.5520)*
- These results support a coherent causal narrative linking the hardware fault to downstream system degradation.

Table of Contents

1. Introduction

Modern SRE and operations workflows generate large volumes of logs, metrics, and alerts, yet root cause analysis (RCA) remains time-consuming and largely manual. A major reason is that observability data is fragmented across different telemetry sources and tools. Windows event logs provide discrete, contextual signals such as faults, driver issues, authentication events, and service state changes, while performance counters capture continuous system behavior such as CPU, memory, disk activity, and uptime. Because these modalities are analyzed separately and often differ in sampling frequency and timing, responders can usually identify what degraded but struggle to reliably explain why it happened and which upstream event triggered the cascade.

This project proposes an AI-driven multi-modal data fusion pipeline that unifies event logs and performance counters into a single causal representation using a knowledge graph. The graph captures systems, components, events, and metrics as entities and links them through temporal precedence, impact relationships, and statistical coupling. Graph analytics are then used to rank likely root causes, and an LLM layer generates human-readable RCA narratives grounded in retrieved graph facts. The result is a practical RCA assistant that supports both technical investigation and incident reporting.

Objective

Design and implement an end-to-end RCA pipeline that fuses Windows event logs and performance counters, constructs a causal knowledge graph, and applies explainable graph-based scoring to identify and rank the most probable root-cause events.

Goal

Reduce manual RCA effort by automatically producing:

1. **Ranked root-cause candidates**
2. **Evidence-based causal/impact paths** from causes to symptoms
3. **Clear, grounded RCA summaries** suitable for post-incident reviews and stakeholder communication

---

2. Dataset and Methodology

2.1 Dataset Overview

This project uses real system telemetry collected from a Windows environment and combines two complementary data sources: **Windows Event Logs** (discrete, time-stamped system and security events) and **Windows Performance Counters** (continuous resource and health metrics). The dataset is organized by system/host (example host: *forensicsacl2*) and includes timestamps that enable cross-modal alignment. The goal of this phase is to transform raw, heterogeneous telemetry into a unified, time-aligned dataset that can support knowledge graph construction, correlation analysis, and downstream root-cause ranking.

2.2 Windows Event Log Modality

Windows event logs capture discrete operational signals that often contain the earliest evidence of abnormal behavior. These include hardware/driver faults, service state changes, authentication and privilege events, and other system notifications. Each event record typically includes a timestamp, event identifier, and textual description. In this project, events are treated as candidate causes or intermediate steps in a cascade. After cleaning and time standardization, events are used to create graph nodes and temporal edges (**PRECEDES**) that model how one event may lead to another within short time intervals.

2.3 Performance Counter Modality

Performance counters capture continuous measurements of system health such as CPU usage, memory availability, cache activity, disk I/O, and uptime. These metrics represent symptoms and system impacts that help validate whether a candidate root-cause event is associated with observable degradation. Because metrics have different units and scales, they are normalized and transformed into anomaly indicators so they can be compared across counters and linked to nearby events during time alignment.

2.4 Data Preprocessing and Feature Engineering

Preprocessing is required because raw logs and metrics are noisy, asynchronous, and not directly comparable. This project applies several steps to make the data suitable for causal analysis:

2.4.1 Timestamp Standardization (UTC Conversion)

All timestamps are converted to **UTC** to avoid time-zone inconsistencies and to ensure

correct ordering across telemetry sources. This is essential when building temporal relationships and aligning events with metric behavior.

2.4.2 Fuzzy Time-Window Join (1-Minute Alignment)

Event logs and performance counters rarely occur at the exact same timestamp. To align them, both are aggregated into **1-minute windows**, and a fuzzy join is performed so that events and metrics occurring within the same minute are treated as co-occurring. This produces a unified dataset where each time bucket contains the relevant event context and the corresponding metric behavior.

2.4.3 Z-Score Normalization and Anomaly Detection

To detect abnormal metric behavior across different units and ranges, **Z-scores** are computed for each performance metric. Values exceeding a threshold (**Z > 3.0**) are flagged as anomalies, which provides a consistent way to represent metric spikes and drops across counters.

2.4.4 Burst / Log-Storm Detection

To capture sudden surges in event frequency, the pipeline computes **events per minute** and flags bursts (log storms). This feature helps distinguish isolated events from incident-like behavior where many related events occur in a short period.

2.4.5 Unified Dataset Construction and Export

After cleaning, alignment, and feature generation, the system produces and saves a unified dataset that serves as the input to knowledge graph construction. This dataset provides the evidence foundation for correlation edges, causal scoring, and narrative generation in later phases.

---

3. Timestamp Standardization (UTC Conversion)

In this step, all timestamps from both telemetry sources (Windows event logs and performance counters) are converted into a single standardized time reference: **UTC**. This is necessary because raw logs and metrics may be recorded using local system time, different time zones, or inconsistent formats. Without normalization, the ordering of events can be incorrect and cross-source alignment becomes unreliable.

After converting timestamps to UTC, the pipeline ensures:

1. **Correct chronological sequencing** of events for building **PRECEDES** relationships in the knowledge graph.

2. **Accurate alignment** between discrete log events and continuous metric samples during the **1-minute fuzzy join**.
3. **Consistent incident timelines** when analyzing telemetry from multiple hosts or data sources.

This standardization forms the foundation for all later steps, especially temporal windowing, correlation computation, and causal path extraction.

---

4. Fuzzy Time-Window Join (1-Minute Alignment)

Windows event logs and performance counters rarely share the exact same timestamp. Event logs are generated irregularly (only when an event occurs), while performance counters are sampled periodically. To combine them into a single analyzable dataset, the project applies a fuzzy alignment strategy based on fixed time windows.

In this step, both datasets are grouped into **1-minute bins** by flooring timestamps to the nearest minute. All events and metric samples that fall within the same 1-minute window are treated as **co-occurring**. This produces a unified time-indexed table where each row represents one minute of system activity containing:

1. **Events observed in that minute** (or derived event-frequency features).
2. **Performance counter values** and **anomaly indicators** for the same minute.

This alignment is critical because it enables downstream correlation analysis (Event–Metric relationships), supports adding **Event → Metric** edges in the knowledge graph, and makes it possible to trace how discrete events relate to continuous symptom changes over time.

---

5. Z-Score Normalization and Anomaly Detection (Z > 3.0)

Performance counters measure different system quantities (bytes, percentages, counts, milliseconds), so raw values are not directly comparable across metrics. To standardize behavior and highlight abnormal deviations, the pipeline applies **Z-score normalization** to each performance metric across the dataset.

For a given metric, the Z-score is computed as:

Z = (x − mean) / standard deviation

A metric value is flagged as anomalous when **Z > 3.0**, meaning it deviates more than three standard deviations from typical behavior. This threshold is commonly used to isolate rare spikes or drops and helps convert continuous measurements into consistent anomaly signals. These anomaly indicators are later used to connect metric spikes with nearby events during correlation analysis and to strengthen causal evidence in the knowledge graph.

---

6. Burst / Log-Storm Detection (Events per Minute)

Individual events can be noisy and may not indicate a real incident. To capture incident-like behavior, the pipeline adds **burst (log-storm)** features by measuring event volume over time.

In this step, event logs are aggregated into **per-minute counts** (events per minute). Sudden increases in event frequency are treated as bursts, indicating abnormal system activity such as repeated failures, repeated restarts, cascading errors, or security-related activity spikes.

This burst signal complements Z-score metric anomalies: while Z-scores highlight abnormal resource behavior, burst detection highlights abnormal log intensity. Together, they improve the system's ability to identify key time windows and to prioritize events that are more likely to be associated with a true root-cause cascade.

---

7. Unified Dataset Construction and Export

After cleaning, time standardization, window-based alignment, and feature engineering, the pipeline builds a single unified dataset that combines the two telemetry modalities into one time-indexed view. Each **1-minute record** represents a consistent snapshot of system activity and contains:

1. **Event-derived information** (event IDs occurring in the window, counts, and burst-related features).
2. **Performance counter values** for the same window, along with anomaly indicators derived from Z-scores.

This unified dataset is the foundation for downstream modeling because it enables:

- **Event–metric correlation analysis** (used to add **CORRELATES_WITH** edges).

- **Consistent feature extraction** for graph construction and causal scoring.
- **Reproducibility and traceability** of results across the pipeline.

The cleaned and unified tables are exported as **CSV files** (e.g., cleaned events, cleaned performance metrics, unified dataset) so that later stages can be rerun independently and results can be verified.

---

8. Knowledge Graph Construction

The knowledge graph converts fused telemetry into an explainable causal structure. Instead of analyzing logs and metrics as isolated tables, the graph represents entities and their relationships, enabling root-cause ranking and impact-path tracing. The graph is built as a **directed graph (NetworkX DiGraph)**, where nodes represent systems, components, events, and metrics, and edges represent temporal, structural, and statistical relationships.

8.1 Ontology (Nodes and Edge Types)

The graph uses a lightweight ontology to standardize how telemetry is represented.

Node types

- **System:** The host/machine being analyzed.
- **Component:** Subsystem categories (e.g., Memory, LogicalDisk, Processor).
- **Event:** Discrete log events (unique event identifiers).
- **Metric:** Performance counters (e.g., Memory_Available_Bytes, System_Up_Time).

Edge types

- **OCCURS_IN:** Links an **Event** to the **System** where it occurred.
- **AFFECTS:** Links an **Event** to an impacted **Component** (or a **Component** to relevant **Metrics**).
- **PRECEDES:** Captures temporal ordering between events (short-latency causality).
- **CORRELATES_WITH:** Links an **Event** with a **Metric** that changes in association with it.

8.2 Graph Building (OCCURS_IN, AFFECTS Edges)

Graph construction begins by creating nodes for the **System**, **Components**, **Events**, and **Metrics**. The pipeline then adds structural edges that encode basic context:

- **OCCURS_IN** edges connect each event node to its system node, preserving where the event happened.
- **AFFECTS** edges connect events to impacted components (and optionally connect components to metrics), enabling the graph to represent where an event likely acts and which measurements represent symptoms for that component.

These foundational edges create a semantic backbone so later temporal and correlation edges can be interpreted in context.

8.3 Temporal Logic (PRECEDES with 60-Second Heuristic)

To model short-term causal cascades, the pipeline adds **PRECEDES** edges between events based on time ordering. Events are sorted chronologically, and a directed edge **Event A → Event B** is added when **Event B occurs within 60 seconds** after **Event A**.

This heuristic captures immediate propagation chains (e.g., a low-level fault followed by authentication/service events and then resource degradation). These edges support path-based reasoning later when tracing how an initial trigger may lead to downstream effects.

8.4 Correlation Edges (CORRELATES_WITH)

To connect discrete events to continuous symptoms, the pipeline computes correlations between event activity and metric behavior across aligned time windows. When an event is strongly associated with a metric (above a defined threshold such as **r > 0.5**), a **CORRELATES_WITH** edge is added between that **Event** and **Metric**.

These edges strengthen evidence for linking potential causes to observed system impact and improve root-cause ranking by increasing the connectivity and explanatory power of key events.

8.5 Graph Visualization and Export (PNG/HTML/GEXF/GraphML)

To support interpretability and external inspection, the graph is visualized and exported in multiple formats:

- **Static image (PNG)** for inclusion in documentation.
- **Interactive HTML visualization (PyVis)** for exploration in a browser.
- **GEXF and GraphML exports** for analysis in graph tools such as Gephi and for long-term reproducibility.

These outputs allow both technical reviewers and non-technical stakeholders to inspect the causal structure, validate relationships, and understand the reasoning behind the ranked root-cause candidates.

---

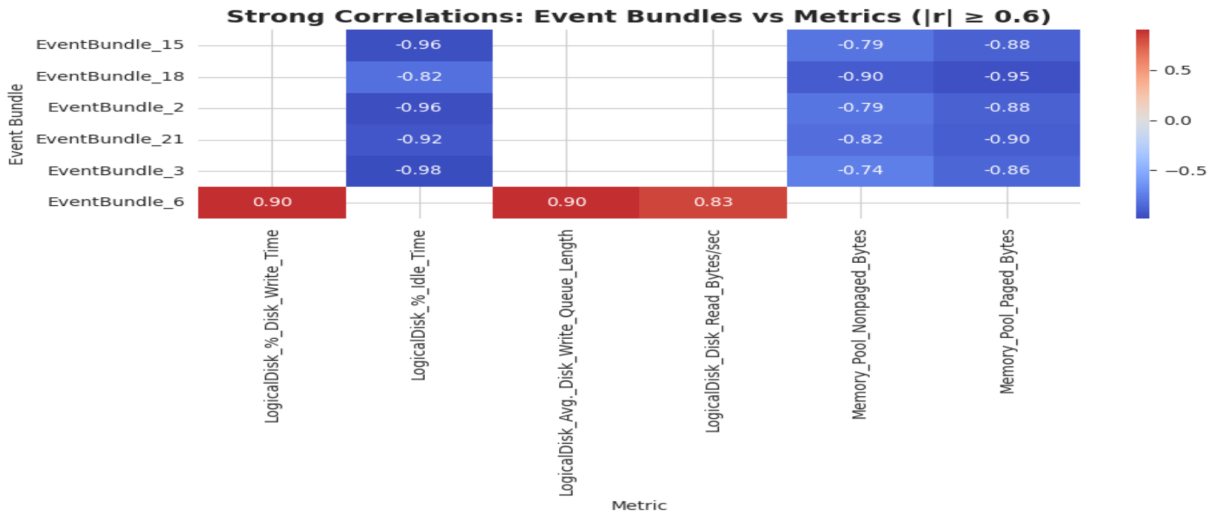
9. Algorithmic Causal Inference and Root Cause Scoring

This section converts the constructed knowledge graph into an RCA engine by adding quantitative evidence (correlations) and applying graph-based algorithms to identify which events are most likely to be true root causes. Instead of relying only on frequency or severity of events, the approach measures how strongly an event is connected to system degradation and how influential it is in propagating effects through the graph. The output of this stage is: **(1) a correlation-enriched causal structure** and **(2) a ranked list of root-cause candidates with explainable scores**.

9.1 Correlation Analysis (Event–Metric Relationships)

To link discrete events to continuous system symptoms, the pipeline computes correlations between event activity and metric behavior across aligned time windows. After the **1-minute alignment**, metrics are analyzed over time and associated with the timing of events (and/or event frequency). For each **Event–Metric** pair, correlation values are computed to measure whether metric changes systematically occur with that event.

When correlation is strong (for example, **r > 0.5**), the relationship is treated as evidence and added to the graph as **CORRELATES_WITH** edges. This step is critical because it provides a quantitative bridge between *what happened* (events) and *what degraded* (metrics), allowing the graph to represent symptom coupling in addition to structural and temporal relationships



9.2 Graph Metrics (Out-Degree, PageRank, Betweenness)

After enriching the graph, the pipeline computes multiple graph centrality measures to estimate causal influence. Each metric captures a different notion of "importance," which is why they are combined later:

1. **Out-Degree**
   Measures how many downstream nodes an event influences (direct outward edges). A high out-degree event tends to act like a trigger that affects many components, metrics, or events.

```
 Top 15 Nodes by In-Degree (Most Affected):
================================================================================
 1. forensicsacl2                          | Type: System     | In-Degree: 50
 2. System_System_Up_Time                  | Type: Metric     | In-Degree: 43
 3. Memory_Cache_Bytes                     | Type: Metric     | In-Degree: 43
 4. Memory_Pool_Paged_Bytes                | Type: Metric     | In-Degree: 42
 5. Memory_Pool_Nonpaged_Bytes             | Type: Metric     | In-Degree: 41
 6. LogicalDisk_Free_Megabytes             | Type: Metric     | In-Degree: 40
 7. LogicalDisk_%_Free_Space               | Type: Metric     | In-Degree: 40
 8. LogicalDisk_%_Idle_Time                | Type: Metric     | In-Degree: 40
 9. Memory_Committed_Bytes                 | Type: Metric     | In-Degree: 38
10. Memory_%_Committed_Bytes_In_Use        | Type: Metric     | In-Degree: 38
11. Process_Handle_Count                   | Type: Metric     | In-Degree: 37
12. Process_Working_Set                    | Type: Metric     | In-Degree: 37
13. Memory_Available_Bytes                 | Type: Metric     | In-Degree: 36
14. Event_4624                             | Type: Event      | In-Degree: 22
15. Event_16                               | Type: Event      | In-Degree: 16

 Top 15 Nodes by Out-Degree (Potential Root Causes):
================================================================================
 1. Event_4672                             | Type: Event      | Out-Degree: 34
 2. Event_16                               | Type: Event      | Out-Degree: 31
 3. Event_15                               | Type: Event      | Out-Degree: 22
 4. Event_16384                            | Type: Event      | Out-Degree: 22
 5. Event_4624                             | Type: Event      | Out-Degree: 21
 6. Event_44                               | Type: Event      | Out-Degree: 21
 7. Event_16394                            | Type: Event      | Out-Degree: 21
 8. Event_5379                             | Type: Event      | Out-Degree: 20
 9. Event_6                                | Type: Event      | Out-Degree: 20
10. Event_43                               | Type: Event      | Out-Degree: 19
11. Event_19                               | Type: Event      | Out-Degree: 19
12. Event_4799                             | Type: Event      | Out-Degree: 18
13. Event_4688                             | Type: Event      | Out-Degree: 18
14. Event_5061                             | Type: Event      | Out-Degree: 18
15. Event_5059                             | Type: Event      | Out-Degree: 18
```

2. **PageRank**
   Measures global influence by considering not only how many connections a node has, but also how influential its neighbors are. Events connected to influential nodes receive higher PageRank, helping capture system-wide impact.

```
 Top 20 Most Influential Nodes (PageRank):
================================================================================
 1. forensicsacl2                                | Type: System     | Score: 0.122340
 2. LogicalDisk_%_Idle_Time                      | Type: Metric     | Score: 0.024374
 3. Memory_Pool_Paged_Bytes                      | Type: Metric     | Score: 0.022812
 4. Event_4624                                   | Type: Event      | Score: 0.021693
 5. Memory_Pool_Nonpaged_Bytes                   | Type: Metric     | Score: 0.020706
 6. System_System_Up_Time                        | Type: Metric     | Score: 0.018787
 7. Memory_Cache_Bytes                           | Type: Metric     | Score: 0.018105
 8. Memory_Committed_Bytes                       | Type: Metric     | Score: 0.017828
 9. Memory_%_Committed_Bytes_In_Use              | Type: Metric     | Score: 0.017828
10. Process_Working_Set                          | Type: Metric     | Score: 0.017619
11. Process_Handle_Count                         | Type: Metric     | Score: 0.017531
12. Memory_Available_Bytes                       | Type: Metric     | Score: 0.017282
13. LogicalDisk_%_Free_Space                     | Type: Metric     | Score: 0.017138
14. LogicalDisk_Free_Megabytes                   | Type: Metric     | Score: 0.017138
15. Event_16                                     | Type: Event      | Score: 0.016992
16. Event_4672                                   | Type: Event      | Score: 0.015480
17. Event_5379                                   | Type: Event      | Score: 0.015419
18. Processor Information_Processor_Frequency | Type: Metric     | Score: 0.012451
19. Event_15                                     | Type: Event      | Score: 0.012314
20. Event_1034                                   | Type: Event      | Score: 0.011793
```

3. **Betweenness Centrality**
   Measures how often a node lies on shortest paths between other nodes. High-betweenness events often behave like bridges or bottlenecks in the propagation of failures.

```
 Top 15 Nodes by Betweenness Centrality (Critical Bridges):
================================================================================
 1. Event_16                                     | Type: Event      | Score: 0.139130
 2. Event_4624                                   | Type: Event      | Score: 0.116752
 3. Event_20                                     | Type: Event      | Score: 0.073794
 4. Event_4672                                   | Type: Event      | Score: 0.059011
 5. Event_4688                                   | Type: Event      | Score: 0.054693
 6. Event_15                                     | Type: Event      | Score: 0.044310
 7. Event_4799                                   | Type: Event      | Score: 0.042166
 8. Event_5379                                   | Type: Event      | Score: 0.040755
 9. Event_18                                     | Type: Event      | Score: 0.035568
10. Event_16384                                  | Type: Event      | Score: 0.035011
11. Event_43                                     | Type: Event      | Score: 0.034763
12. Event_6                                      | Type: Event      | Score: 0.034740
13. Event_27                                     | Type: Event      | Score: 0.034200
14. Event_25                                     | Type: Event      | Score: 0.032832
15. Event_238                                    | Type: Event      | Score: 0.032194
```

9.3 Composite Root-Cause Score

To produce a single interpretable ranking, the project combines the graph metrics into a weighted composite score:

Score = 0.4(OutDegree) + 0.3(PageRank) + 0.3(Betweenness)

The weighting emphasizes out-degree because root causes typically create many downstream effects, while PageRank and betweenness provide additional global and structural importance signals. The final ranking is then used to identify the most probable root-cause events (for example, the top event in the incident was **Event 16** with the highest composite score and high out-degree).

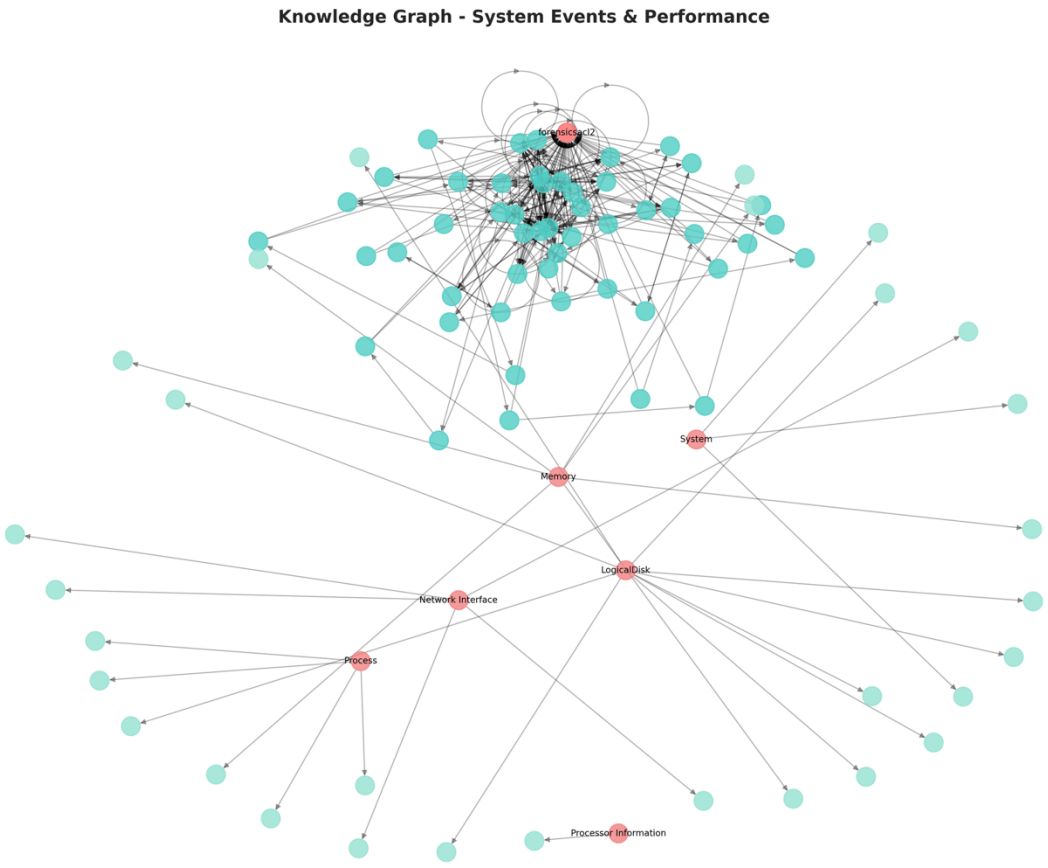
Rank	Event ID	Description	Composite Score	Out-Degree	PageRank	Betweenness
1	Event_16	The iommu fault reporting has been initialized.	0.7064	31	0.1389	1.0000
2	Event_4672	Special privileges assigned to new logon (SYSTEM).	0.5652	34	0.1265	0.4241
3	Event_4624	An account was successfully logged on.	0.5520	21	0.1773	0.8392
4	Event_15	Hive SystemRoot\System32\Config\SOFTWARE was reorganized.	0.3846	22	0.1007	0.3185
5	Event_5379	Credential Manager credentials were read.	0.3610	20	0.1260	0.2929
6	Event_4688	A new process has been created.	0.3564	18	0.0890	0.3931
7	Event_16384	Successfully scheduled Software Protection service for re-start.	0.3547	22	0.0681	0.2516
8	Event_6	File System Filter 'FileInfo' has successfully loaded.	0.3331	20	0.0764	0.2497
9	Event_4799	A security-enabled local group membership was enumerated.	0.3290	18	0.0876	0.3031
10	Event_16394	Offline downlevel migration succeeded.	0.3272	21	0.0661	0.2012

9.4 Impact Path Extraction (Cause → Effects)

After ranking, the pipeline traces causal/impact paths from a top-ranked root-cause event to downstream affected nodes. This is done by traversing the directed graph to extract short paths such as:

Root Event → Intermediate Event(s) → Metric(s) / Component(s)

These paths provide the explainability layer for RCA: instead of only stating that an event is likely a root cause, the system shows how that event connects to symptoms (e.g., memory availability drops, cache increases, uptime anomalies) through temporal and correlation edges. This is particularly useful for validating plausibility during investigations and for writing post-incident summaries.



10. LLM Integration and RCA Report Generation

This section adds a natural-language interface on top of the knowledge graph so that the system can answer investigation questions and generate a readable RCA report. Instead of producing only a ranked list of root causes, the pipeline converts graph evidence into structured facts, retrieves the most relevant evidence for a user query, and uses language models to generate grounded explanations. The LLM stage is designed to be evidence-based: the models are given only retrieved graph facts (triples) so the narrative stays connected to telemetry-derived relationships.

10.1 Triple Extraction and Graph Serialization

The knowledge graph is transformed into a text-based representation by serializing edges into readable triples. Each triple follows a simple **Subject–Predicate–Object** structure derived directly from graph edges, for example:

- **Event_16 PRECEDES Event_4672**
- **Event_16 CORRELATES_WITH Memory_Available_Bytes**
- **Event_16 AFFECTS Memory**
- **Event_16 OCCURS_IN forensicsacl2**

This conversion makes the graph compatible with LLM inputs because it produces a compact, human-readable evidence layer that can be retrieved and inserted into prompts. It also improves transparency because the final narrative can be traced back to specific triples used as supporting evidence.

```
Sample triples:
  • Event_5379 occurs in forensicsacl2
  • Event_5379 precedes Event_5379
  • Event_5379 precedes Event_4799
  • Event_5379 precedes Event_4624
  • Event_5379 precedes Event_44
  • Event_5379 precedes Event_8230
  • Event_5379 precedes Event_7040
  • Event_5379 precedes Event_16
  • Event_5379 precedes Event_6
  • Event_5379 precedes Event_19
```

10.2 Retrieval of Relevant Facts for a Query

Before calling any LLM, the system performs a retrieval step that selects the most relevant triples for the user's question. Instead of feeding the entire graph (which is too large and

noisy), the pipeline ranks triples based on their relevance to the query (keyword overlap / matching on triple text and entities). The **top-k** triples are then used as the context for answer generation.

This retrieval step is important because it:

1. Keeps responses focused on the most relevant evidence.
2. Reduces hallucination risk by restricting the model's context to known graph facts.
3. Improves performance and speed because only a small evidence set is passed to the model.

10.3 Query Answering with Local LLMs (FLAN-T5, TinyLlama)

The project supports two local models for offline question answering:

1. **google/flan-t5-base (local)**
   The primary local model used for text-to-text generation. It takes the user question plus the retrieved triples and outputs a grounded response. It supports privacy-preserving operation because no external API call is required.

   Fig: Sample query of google/flan-t5-base

```
 Query: How many events precedes with another event?
================================================================================

 Found 10 relevant facts:
  1. Event_5379 precedes Event_5379
  2. Event_4624 precedes Event_4672
  3. Event_4672 precedes Event_4624
  4. Event_1034 precedes Event_1034
  5. Event_4799 precedes Event_4799

 Generating answer...

 Answer:
3
================================================================================
```

2. **TinyLlama 1.1B-Chat-v1.0 (local)**
   A lightweight chat-oriented local model included as an additional generator option. It uses the same retrieved triples as context and can be used when a more conversational answer style is desired or as an alternative local model.

Fig: Sample query of TinyLlama 1.1B

```
 Query: How many events precedes with another event?
================================================================================
Found 15 relevant facts.

Answer:
All the events in the Knowledge graph precede with another event.
================================================================================
```

Both local models are intended for environments where logs must remain internal and the RCA assistant must run without external network dependence.

10.4 Optional API Narrative Synthesis (OpenAI gpt-5-mini)

In addition to local models, the notebook includes an optional OpenAI API integration using model **gpt-5-mini**. This model is used specifically for higher-quality narrative synthesis when a polished incident summary is required (for example, executive summaries or final postmortem documents). Importantly, the same retrieval constraint applies: only the retrieved triples are provided as context to keep the output grounded in graph evidence.

Fig: Sample query of gpt-5-mini

```
 Query: How many events precedes with another event?
================================================================================

 Found 10 relevant facts:
  1. Event_5379 precedes Event_5379
  2. Event_4624 precedes Event_4672
  3. Event_4672 precedes Event_4624
  4. Event_1034 precedes Event_1034
  5. Event_4799 precedes Event_4799

 Generating answer with OpenAI gpt-5-mini...

 Answer:
4 events: Event_4624, Event_4672, Event_5058, and Event_16.
================================================================================
```

10.5 Generated RCA Report Output

Finally, the system compiles results into an RCA report output (saved as a text file) containing:

1. Incident context (system/host, timeframe if available)
2. Top-ranked root causes (event IDs and scores)

3. Supporting evidence (key triples and/or correlation signals)
4. Causal/impact paths from root cause to affected metrics/events
5. A natural-language summary generated by the selected LLM

This report format makes the pipeline usable in practice because it produces a single artifact that can be attached to tickets, used in incident reviews, and included in documentation.

```
================================================================================
==========================
ROOT CAUSE ANALYSIS REPORT
================================================================================
==========================

## Executive Summary

Based on analysis of 87 entities and 828 relationships, we have identified
10 primary root causes affecting system performance.

## Top Root Causes

### 1. Event_16

**Description:** The iommu fault reporting has been initialized.

**Impact Score:** 0.7064

**Affects:** 31 downstream entities

**Directly Impacts:**
  - forensicsacl2 (System)
  - Event_6 (Event)
  - Event_11 (Event)
  - Event_4688 (Event)
  - Event_4624 (Event)


### 2. Event_4672

**Description:** Special privileges assigned to new logon.  Subject:
      Security ID:           S-1-5-18    Account Name:          SYSTEM
      ...

**Impact Score:** 0.5652

**Affects:** 34 downstream entities

**Directly Impacts:**
  - forensicsacl2 (System)
  - Event_4624 (Event)
  - Event_16 (Event)
```

```
  - Event_4648 (Event)
  - Event_24 (Event)


### 3. Event_4624

**Description:** An account was successfully logged on.  Subject:
      Security ID:           S-1-0-0    Account Name:          -
      Account D...

**Impact Score:** 0.5520

**Affects:** 21 downstream entities

**Directly Impacts:**
  - forensicsacl2 (System)
  - Event_4624 (Event)
  - Event_4672 (Event)
  - Event_44 (Event)
  - Event_16 (Event)


### 4. Event_15

**Description:** Hive \SystemRoot\System32\Config\SOFTWARE was reorganized
with a starting size of 74772480 bytes and...

**Impact Score:** 0.3846

**Affects:** 22 downstream entities

**Directly Impacts:**
  - forensicsacl2 (System)
  - Event_24 (Event)
  - Event_5379 (Event)
  - Event_44 (Event)
  - Event_4624 (Event)


### 5. Event_5379

**Description:** Credential Manager credentials were read.  Subject:
      Security ID:           S-1-5-18    Account Name:          forensic...

**Impact Score:** 0.3610

**Affects:** 20 downstream entities

**Directly Impacts:**
  - forensicsacl2 (System)
  - Event_5379 (Event)
  - Event_4799 (Event)
  - Event_4624 (Event)
  - Event_44 (Event)
```

```
## Recommendations

1. **Monitor high priority events:** Focus on events with high out degree
centrality.
2. **Implement early warning systems:** Set up alerts for critical root
cause events.
3. **Review system architecture:** Consider isolating components with very
high impact.
4. **Conduct deeper analysis:** Investigate temporal patterns of recurring
root cause events.

================================================================================
==========================
```

---

11. Results

This section summarizes the measurable outputs produced by the pipeline across data preparation, graph construction, correlation enrichment, and root-cause ranking. Results are reported using the artifacts generated by the notebook (CSV exports, saved plots, graph files, and ranked event outputs). The intent is to demonstrate that the system builds a coherent causal structure from raw telemetry and produces explainable rankings and narratives that match observed incident behavior.

11.1 Dataset Processing Summary

The pipeline begins by loading raw event logs and performance counter records and then applies cleaning, time standardization, and alignment. After preprocessing, the system exports cleaned and unified datasets for reproducibility. Typical exported files include:

- **Cleaned event log table** (*events_cleaned.csv*)
- **Cleaned performance counter table** (*perf_cleaned.csv*)
- **Unified aligned dataset** (*unified_data.csv*)

This stage confirms that the telemetry sources can be successfully standardized into a common time index and prepared for downstream correlation and graph building.

Fig: Preprocessing & Normalization Pipeline

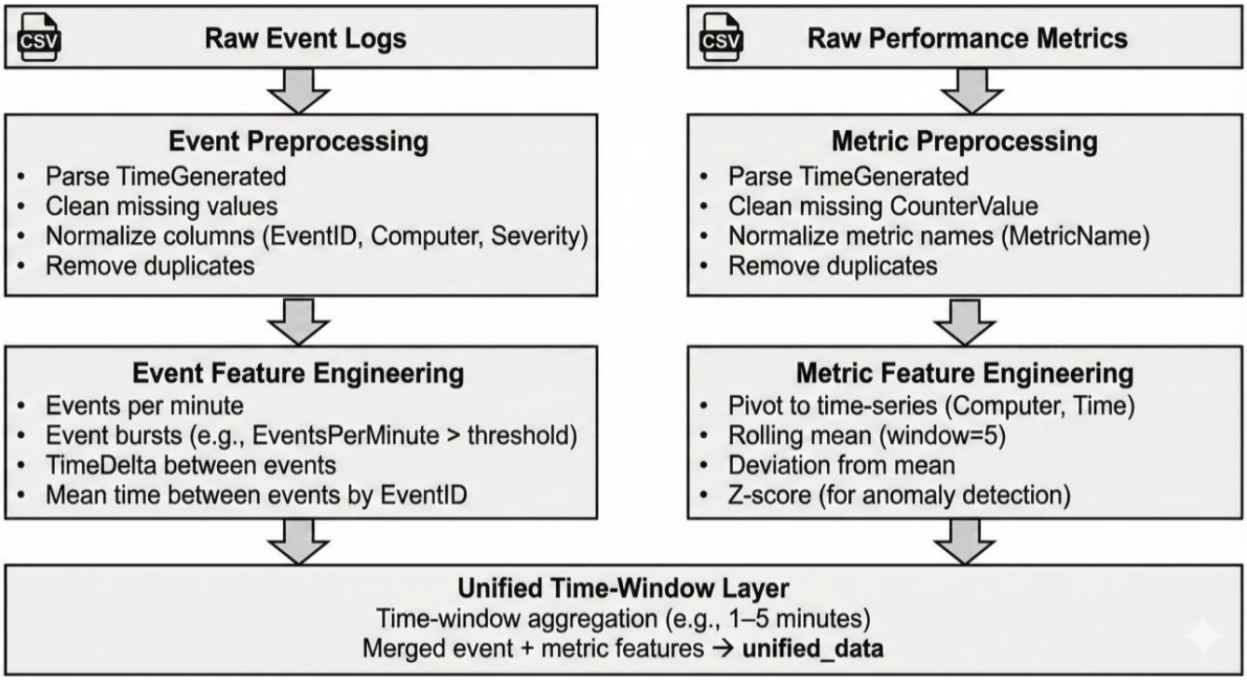


Fig: Sample Images showing Before and after cleaning events and Metrics

```
Events data cleaned!
Original shape: (733, 21) → Cleaned shape: (658, 23)
Removed 75 duplicate rows

Performance data cleaned!
Original shape: (1692, 18) → Cleaned shape: (1692, 20)
Removed 0 rows with missing/invalid values

Data merged successfully!
Unified dataset shape: (43, 191)
```

11.2 Knowledge Graph Statistics (Nodes and Edges)

After graph construction and enrichment, the system reports core graph statistics. These include the total number of nodes (Systems, Components, Events, Metrics) and the total number of directed edges after adding structural, temporal, and correlation relationships.

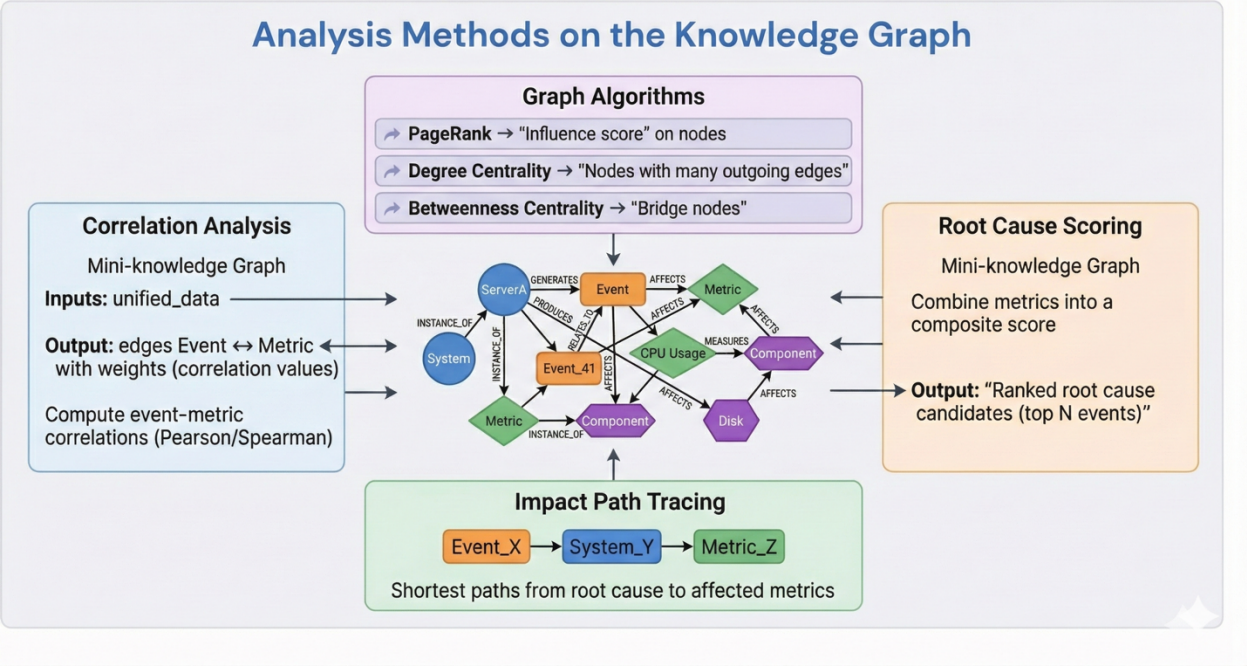
For the *forensicsacl2* incident run, the final graph contained:

- **87 nodes**
- **828 directed edges**

These statistics reflect the combined structure of:

- Base ontology edges (**OCCURS_IN**, **AFFECTS**)
- Temporal cascade edges (**PRECEDES**)
- Correlation edges (**CORRELATES_WITH**)

Fig: Knowledge Graph and Analysis layer



11.3 Correlation Results (Top Relationships)

The correlation module computes **Event–Metric** correlations over aligned time windows to identify which events are most strongly associated with symptom changes. Across the run, the system computed **4,002** Event–Metric correlations. Applying a threshold (**r > 0.5**) added **555** correlation-based edges to the graph.

This result is important because it quantitatively connects discrete event activity with continuous performance degradation, strengthening the causal evidence used for ranking.

11.4 Root Cause Ranking Results (Top Events and Scores)

Root causes are ranked using the composite score:

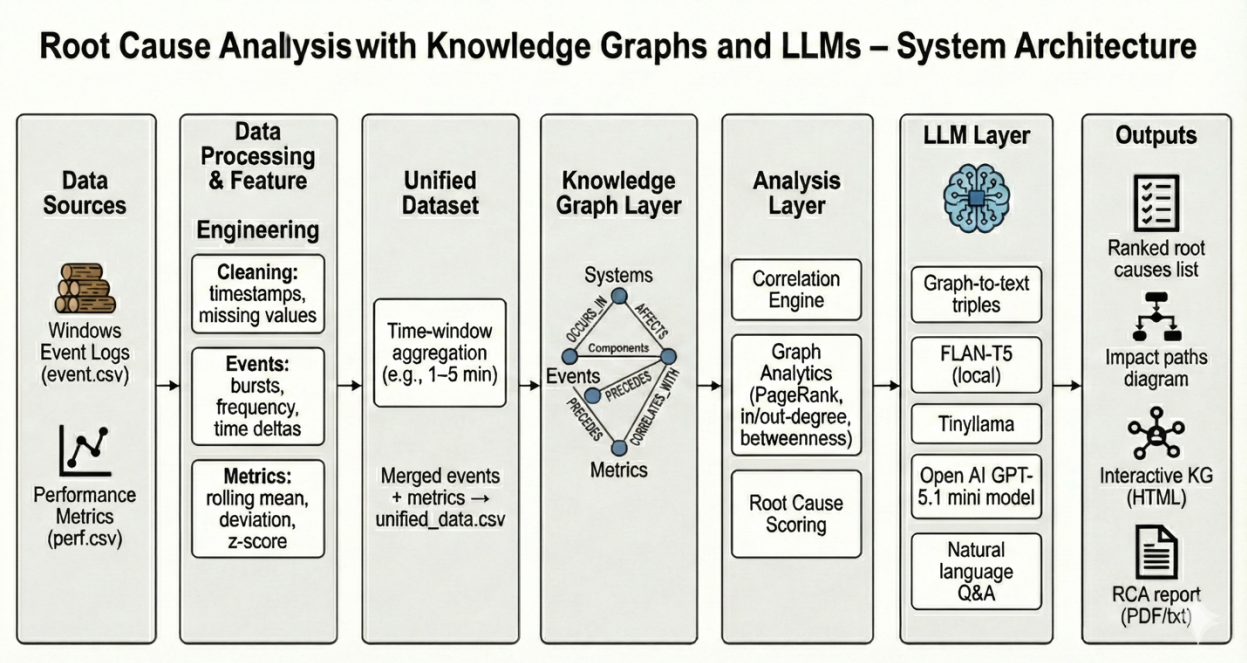
Score = 0.4(OutDegree) + 0.3(PageRank) + 0.3(Betweenness)

For the *forensicsacl2* incident, the top-ranked root-cause candidates were:

1. **Event 16 (IOMMU Fault)** — *score = 0.7064; out-degree = 31*
2. **Event 4672** — *score = 0.5652*
3. **Event 4624** — *score = 0.5520*

The ranking supports a coherent explanation where **Event 16** acts as the earliest high-impact trigger connected to multiple downstream events and degraded metrics.

Fig: Complete System Architecture Overflow



11.5 Example Queries and Generated Explanations

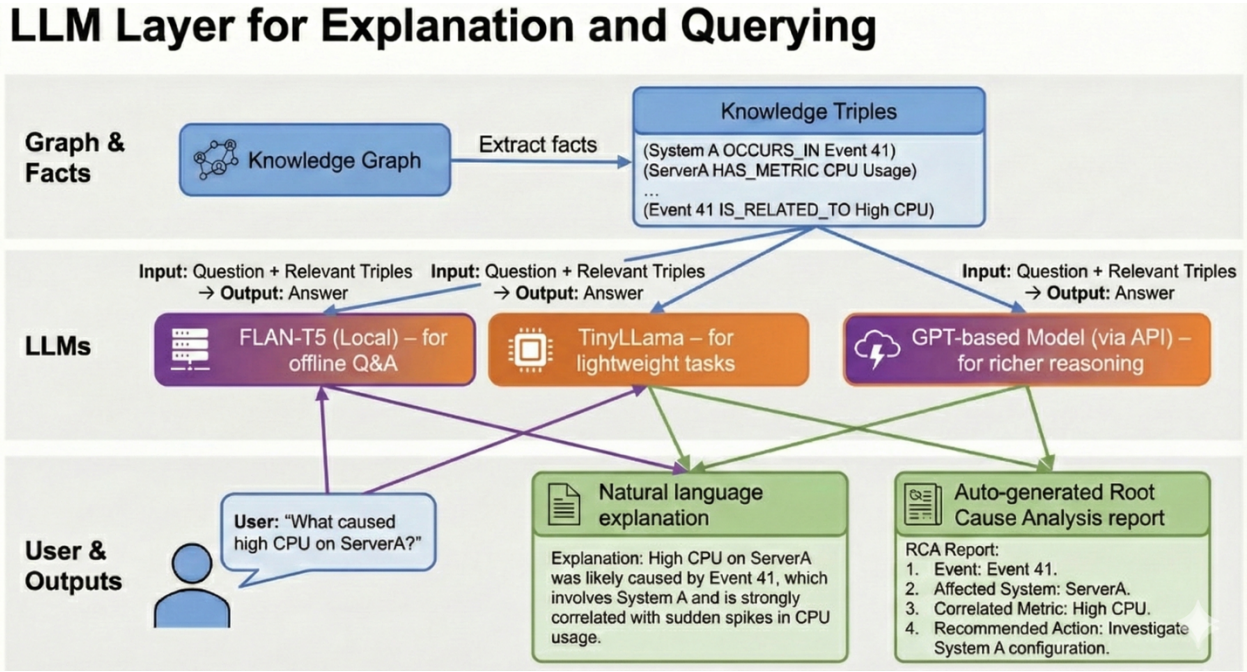
To demonstrate usability, the system supports query-driven explanations using the knowledge graph triple store and LLM generation. Example questions include:

- "What are the most likely root causes affecting system *forensicsacl2*?"

- “Which events are associated with low **Memory_Available_Bytes**?”
- “Show the impact path from the top-ranked event to downstream metrics.”

The generated answers are grounded in retrieved triples and reflect both the ranked root-cause list and the causal paths encoded in the graph. This shows that the system can translate graph evidence into readable explanations suitable for incident reporting.

Fig: LLM Integration layer



12. Discussion

This section interprets the results from an operational perspective and explains why the knowledge-graph approach is useful for root cause analysis in real SRE workflows. It also clarifies the boundaries of the current implementation, including methodological limitations, and addresses privacy and security concerns when handling production telemetry and LLM-based reporting.

12.1 Operational Insights

The pipeline demonstrates that fusing logs and metrics into a single causal structure can reduce investigation friction. In standard incident response, engineers manually correlate log timestamps with metric spikes across dashboards and search tools. Here, the knowledge graph provides a unified view where discrete event triggers and continuous

symptom signals are connected through temporal links (**PRECEDES**) and quantitative coupling (**CORRELATES_WITH**).

A key operational outcome is **prioritization**: instead of scanning thousands of events, responders receive a ranked shortlist of candidate root causes supported by graph evidence (connectivity, influence, and bridge behavior). In practice, this means triage can focus quickly on high-impact events (e.g., an early hardware fault) and validate plausibility by following impact paths from that event to affected metrics and later events. This is particularly useful during postmortems because the system preserves a coherent causal story rather than a collection of disconnected observations.

12.2 Strengths of the Graph-Based Approach

1. **Explainability and Traceability**
   Unlike black-box anomaly detectors, the knowledge graph can justify why an event is ranked highly using transparent signals such as **out-degree** (many downstream effects), **PageRank** (global influence), and **betweenness** (bridge/bottleneck role). The RCA narrative can be traced back to specific triples and edges, which supports auditability.

2. **Multi-Modal Fusion**
   The approach naturally combines discrete and continuous telemetry. Logs provide candidate triggers and context, while metrics provide measurable system impact. Correlation edges connect these modalities, improving confidence that a candidate event is related to observed degradation.

3. **Causal Path Reasoning**
   The graph supports impact-path extraction (**Cause → Effects**). This is operationally valuable because it helps responders quickly understand how an initial trigger propagated, and it enables structured explanations for incident reporting.

4. **Extensible Architecture**
   New node and edge types can be added without redesigning the entire system (e.g., services, processes, network connections, ticket metadata). This makes the approach adaptable to different environments and tool stacks.

12.3 Limitations

1. **Correlation Does Not Guarantee Causation**
   CORRELATES_WITH edges represent statistical association, not proof of causality. Two variables can correlate due to a shared hidden factor, delayed effects, or

coincidental timing. The current system uses correlation as evidence, not as a definitive causal claim.

2. **Sensitivity to Time-Window Choices**
   The **1-minute alignment window** and the **60-second PRECEDES heuristic** are design choices. If the real causal delay is longer or shorter, some relationships may be missed or incorrectly linked. Window sizing also affects correlation stability.

3. **Limited Temporal Modeling**
   The current graph is largely static for an incident snapshot. It does not explicitly model how relationships change over time (e.g., phases of degradation, recovery, feedback loops). This can limit fidelity for complex incidents that evolve over long periods.

4. **Retrieval Simplicity**
   The retrieval step selects triples primarily through text matching. While effective for simple queries, it can miss semantically relevant facts when vocabulary differs (e.g., "memory pressure" vs. **Memory_Available_Bytes**). Dense embedding retrieval would improve this.

5. **LLM Output Quality Depends on Context**
   Local models may produce generic or repetitive outputs if the prompt is long or the retrieved evidence is noisy. The system reduces this risk by retrieving only top facts, but narrative quality still varies by model and prompt design.

12.4 Security and Privacy Considerations

Production telemetry can contain sensitive information (hostnames, usernames, security events, file paths, or workload indicators). The project supports a **local-first** workflow by using local LLMs (**FLAN-T5** and **TinyLlama**) for query answering without requiring logs to leave the environment.

When using an external API model (OpenAI **gpt-5-mini**), only a reduced, retrieved set of graph triples should be shared rather than raw logs or full datasets. This minimizes exposure while still enabling high-quality narrative summaries.

---

13. Future Directions

While the current project scope delivers a robust causal analysis engine, the rapidly evolving field of **AIOps** presents several avenues for future enhancement. The directions below focus on improving retrieval quality, strengthening causal validity, and expanding the system from reactive RCA toward predictive and simulation-driven operations.

13.1. Semantic Retrieval Using Transformers (Dense Retrieval)

The current retrieval step relies on triple extraction and primarily sparse/keyword-based matching to retrieve context for the RAG pipeline. A strong next step is to implement **dense retrieval** using transformer-based models (e.g., **LogBERT** or **Sentence-BERT**). Unlike keyword matching, transformers generate **dense vector embeddings** that capture the semantic intent of log messages and queries. This directly addresses the **vocabulary mismatch** problem common in IT telemetry—for example, a query about "disk saturation" could match evidence containing "no space left on device" or "I/O bottleneck," even when no keywords overlap. This improvement would increase evidence recall and produce more accurate, grounded LLM answers.

13.2. Ontology-Based Reasoning (OWL/RDF + SWRL)

The current system uses a simplified schema with four node types (*System, Component, Event, Metric*). A future enhancement is to evolve this schema into a formal **OWL/RDF ontology** aligned with industry standards and to incorporate **SWRL rules** for automated reasoning. A formal ontology enables **logical deduction** and the discovery of implicit dependencies that may not be explicit in raw telemetry. For example: If *Event A* affects *Component B*, and *Component B* hosts *Service C*, then *Event A* affects *Service C*. This would deepen root-cause discovery by revealing indirect relationships across services and infrastructure layers.

13.3. Dynamic Graph Simulation (Temporal KGs and GNNs)

The current graph represents a static incident snapshot within a fixed time window. A major next step is to implement **Temporal Knowledge Graphs (TKGs)** and **Graph Neural Networks (GNNs)** so the graph evolves over time and can support dynamic simulation. This enables **digital twin** capabilities, including "what-if" fault propagation analysis. For instance, injecting a hypothetical failure (e.g., "What if the database goes down?") would allow the model to predict **cascading impacts** across dependent services and metrics before an incident occurs—shifting the system from reactive RCA toward **predictive maintenance**.

13.4. Reinforcement Learning from Human Feedback (RLHF)

The current composite scoring formula: *0.4(OutDegree) + 0.3(PageRank) + 0.3(Betweenness)* is static. A future iteration can introduce an **RLHF loop** where operators confirm or reject predicted root causes. A **reinforcement learning agent** can then adjust scoring weights dynamically to optimize ranking accuracy based on real incident outcomes, reducing false positives and aligning the system with operational ground truth.

13.5. Neuro-Symbolic AI for Failure Diagnosis

A longer-term research direction is a **neuro-symbolic architecture** that combines the neural generalization of LLMs with the rule-based interpretability of knowledge graphs and

ontologies. This hybrid approach supports **scalable learning** while preserving **explainability** particularly valuable for complex failure diagnosis where both statistical patterns and logical dependency reasoning matter.

---

14. Conclusion

The *Root Cause Analysis via Knowledge Graphs and LLMs* project proposes a shift in how system observability is approached. Instead of treating incident response as a search problem where engineers manually hunt for matching log strings and correlate dashboards this work frames RCA as a *graph problem*, where events, metrics, and components are connected into a structured dependency network. By analyzing these relationships, the system can surface hidden causal chains and propagation paths that are difficult to identify using traditional tools alone.

The **forensicsacl2** incident demonstrates the value of this approach. The pipeline links the discrete trigger of an *IOMMU fault (Event 16)* to downstream system degradation by fusing event logs with performance counters, enriching relationships through correlation evidence, and ranking candidates using explainable impact scoring. The LLM layer then converts retrieved graph facts into clear, grounded narratives that support both investigation and reporting.

Overall, the system goes beyond identifying what failed and where it happened it provides a practical way to explain *why* the incident unfolded, enabling faster triage, more confident root-cause identification, and higher-quality post-incident documentation.

Code Availability

All code used for data preprocessing, knowledge graph construction, correlation analysis, root-cause scoring, visualization, and LLM-based report generation is provided in the project notebook and supporting files.

Git Hub Link: https://github.com/Binduvelpula04/Capstone-Project

---

Approver Table

Approver Name	Title	Signature	Date
Alharthi Dala	**Faculty Advisor/ Mentor**		