electronics

Article

Leveraging Large Language Models for Efficient Alert Aggregation in AIOPs

Junjie Zha *, Xinwen Shan, Jiaxin Lu, Jiajia Zhu and Zihan Liu

State Grid Jiangsu Electric Power Co., Ltd., Information& Telecommunication Branch, Nanjing 210009, China; wxshan@datawisen.cn (X.S.); jxlu@datawisen.cn (J.L.); jjzhu@datawisen.cn (J.Z.); halzh123@126.com (Z.L.)
* Correspondence: zhajunjie08@gmail.com

Abstract: Alerts are an essential tool for the detection of anomalies and ensuring the smooth operation of online service systems by promptly notifying engineers of potential issues. However, the increasing scale and complexity of IT infrastructure often result in "alert storms" during system failures, overwhelming engineers with a deluge of often correlated alerts. Therefore, effective alert aggregation is crucial in isolating root causes and accelerating failure resolution. Existing approaches typically rely on either semantic similarity or statistical methods, both of which have significant limitations, such as ignoring causal relationships or struggling to handle infrequent alerts. To overcome these drawbacks, we propose a novel two-phase alert aggregation approach. We employ temporal–spatial clustering to group alerts based on their temporal proximity and spatial attributes. In the second phase, we utilize large language models to trace the cascading effects of service failures and aggregate alerts that share the same root cause. Experimental evaluations on datasets from real-world cloud platforms demonstrate the effectiveness of our method, achieving superior performance compared to traditional aggregation techniques.

Keywords: alert aggregation; large language model; artificial intelligence for IT operations

check for updates

Citation: Zha, J.; Shan, X.; Lu, J.; Zhu, J.; Liu, Z. Leveraging Large Language Models for Efficient Alert Aggregation in AIOPs. *Electronics* **2024**, *13*, 4425. https://doi.org/10.3390/electronics13224425

Academic Editors: Shuo Yu, Shuai Xu and Minghui Qian

Received: 9 October 2024
Revised: 7 November 2024
Accepted: 8 November 2024
Published: 12 November 2024

1. Introduction

In the contemporary digital era, online services have become indispensable, facilitating critical functions such as online banking, shopping, communication, and information access. The reliability and performance of these systems are paramount, as even brief downtimes can result in substantial financial losses and user dissatisfaction [1–5]. As a result, maintaining seamless online operations is a top priority for businesses. However, system failures are inevitable due to the large scale and complexity of these services [6]. Therefore, service providers deploy monitoring tools that track key performance indicators, logs, and other data reflecting system health. When a metric, such as the latency, surpasses a predefined threshold, an alert is triggered to notify on-call engineers. These real-time alerts allow engineers to diagnose and resolve issues quickly. The alert system faces significant challenges, particularly during "alert storms" [7], where a single system failure triggers a flood of correlated alerts within a short time frame. In large-scale online services, a failure can generate thousands of alerts per minute, overwhelming monitoring systems and complicating issue diagnosis. This overwhelming volume of alerts hinders manual handling, delaying resolution and increasing the risk of prolonged downtime [8].

Motivation. To manage alert storms, researchers have developed alert aggregation techniques to group related alerts, helping engineers to focus on core issues. Current methods include semantic similarity-based and statistical approaches [7,9–12]. Semantic similarity-based methods use natural language processing (NLP) techniques to group alerts with similar textual content. For example, an alert indicating "CPU usage exceeding 90%" might be clustered with one about "high resource utilization". However, these methods struggle to capture the causal relationships between alerts with different textual representations.

Statistical methods, meanwhile, analyze alert co-occurrence patterns based on historical data but may falter as the system configurations change, rendering historical patterns outdated and ineffective for new or infrequent alerts.

The limitations of these methods underscore the critical need for more robust and adaptive alert aggregation techniques that go beyond surface-level similarity or outdated correlation patterns. The inability to detect causal relationships or to adapt to new, unseen alert patterns can lead to prolonged response times, inefficient troubleshooting, and increased service downtime. In particular, as systems scale and the volume of alerts grows, efficient aggregation becomes indispensable in managing alerts in real time and reducing the cognitive load on engineers. Addressing these challenges requires innovative aggregation solutions that can capture the complex dependencies between alerts, ensuring more accurate, efficient, and context-aware incident management in large-scale systems.

Our Approach. To overcome the limitations of existing approaches, we propose a novel two-phase alert aggregation algorithm designed to fully exploit the temporal–spatial locality and cascading effects of service failures. Our approach is grounded in two key observations: (1) alerts triggered by the same root cause tend to occur close together in both time and space and (2) service failures often trigger cascading effects, where a failure in one service can lead to a chain reaction affecting multiple other services. By leveraging these insights, our method effectively groups alerts that are both temporally and spatially related and traces the cascading effects to refine the aggregation. Specifically, our algorithm begins with the coarse-grained temporal–spatial clustering of alerts, grouping those occurring within close time frames and refining clusters based on spatial and textual data. In the second phase, we use large language models (LLMs) to analyze cascading service failures. By mapping clusters from phase one onto a service dependency graph, the LLM traces the failure propagation across services, further aggregating alerts tied to the same root cause. This two-phase approach ensures the comprehensive and accurate aggregation of alerts, enhancing the ability of engineers to diagnose and resolve issues efficiently.

Contributions. We make the following contributions in this paper.

- A novel two-phase alert aggregation algorithm. We introduce a new method for alert aggregation that effectively combines temporal–spatial clustering with LLM-based cascading effect tracing. This approach significantly improves the effectiveness of alert aggregation.
- Application of LLMs in incident management. We demonstrate the use of LLMs to analyze the relationships between services in an online system and trace the cascading effects of failures. This represents a novel application of LLMs in the domain of incident management and alert aggregation.
- Comprehensive evaluation. We conduct extensive experiments using three real-world datasets collected from a production-level online service system in the electric power industry. Our results show that our method significantly outperforms existing approaches.

This paper is structured as follows. Section 2 presents the background and problem statement. In Section 3, we provide a detailed description of our approach. Section 4 evaluates the effectiveness of our approach. Finally, Section 6 concludes the paper.

2. Background and Problem Statement

In this section, we provide an overview of the incident management workflow in online information systems and briefly introduce large language models. We then formalize the problem addressed in this paper.

2.1. Incident Management in Online Service Systems

Online service systems host a vast array of services and applications that must operate continuously to meet user expectations. However, incidents such as system outages, slowdowns, or security vulnerabilities can occur, impacting service availability and the user experience. Incident management refers to the structured process of identifying, assessing, and resolving such disruptions to ensure service continuity and minimize downtime. The four main stages of incident management are illustrated in Figure 1.

- Stage ① **Detection**: Detection is the first step in incident management, where automated systems monitor service performance and generate alerts for anomalies or threshold breaches. These alerts, whether triggered by performance, failures, or security issues, are reported as incidents, with users or engineers also able to report issues manually. The goal is early problem identification to minimize the impact.
- Stage ② **Triaging**: After detection, incidents enter a triaging process in a centralized management system, where they are prioritized by severity and urgency, and on-call engineers (OCEs) are assigned. High-priority incidents, like outages or security breaches, receive immediate attention, while less critical issues follow in order of priority. This process ensures that resources are focused on critical problems, reducing bottlenecks, minimizing false alarms, and routing incidents effectively.
- Stage ③ **Root cause analysis**: Root cause analysis seeks to identify the underlying cause of an incident beyond its symptoms. OCEs use diagnostic tools like log analysis and tracing, collaborating across teams to examine factors such as the infrastructure, code, and dependencies. By pinpointing the root cause, they can implement solutions to prevent future incidents.
- Stage ④ **Mitigation**: In the mitigation stage, OCEs take corrective actions—like rolling back updates, applying patches, or restarting services—to resolve the incident and restore normal operations. While focused on quick resolution, mitigation also involves steps to prevent recurrence, including refining detection systems and updating configurations to avoid future disruptions.

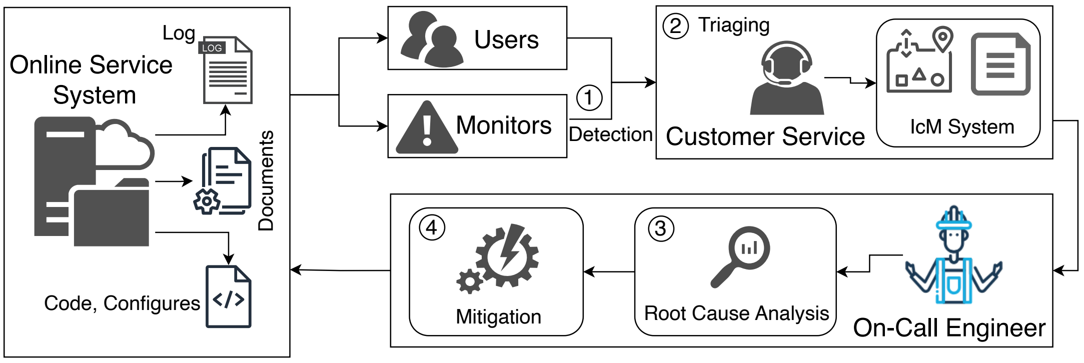
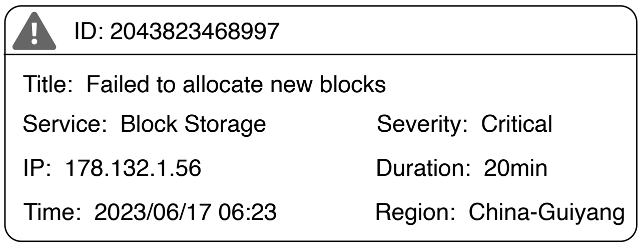


Figure 1. The overall workflow of incident management.

2.2. Alerts in Online Service Systems

In online service systems, alerts are essential notifications that signal potential anomalies or issues, triggered when predefined rules or thresholds are violated. Monitoring tools continuously track system metrics, logs, and traces, capturing key performance indicators such as the CPU usage, memory consumption, and error rates. Engineers set specific thresholds or define patterns to monitor these metrics. When the system behavior deviates, alerts are automatically generated to highlight the issue. These alerts enable the rapid detection of problems, allowing engineers to diagnose and address them before they escalate, thereby minimizing service disruptions.

Alerts in online service systems possess key attributes that help engineers to diagnose and resolve issues efficiently, including the title of the alert, service name, severity level, time, duration, and region information. The title of an alert provides a brief yet informative description, often including the affected service or microservice and the nature of the issue, such as performance degradation or failure. The service name identifies the specific service experiencing the anomaly. The severity level categorizes the alert based on its criticality, indicating how urgently it should be addressed. The time of occurrence marks when the alert was triggered, while the duration tracks how long the alert persists until it is cleared. The region information provides precise details about where the problem occurred, whether in a particular server, cluster, or data center. These attributes help engineers to quickly assess and respond to incidents, following predefined actions to resolve the issue efficiently. Figure 2 shows an example of an alert.



ID: 2043823468997

Title: Failed to allocate new blocks

Service: Block Storage — Severity: Critical

IP: 178.132.1.56 — Duration: 20min

Time: 2023/06/17 06:23 — Region: China-Guiyang

Figure 2. Example of an alert.

2.3. Large Language Models

LLMs are advanced AI models designed to understand and generate human language. Trained on vast datasets from diverse text sources, LLMs capture complex linguistic patterns and contexts. These models utilize deep learning, especially Transformer architectures, to handle long-range dependencies between words. Popular examples include OpenAI's GPT, Google's Gemini, and Meta's LlaMA. LLMs excel in natural language tasks like text generation, translation, summarization, and question answering. Their ability to generate contextually relevant text has proven invaluable in domains like customer service, content creation, and research. In the field of Artificial Intelligence for IT Operations (AIOps), LLMs offer transformative solutions for the handling of vast amounts of operational data. They enhance log analysis, automate incident resolution, and provide contextual insights into system anomalies by interpreting both structured and unstructured data. As LLMs continue to evolve, their integration into AIOps is revolutionizing IT management, providing smarter, faster, and more efficient operational workflows.

2.4. Problem Statement

In this paper, we study the problem of alert aggregation, which involves automatically clustering alerts triggered by the same underlying failure in large-scale online service systems. When a failure occurs in these systems, it often generates numerous alerts, each highlighting different symptoms or affected components. Engineers are then faced with the tedious and time-consuming task of manually examining each alert to diagnose the root cause. This process becomes increasingly challenging as the volume of alerts grows in complex systems. Our goal is to develop an automated alert aggregation solution that clusters related alerts in real time. This approach reduces the noise caused by redundant or unrelated alerts, providing engineers with a clearer view of the problem's scope. By focusing on the core issues, engineers can improve the speed and effectiveness of failure diagnosis and resolution.

3. Methodology

3.1. Overview of Our Approach

In large-scale online service systems, alerts generated by monitoring tools often co-occur due to underlying interdependencies. Our approach seeks to identify and aggregate these co-occurring alerts to simplify the analysis. Before introducing our algorithm, we present two key observations regarding alert behavior.

- **Observation 1: Temporal–Spatial Locality.** A fundamental principle in alert aggregation is temporal–spatial locality, which suggests that alerts originating from a common root cause tend to occur in close temporal and spatial proximity. When a service in an online system malfunctions, it typically impacts other closely related components or services. For instance, a network outage in a cloud environment may trigger a series of alerts from interconnected services dependent on the affected network segment. Temporally, alerts appearing in quick succession are likely related, while, spatially, alerts from topologically adjacent services often indicate a shared underlying issue. By leveraging these patterns, alert aggregation methods can consolidate related alerts, helping engineers to more efficiently identify the root causes of failures.

- **Observation 2: Cascading Effects of Service Failures.** Another critical insight involves the cascading effects of service failures. In online service ecosystems, a failure in one service can trigger a chain reaction, leading to the breakdown of multiple dependent services and the generation of numerous alerts. For example, the failure of a database may result in a cascade of alerts from services relying on database access. This domino effect can produce a flood of alerts, obscuring the primary failure under a multitude of secondary issues. Recognizing this cascading phenomenon enables alert aggregation techniques to cluster related alerts, allowing engineers to focus on resolving the core issue rather than being overwhelmed by the volume of derivative alerts.

Based on the above two observations, the key to effectively aggregating the alerts is to fully exploit the temporal–spatial locality between alerts and the underlying cascading effects of service failures that lead to the flood of alerts. To achieve this, we propose a two-phase alert aggregation algorithm, as illustrated in Figure 3.

- In Phase 1, we perform coarse-grained temporal–spatial clustering in two steps. (1) In the first step (step ① in Figure 3), alerts are grouped based on their temporal information. (2) In the second step (step ② in Figure 3), these grouped alerts are further clustered based on their spatial and textual information. By the end of Phase 1, temporally and spatially similar alerts are grouped together.
- In Phase 2, we leverage LLMs to trace the cascading effects in the alerts. Specifically, (1) we map the clusters obtained in Phase 1 into a service dependence graph that captures possible relationships between services in the online service system that could lead to cascading failures (step ③ in Figure 3). (2) We then aggregate the clustered alerts that were caused by the same service failure (step ④ in Figure 3).

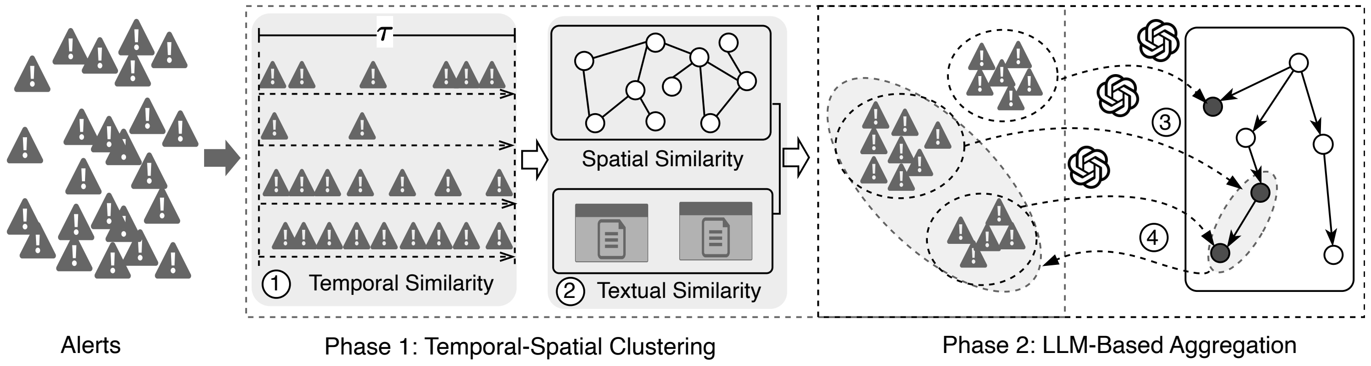


Figure 3. Overview of our two-phase alert aggregation algorithm.

Following the above two phases, our approach ensures that alerts are effectively aggregated, providing a clearer and more manageable view of the underlying issues, thereby enhancing the efficiency and effectiveness of failure diagnosis and resolution.

3.2. *A Two-Phase Alert Aggregation Algorithm*

Following the main idea of our approach presented in Section 3.1, in this section, we first introduce the coarse-grained temporal–spatial clustering algorithm in Section 3.2.1. We then detail the fine-grained LLM-based aggregation algorithm in Section 3.2.2.

3.2.1. A Coarse-Grained Temporal–Spatial Clustering Algorithm

Our coarse-grained hybrid clustering method aims to group temporally and spatially related alerts. To achieve this, we first define the temporal similarity between two alerts, a_1 and a_2, as follows.

Definition 1 (**Alert Temporal Similarity**). *Given two alerts a_1 and a_2 with creation time $a_1.time$ and $a_2.time$, respectively, let τ be a given threshold defining the allowable temporal difference between the two alerts; the temporal similarity of a_1 and a_2, denoted by $\mathcal{S}_{temp}(a_1, a_2)$, is defined as*

$$\mathcal{S}_{temp}(a_1, a_2) = \begin{cases} 1 & if |a_1.time - a_2.time| \leq \tau, \\ 0 & otherwise. \end{cases}$$

Definition 1 captures the closeness of alerts in time. Two alerts triggered within a time interval τ are considered similar; otherwise, they are regarded as dissimilar. For spatial similarity, based on the information provided by the alert introduced in Section 2.2, we can easily identify the component that generates the given alert using information such as the service name, IP, and region. On the other hand, graphs excel in representing relationships between entities [13–22], and the enterprise topology graph [23–25], a fine-grained technical snapshot of the enterprise's IT infrastructure, provides a formal way to model these components and their relationships and can be easily obtained practically, as defined below.

Definition 2 (**Enterprise Topology Graph**). *An enterprise topology graph is a graph $\mathcal{G}_{etg} = (\mathcal{V}, \mathcal{E})$ in which each node $v \in V$ represents an IT component, such as processes, services, software, or infrastructure, and a typed edge $e \in E$ connecting two nodes represents the relation between the corresponding components.*

With Definition 2, for each alert, we can find the corresponding node in the enterprise topology graph discussed above. Therefore, we can capture the spatial similarity between two alerts by defining the similarity of the corresponding nodes in the enterprise topology graph. To achieve this goal, we use node2vec [26], a graph embedding algorithm that learns low-dimensional representations of the nodes in a graph by optimizing random walk strategies to capture both local and global network structures so as to compute the embedding of these two nodes. Then, the similarity of these two nodes is measured by the cosine similarity of the corresponding embeddings.

Definition 3 (**Alert Spatial Similarity**). *Given two alerts a_1 and a_2, with corresponding nodes v_1 and v_2 in the enterprise topology graph that generate a_1 and a_2, let $\mathsf{node2vec}(v_1)$ and $\mathsf{node2vec}(v_2)$ represent the corresponding embeddings of v_1 and v_2 generated by the node2vec model. The spatial similarity of a_1 and a_2, denoted by $\mathcal{S}_{spa}(a_1, a_2)$, is the cosine similarity between $\mathsf{node2vec}(v_1)$ and $\mathsf{node2vec}(v_2)$:*

$$\mathcal{S}_{spa}(a_1, a_2) = \frac{\mathsf{node2vec}(v_1) \cdot \mathsf{node2vec}(v_2)}{\|\mathsf{node2vec}(v_1)\| \|\mathsf{node2vec}(v_2)\|},$$

where $\cdot$ denotes the dot product, and $\|\mathsf{node2vec}(v_1/v_2)\|$ are the Euclidean norms of $\mathsf{node2vec}(a_1/a_2)$.

Moreover, alerts describing the same or closely related issues often share semantically similar descriptions, even if they do not match exactly in wording [7]. Thus, measuring the textual similarity helps to group related alerts and reduce alert fatigue. To compute the textual similarity, we use Sentence-BERT (SBERT) [27], a neural model that generates high-quality sentence embeddings. SBERT is a modified version of the BERT (Bidirectional Encoder Representations from Transformers) model [28], specifically optimized for tasks involving sentence pairs, such as similarity comparisons. Unlike the traditional BERT, which requires pairwise comparisons for sentence similarity, SBERT maps sentences to a fixed-dimensional vector space, where the semantic similarities between sentences are captured in their spatial proximity. This significantly reduces the computational complexity and allows for an efficient comparison between a large number of alerts. SBERT has demonstrated superior performance over traditional methods like TF-IDF or bag-of-words, especially in capturing nuanced semantic relationships between texts [29]. Once the textual embeddings for the alerts are generated, we compute the cosine similarity between the embeddings of two alerts to quantify their semantic closeness. Formally, we have the following.

Definition 4 (Alert Textual Similarity). *Given two alerts a_1 and a_2, let $\mathsf{SBERT}(a_1)$, $\mathsf{SBERT}(a_2)$ be the corresponding embeddings generated by the Sentence-BERT model regarding a_1 and a_2. The textual similarity of a_1 and a_2, denoted by $\mathcal{S}_{text}(a_1, a_2)$, is defined as the cosine similarity between $\mathsf{SBERT}(a_1)$ and $\mathsf{SBERT}(a_2)$, i.e.,*

$$\mathcal{S}_{text}(a_1, a_2) = \frac{\mathsf{SBERT}(a_1) \cdot \mathsf{SBERT}(a_2)}{\|\mathsf{SBERT}(a_1)\| \|\mathsf{SBERT}(a_2)\|},$$

where $\cdot$ is the dot product, and $\|\mathsf{SBERT}(a_1/a_2)\|$ are the Euclidean norms of $\mathsf{SBERT}(a_1/a_2)$.

Following the alert spatial similarity and alert textual similarity, we further define the alert hybrid similarity by combining them together as follows.

Definition 5 (Alert Hybrid Similarity). *Given two alerts a_1 and a_2, the hybrid similarity of a_1 and a_2, denoted by $\mathcal{S}(a_1, a_2)$, is defined as*

$$\mathcal{S}(a_1, a_2) = \alpha \times \mathcal{S}_{spa}(a_1, a_2) + (1 - \alpha) \times \mathcal{S}_{text}(a_1, a_2),$$

where α is a hyper-parameter balancing the weights of the alert spatial similarity and alert textual similarity, respectively.

Based on the alert temporal similarity and alert hybrid similarity, we conduct our coarse-grained clustering following the DBSCAN framework [9], due to its effectiveness and scalability in practice, as detailed in Algorithm 1.

Algorithm 1: Temporal–Spatial Cluster ($\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, τ ϵ, μ)

```
Initialize set of clusters $\mathbb{C} = \emptyset$;
Mark all alerts in $\mathcal{A}$ as unvisited;
Sort the alerts in $\mathcal{A}$ based on their creation time;
Partition the alerts in $\mathcal{A}$ into subgroups $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ following alert temporal similarity based on
  Definition 1;
for each alert $a_i \in \mathcal{A}_i$ do
    if $a_i$ is unvisited then
        Mark $a_i$ as visited;
        $N_\epsilon(a_i) \leftarrow \{a_j \in \mathcal{A}_i \mid \mathcal{S}(a_i, a_j)$(Definition 5) $\geq \epsilon\}$;
        if $|N_\epsilon(a_i)| \geq \mu$ then
            Create a new cluster $C_k$;
            Add $a_i$ to $C_k$;
            ExpandCluster ($a_i$, $C_k$, $N_\epsilon(a_i)$, $\mathcal{A}_i$, $\epsilon$, $\mu$);
            Add $C_k$ to $\mathbb{C}$;
return $\mathbb{C}$ ;
Procedure ExpandCluster ($a_i, C_k, N_\epsilon(a_i), \mathcal{A}_i, \epsilon, \mu$) ;
for each alert $a_j \in N_\epsilon(a_i)$ do
    if $a_j$ is unvisited then
        Mark $a_j$ as visited;
        $N_\epsilon(a_j) \leftarrow \{a_l \in \mathcal{A}_i \mid \mathcal{S}(a_j, a_l)$ (Definition 5) $\geq \epsilon\}$;
        if $|N_\epsilon(a_j)| \geq \mu$ then
            $N_\epsilon(a_i) \leftarrow N_\epsilon(a_i) \cup N_\epsilon(a_j)$;
    if $a_j$ is not yet in any cluster then
        Add $a_j$ to cluster $C_k$;
```

Algorithm. Given an alert set $\mathcal{A} = (a_1, a_2, \ldots, a_n)$ and parameters τ, ϵ and μ, the algorithm begins by initializing an empty set $\mathbb{C}$ to store the clusters (line 1). Each alert in $\mathcal{A}$ is marked as "unvisited" to track the processing status (line 2). The alerts are then sorted chronologically by their creation time and grouped based on their temporal similarity, as defined in Definition 1 (lines 3–4). The algorithm then computes $N_\epsilon(a_i)$, the set of alerts in $\mathcal{A}i$ that have similarity greater than or equal to ϵ (as shown in Definition 5). If the number of alerts in $N\epsilon(a_i)$ is at least μ, a new cluster C_k is created, and a_i is added to it, followed by the expansion of the cluster through the ExpandCluster procedure (lines 10–13). After processing all alerts, the algorithm returns the set of clusters $\mathbb{C}$ (line 14). The ExpandCluster procedure iterates over each alert a_j in $N_\epsilon(a_i)$ (line 16). If a_j is unvisited (line 17), it is marked as visited (line 18), and $N_\epsilon(a_j)$ is computed to identify all alerts within the similarity threshold ϵ. If $|N_\epsilon(a_j)| \geq \mu$ (line 20), $N_\epsilon(a_i)$ is expanded to include $N_\epsilon(a_j)$ (line 21). If a_j is not yet assigned to a cluster, it is added to the current cluster C_k (lines 22–23).

Example 1. Consider the example alerts shown in Table 1. In this scenario, alerts 1, 3, and 6 are part of an alert storm triggered by the limited disk space. Assume the given allowable temporal difference $\tau = 15$ min. Following Definition 1, alerts 1, 3, 5, and 6 are temporally similar to each other, while alerts 2 and 4 are temporally dissimilar from any other alerts. Among alerts 1, 3, 5, and 6, alerts 1, 3, and 6 are spatially similar, as they originate from the same region and are associated with closely related components based on Definition 3. Additionally, as $\mathcal{S}_{text}(alert1, alert6) = \frac{0.7137}{0.9999} = 0.7137$ according to Definition 4 based on their alert titles, they are considered textually similar. Therefore, when Algorithm 1 finishes, we have the following clusters: $\{\{\{alert1, alert6\}, \{alert3\}\}, alert5\}\}\{alert2\}\{alert4\}$.

Table 1. Alert examples used in the experiments.

No.	ID	Title	Time	Duation	Region	Severity	Service	Component
1	2021103	Failed to allocate new blocks	2022/09/10 06:28	10 min	China-GuiYang	High	Storage	CT1C1DISK
2	1921202	CPU usage is 82%	2022/10/10 06:28	1 min	China-GuiYang	Low	CPU	CT1C1CPU
3	3121239	Commit changes error	2022/09/10 06:32	1 min	China-GuiYang	High	Database	CT1C1DB
4	2483108	Packet loss 30%	2022/10/11 06:28	2 min	China-GuiYang	Medium	Network	CT2C1NET
5	4092123	Commit changes error	2022/09/10 06:38	1 min	China-Shanghai	High	Database	CT2C2DB
6	2021112	Disk is full, block allocation error	2022/09/10 06:28	10 min	China-Guiyang	High	Storage	CT1C1DISK

3.2.2. A Fine-Grained LLM-Based Aggregation Algorithm

In Phase 1, the coarse-grained temporal–spatial clustering algorithm (Algorithm 1) groups the temporally and spatially related alerts together. However, it treats the failures of the online service system in isolation, while a failure in one service can trigger a chain reaction, leading to the breakdown of multiple dependent services and the generation of numerous alerts, as shown in Observation 2 of Section 3.1. In this section, we leverage the LLM to further aggregate the related alerts following the cascading effects. Nevertheless, as our preliminary experimental result shows, directly using LLMs for this purpose is impractical because LLMs lack sufficient knowledge about the relationships between services in an online system. This raises the question of whether understanding the interconnected relationships between services—where one service relies on the functionality or data of another—could enhance our approach. Intuitively, if these dependency relationships are known, issues in one service that impact the performance or stability of others can be identified, thereby improving the effectiveness of our second phase. Fortunately, service dependency graphs [30,31], which provide a comprehensive view of the service dependency relationships within an online service platform, are well studied in the literature.

Definition 6 (Service Dependency Graph [30,31]). *A service dependency graph is a directed graph $\mathcal{G}_{sdg}(V, E)$, where each node $v \in V$ represents a service in an online service system, and each directed edge $e = (v_i, v_j) \in E$ indicates a dependency between service v_i and service v_j.*

As shown in [32,33], service dependency graphs can be easily constructed and successfully applied in practice. Therefore, we use the service dependency graph to guide our alert aggregation algorithm. Specifically, for the temporally similar alert clusters obtained by Algorithm 1, we use an LLM to obtain a summary of the alerts in each cluster. After this, we leverage the LLM to map each cluster to a node in the service dependency graph. Then, we compute the weakly connected component among the mapping nodes in the service dependency graph, and clusters in the same weakly connected component can be aggregated together. Following this idea, our LLM-based aggregation algorithm is shown in Algorithm 2.

Algorithm 2: LLM Aggregate($\mathbb{C}, \mathcal{G}_{sdg}$)

```
1  Initialize an empty set of final clusters $\mathbb{C}_{final} = \emptyset$;
2  for each cluster set $\mathbb{C}' \subseteq \mathbb{C}$ such that all the alerts in $\mathbb{C}'$ are temporally and spatially
     similar do
3      for each cluster $C_i \in \mathbb{C}'$ do
4          Use LLM to generate a summarization of $C_i$ ;
5          Use LLM to map $C_i$ to the corresponding node $v_i$ in $\mathcal{G}_{sdg}$;
6  $\mathcal{G}'_{sdg} \leftarrow$ the subgraph of $\mathcal{G}_{sdg}$ induced by the mapped nodes $v_i$ ;
7  Compute the weakly connected components $\mathbb{W}$ in $\mathcal{G}'_{sdg}$;
8  for each weakly connected components $W \in \mathbb{W}$ do
9      Initialize an empty set of clusters $\mathbb{C}' = \emptyset$;
10     for each node $v_i \in W$ do
11         Add $C_i$ to $\mathbb{C}'$;
12     Add $\mathbb{C}'$ to $\mathbb{C}_{final}$;
13 return $\mathbb{C}_{final}$;
```

Algorithm. Given the clustering result $\mathbb{C}$ obtained by Algorithm 1 and the service dependency graph $\mathcal{G}_{sdg}$, the system initializes an empty set $\mathbb{C}_{final}$ to store the final aggregated clusters (line 1). The algorithm then iterates over each subset of clusters $\mathbb{C}' \subseteq \mathbb{C}$, where the alerts within $\mathbb{C}'$ are temporally and spatially similar (line 2). For each cluster $C_i \in \mathbb{C}'$, a summary of C_i is generated and C_i is mapped to its corresponding node v_i in $\mathcal{G}_{sdg}$ based on the summary by using the LLM (line 4–5). Next, it constructs $\mathcal{G}'_{sdg}$, which is a subgraph of $\mathcal{G}_{sdg}$ derived from the mapped nodes in line 5 (line 6). The algorithm then computes the weakly connected components within $\mathcal{G}'_{sdg}$ (line 7). According to Definition 6, the services within the weakly connected component exhibit high interdependency, implying that the alerts in the corresponding clusters may stem from a single root failure. Consequently, clusters whose corresponding nodes belong to the same weakly connected component are grouped together (line 8–12). Finally, the final aggregated clusters are returned in line 13. To conduct the alert summarization and node mapping in lines 4 and 5, we use the prompt in Figure 4.

Prompt: Please summarize the following alert titles, and recommend the mostly related service name and provide the explanations. The information of these alerts: {Alerts}. And the possible services names: {Service Names}. Please answer in the format like "Summarization: Service Name: Explanation".

Figure 4. The prompt for alert summarization and node mapping.

Example 2. Following Example 1, $\{alert1, alert6\}$ and $\{alert3\}$ are identified as temporally and spatially similar when Algorithm 1 finishes. In Algorithm 2, the LLM maps $\{alert1, alert6\}$ to a disk allocation service and $\{alert3\}$ to a database service. Based on the service dependency graph constructed for our experiment, the database service depends on the disk allocation service. Consequently, $\{alert1, alert6\}$ and $\{alert3\}$ are aggregated together when Algorithm 2 finishes.

4. Evaluation

This section presents our experimental results. All experiments were conducted on a machine equipped with an Intel Xeon 2.4 GHz CPU, 256 GB of RAM, and four NVIDIA RTX 4090 Ti GPUs, running Red Hat Linux 7.3, 64-bit.

Datasets. The three datasets used in our evaluation are sourced from a production-level online service system in the electric power industry. These datasets contain alerts generated between 1 September 2022 and 31 December 2022, spanning over 30 services across 10 distinct regions. Collectively, the datasets comprise approximately 100,000 alerts, along with corresponding information about their correlations. The severity levels of these alerts are categorized as critical, high, medium, and low. Since the datasets are sourced from a production-level online service system, we derive alert storm information based on incident reports submitted by OCEs. Specifically, Dataset I includes 20,123 alerts with 21 alert storms, Dataset II contains 33,424 alerts with 42 alert storms, and Dataset III comprises 46,179 alerts with 67 alert storms. The duration of the alert storms ranges from 2 min to 35 min. Within each alert storm, the number of alerts ranges from a minimum of 370 to a maximum of 6248, and the time between subsequent alerts ranges from 10 s to 5 min. Moreover, each alert storm comprises alerts of different types. For example, a typical alert storm occurs when the database is down, and other services that depend on the database can generate a series of alerts. If the failure is not repaired in time, the generated alerts form an alert storm. Table 1 presents examples of the alerts used in our experiments. Note that the duration of an alert refers to the time span from when the alert is first triggered until it is resolved or no longer active.

Baselines. We compare our approach with two traditional machine learning techniques and one recent research method. FPGrowth [10] is an algorithm used for frequent pattern mining in data mining and association rule learning. DBSCAN [9] is a popular density-based clustering algorithm that can discover semantic clusters by representing alert titles as a bag of words. AlertStorm [7] is a method that enables an empirical investigation of alert storms and introduces a technique to address them. It includes alert storm detection based on Extreme Value Theory and a summarization component focused on denoising and alert discrimination. Our evaluation primarily targets its summarization module. For non-open-source baselines, we replicate their approaches using documented techniques and parameter settings to ensure comparable performance. We set default values of τ to 15 min in Definition 1 and α to 0.4 in Definition 5. To evaluate the effectiveness, we define the following metrics: true positives (TP) are alert pairs identified as correlated in the results and confirmed as related in the ground truth, while true negatives (TN) are alert pairs deemed irrelevant in both the results and ground truth; false positives (FP) are alert pairs identified as correlated in the results but marked as unrelated in the ground truth, and false negatives (FN) are pairs identified as irrelevant in the results but actually correlated in the ground truth. Using these metrics, we define precision as $\frac{TP}{TP+FP}$, recall as $\frac{TP}{TP+FN}$, and F1 as $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision}+\text{recall}}$.

4.1. Effectiveness Compared with Baselines

In this effectiveness evaluation, the proposed alert aggregation algorithm is compared with the three baseline methods: FPGrowth, DBSCAN, and AlertStorm. The results, presented in Table 2, highlight the performance differences in terms of precision, recall, and the F1 score across the three datasets.

Table 2. Effectiveness compared with baselines.

Method	Dataset I			Dataset II			Dataset III		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
FPGrowth	0.428	0.732	0.540	0.412	0.701	0.519	0.371	0.660	0.475
DBSCAN	0.173	0.443	0.248	0.189	0.371	0.250	0.231	0.410	0.295
AlertStorm	0.368	0.643	0.468	0.361	0.612	0.408	0.341	0.642	0.445
Ours	0.801	0.831	0.815	0.820	0.842	0.831	0.830	0.871	0.850

As shown in Table 2, our algorithm demonstrates the highest performance across all datasets, significantly surpassing the baseline methods. For instance, in Dataset I, our method achieves an F1 score of 0.815, compared to FPGrowth, DBSCAN, and AlertStorm, which score 0.540, 0.248, and 0.468, respectively. This trend holds consistently across Datasets II and III, highlighting the robustness of our approach. The superior performance can be attributed to our method's ability to leverage multiple sources of information for the accurate clustering and correlation of alerts, unlike the baselines, which rely on limited or single-dimensional techniques.

Examining the baselines more closely, we observe that while FPGrowth performs reasonably well in terms of recall, it struggles with low precision. For example, in Dataset I, FPGrowth achieves a recall value of 0.732 but a precision value of only 0.428, resulting in a moderate F1 score of 0.540. The high recall is due to its tendency to detect many potential correlations, although this comes at the expense of generating numerous false positives, lowering the overall precision. This outcome reflects FPGrowth's pattern-mining approach, which clusters alerts effectively but lacks a nuanced mechanism to verify true correlations.

DBSCAN, on the other hand, performs the worst among the methods. In Dataset I, it achieves precision of only 0.173 and an F1 score of 0.248, with similarly poor results in other datasets, indicating that its density-based clustering struggles to capture the alert patterns effectively. AlertStorm performs slightly better than DBSCAN but still falls short in terms of accuracy compared to both FPGrowth and our proposed method, achieving an F1 score of 0.468 in Dataset I.

These results underscore the effectiveness of our algorithm, as it significantly outperforms the baseline methods in terms of precision, recall, and the F1 score across all three datasets. This improvement is primarily due to our method's ability to exploit the temporal–spatial locality and cascading effects of service failures, thereby achieving more comprehensive and precise alert aggregation.

4.2. Ablation Study

In this ablation study, we examine the contributions of different components of our proposed algorithm by systematically removing each part and analyzing the results across the three datasets. Table 3 reports the precision, recall, and F1 scores for each ablation setting.

Table 3. Ablation study.

Method	Dataset I			Dataset II			Dataset III		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
w/o temporal info.	0.521	0.561	0.540	0.530	0.572	0.550	0.560	0.581	0.570
w/o spatial info.	0.756	0.761	0.758	0.770	0.783	0.776	0.789	0.791	0.789
w/o textual info.	0.691	0.701	0.695	0.689	0.699	0.693	0.673	0.669	0.670
w/o phase 2	0.511	0.532	0.521	0.501	0.520	0.510	0.560	0.571	0.565
Ours	0.801	0.831	0.815	0.820	0.842	0.831	0.830	0.871	0.850

When temporal information is removed (i.e., setting τ to infinity), the algorithm's performance experiences a notable decline across all datasets. For instance, in Dataset I, the F1 score drops from 0.815 (full algorithm) to 0.540, demonstrating the critical importance of temporal locality in grouping alerts occurring close in time. In online service systems, the temporal co-occurrence of alerts often signals related failures, making temporal clustering essential. By clustering alerts based on temporal proximity, our approach effectively captures these temporal correlations, while the absence of this clustering weakens the algorithm's ability to group related alerts accurately.

When spatial information is omitted (i.e., setting $\alpha = 0$), the results remain relatively stable, with the F1 score on Dataset I only slightly reduced to 0.758, compared to 0.815 with the full algorithm. This suggests that spatial data, while useful, play a less substantial role than temporal and textual information in the alert grouping process. On the other hand, when textual information is removed (setting $\alpha = 1$), there is a moderate decline in performance, with the F1 score decreasing to 0.695 for Dataset I. Our use of Sentence-BERT, which identifies textual similarity, contributes significantly to the algorithm's success. By capturing semantic relationships beyond simple keyword matching, Sentence-BERT infers deeper correlations between alerts, enhancing the algorithm's ability to detect related alerts that may be worded differently but stem from the same root cause.

Removing Phase 2, which focuses on aggregating alerts based on cascading service failures, leads to a substantial drop in performance across all datasets, with the F1 score in Dataset I falling to 0.521. This decrease demonstrates Phase 2's crucial role in capturing the cascading effects of service dependencies, where failures in one service impact others. Without this phase, the algorithm would only detect isolated failures, missing downstream dependencies, which are often present in complex systems.

In conclusion, this ablation study underscores the necessity of temporal and textual components, along with the cascading aggregation in Phase 2, to achieve high algorithmic performance. Temporal information is pivotal in detecting closely occurring failures, while textual similarity aids in identifying related alerts with varied expressions. Phase 2 enables comprehensive alert aggregation, accurately capturing interconnected failures, thus enhancing the overall robustness and reliability of the algorithm.

5. Related Work

In this section, we review the related works on alert management and artificial intelligence techniques for IT operations.

5.1. Alert Management

In the literature, a considerable amount of research has focused on alert management. Ref. [34] proposes a method for API performance monitoring that employs a maximal information criterion-based correlation algorithm to aggregate alerts into more comprehensive and informative hyper-alerts for user notification. Ref. [35] introduces a peer review mechanism designed to rank and assess the significance of alerts. Ref. [36] develop a framework utilizing unsupervised machine learning to cluster alerts and incident tickets based on their textual content. Ref. [7] conducts the first empirical analysis of alert storms using large-scale real-world data, offering valuable insights. Ref. [37] integrates semantic and behavioral information through a deep learning model to synthesize alert data. These approaches primarily emphasize textual information and temporal co-occurrence patterns, while the integration of the service topology remains underexplored. Compared to our proposed method, the existing approaches [7,9,10] generally aggregate alerts in a single phase. This one-phase approach often fails to capture the causal relationships between distinct alerts and struggles to identify new or infrequent alerts, as discussed in Section 1.

5.2. Artificial Intelligence for IT Operations

In response to the increasing demands for reliability and scalability in today's complex, distributed computing environments, Artificial Intelligence for IT Operations (AIOps) has emerged, leveraging machine learning, AI, and big data to tackle modern IT challenges [38–40]. Refs. [41,42] evaluate ChatGPT's capability to parse logs in large online service systems, examining its effectiveness and performance with different prompting methods. Ref. [43] employs Sentence-BERT to extract the semantics of log events and subsequently uses a gated recurrent unit (GRU) model for anomaly detection. Ref. [44] analyzes root causes by constructing a real-time causality graph based on events that summarize various types of metrics, logs, and activities in the system under analysis. Ref. [6] clusters incidents with common causes to facilitate timely triage. For internal failure reports, ref. [45] utilizes troubleshooting guides to reduce the manual workloads of on-call engineers in incident handling. Ref. [46] proposes the RealTCD framework, which leverages domain knowledge to discover temporal causal relationships without requiring interventional targets. Ref. [47] leverage data from various stages of the software development life cycle, employing an in-context example retrieval and retrieval-augmented generation approach to generate mitigation strategies. Refs. [1,2] provides a comprehensive survey of this field.

6. Conclusions

In this paper, we investigate the problem of alert aggregation in online service systems. To address this challenge, we propose a novel method that leverages temporal, spatial, and semantic information, integrated with a large language model (LLM) based on a service dependency graph. Our experiments, conducted on three datasets, demonstrate that our method outperforms traditional approaches in terms of its precision, recall, and F1 score. Future work will focus on refining the service dependency graph and enhancing the LLM's capability to handle rare or unseen alert scenarios.

Author Contributions: Conceptualization, J.Z. (Junjie Zha); Methodology, J.Z. (Junjie Zha) and X.S.; Software, J.Z. (Junjie Zha); Supervision, J.Z. (Jiajia Zhu); Visualization, Z.L.; Writing—original draft, J.Z. (Junjie Zha); Writing—review and editing, J.L., J.Z. (Jiajia Zhu) and Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are unavailable due to privacy or ethical restrictions.

Conflicts of Interest: Authors Xinwen Shan, Jiaxin Lu, Jiajia Zhu and Zihan Liu were employed by the company State Grid Jiangsu Electric Power Co., Ltd. The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AIOps	Artificial Intelligence for IT Operations
LLM	Large Language Model
OCEs	On-Call Engineers

References

1. Notaro, P.; Cardoso, J.; Gerndt, M. A survey of aiops methods for failure management. *ACM Trans. Intell. Syst. Technol. (TIST)* **2021**, *12*, 1–45. [CrossRef]
2. Zhang, L.; Jia, T.; Jia, M.; Yang, Y.; Wu, Z.; Li, Y. A Survey of AIOps for Failure Management in the Era of Large Language Models. *arXiv* **2024**, arXiv:2406.11213.
3. Chen, J.; He, X.; Lin, Q.; Xu, Y.; Zhang, H.; Hao, D.; Gao, F.; Xu, Z.; Dang, Y.; Zhang, D. An empirical investigation of incident triage for online service systems. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montréal, QC, Canada, 25–31 May 2019; pp. 111–120.
4. Xu, S.; Guan, D. CrossPred: A Cross-City Mobility Prediction Framework for Long-Distance Travelers via POI Feature Matching. In Proceedings of the CIKM, Boise, ID, USA, 21–25 October 2024; pp. 4148–4152.
5. Xu, S.; Fu, X.; Pi, D.; Ma, Z. Inferring Individual Human Mobility From Sparse Check-in Data: A Temporal-Context-Aware Approach. *IEEE Trans. Comput. Soc. Syst.* **2024**, *11*, 600–611. [CrossRef]
6. Li, L.; Zhang, X.; Zhao, X.; Zhang, H.; Kang, Y.; Zhao, P.; Qiao, B.; He, S.; Lee, P.; Sun, J.; et al. Fighting the fog of war: Automated incident detection for cloud systems. In Proceedings of the USENIX Annual Technical Conference, Online, 14–16 July 2021; pp. 131–146.
7. Zhao, N.; Chen, J.; Peng, X.; Wang, H.; Wu, X.; Zhang, Y.; Chen, Z.; Zheng, X.; Nie, X.; Wang, G.; et al. Understanding and handling alert storm for online service systems. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, Seoul, Republic of Korea, 5–11 October 2020; pp. 162–171.

8. Yu, Q.; Zhao, N.; Li, M.; Li, Z.; Wang, H.; Zhang, W.; Sui, K.; Pei, D. A survey on intelligent management of alerts and incidents in IT services. *J. Netw. Comput. Appl.* **2024**, *224*, 103842. [CrossRef]
9. Ester, M.; Kriegel, H.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, ON, USA, 2–4 August 1996; AAAI Press: Washington, DC, USA, 1996; pp. 226–231.
10. Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. *ACM Sigmod Rec.* **2000**, *29*, 1–12. [CrossRef]
11. Man, D.; Yang, W.; Wang, W.; Xuan, S. An alert aggregation algorithm based on iterative self-organization. *Procedia Eng.* **2012**, *29*, 3033–3038. [CrossRef]
12. Sun, B.; Wu, K.; Pooch, U.W. Alert aggregation in mobile ad hoc networks. In Proceedings of the 2nd ACM Workshop on Wireless Security, San Diego, CA, USA, 19 September 2003; pp. 69–78.
13. Meng, L.; Shao, Y.; Yuan, L.; Lai, L.; Cheng, P.; Li, X.; Yu, W.; Zhang, W.; Lin, X.; Zhou, J. A survey of distributed graph algorithms on massive graphs. *ACM Comput. Surv.* **2024**, *57*, 1–39. [CrossRef]
14. Meng, L.; Yuan, L.; Chen, Z.; Lin, X.; Yang, S. Index-based structural clustering on directed graphs. In Proceedings of the International Conference on Data Engineering, Kuala Lumpur, Malaysia, 9–12 May 2022, pp. 2831–2844.
15. Wang, Y.; Yuan, L.; Chen, Z.; Zhang, W.; Lin, X.; Liu, Q. Towards efficient shortest path counting on billion-scale graphs. In Proceedings of the International Conference on Data Engineering, Anaheim, CA, USA, 3–7 April 2023; pp. 2579–2592.
16. Chen, Z.; Yuan, L.; Han, L.; Qian, Z. Higher-order truss decomposition in graphs. *IEEE Trans. Knowl. Data Eng.* **2021**, *35*, 3966–3978. [CrossRef]
17. Zhang, J.; Li, W.; Yuan, L.; Qin, L.; Zhang, Y.; Chang, L. Shortest-path queries on complex networks: Experiments, analyses, and improvement. *Proc. VLDB Endow.* **2022**, *15*, 2640–2652. [CrossRef]
18. Wu, X.; Yuan, L.; Lin, X.; Yang, S.; Zhang, W. Towards efficient k-tripeak decomposition on large graphs. In Proceedings of the International Conference on Database Systems for Advanced Applications, Chiang Mai, Thailand, 22–25 April 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 604–621.
19. Chen, Z.; Yuan, L.; Lin, X.; Qin, L.; Zhang, W. Balanced clique computation in signed networks: Concepts and algorithms. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 11079–11092. [CrossRef]
20. Zhang, U. Efficient label-constrained shortest path queries on road networks: A tree decomposition approach. *Proc. VLDB Endow.* **2021**, *15*, 686–698. [CrossRef]
21. Jiang, W.; Ning, B.; Li, G.; Bai, M.; Jia, X.; Wei, F. Graph-decomposed k-NN searching algorithm on road network. *Front. Comput. Sci.* **2024**, *18*, 183609. [CrossRef]
22. Yuan, L.; Li, X.; Chen, Z.; Lin, X.; Zhao, X.; Zhang, W. I/O Efficient Label-Constrained Reachability Queries in Large Graphs. *Proc. VLDB Endow.* **2024**, *17*, 2590–2602. [CrossRef]
23. Binz, T.; Fehling, C.; Leymann, F.; Nowak, A.; Schumm, D. Formalizing the cloud through enterprise topology graphs. In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 742–749.
24. Binz, T.; Breitenbücher, U.; Kopp, O.; Leymann, F. Automated discovery and maintenance of enterprise topology graphs. In Proceedings of the 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, Koloa, HI, USA, 16–18 December 2013; pp. 126–134.
25. He, D.; Yuan, P.; Jin, H. Answering reachability queries with ordered label constraints over labeled graphs. *Front. Comput. Sci.* **2024**, *18*, 181601. [CrossRef]
26. Grover, A.; Leskovec, J. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
27. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP, Hong Kong, China, 3–7 November 2019; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 3980–3990.
28. Devlin, J. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv Preprint* **2018**, arXiv:1810.04805.
29. Kashyap, A.R.; Nguyen, T.; Schlegel, V.; Winkler, S.; Ng, S.; Poria, S. A Comprehensive Survey of Sentence Representations: From the BERT Epoch to the CHATGPT Era and Beyond. In Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL, St. Julians, Malta, 17–22 March 2024; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 1738–1751.
30. Ma, S.P.; Fan, C.Y.; Chuang, Y.; Lee, W.T.; Lee, S.J.; Hsueh, N.L. Using service dependency graph to analyze and test microservices. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 2, pp. 81–86.
31. Gaidels, E.; Kirikova, M. Service dependency graph analysis in microservice architecture. In Proceedings of the Perspectives in Business Informatics Research: 19th International Conference on Business Informatics Research, Vienna, Austria, 21–23 September 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 128–139.
32. Lin, J.; Chen, P.; Zheng, Z. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In Proceedings of the Service-Oriented Computing: 16th International Conference, ICSOC, Hangzhou, China, 12–15 November 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 3–20.

33. Luo, S.; Xu, H.; Lu, C.; Ye, K.; Xu, G.; Zhang, L.; Ding, Y.; He, J.; Xu, C. Characterizing microservice dependency and performance: Alibaba trace analysis. In Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 1–4 November 2021; pp. 412–426.
34. Xu, J.; Wang, Y.; Chen, P.; Wang, P. Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment. In Proceedings of the 2017 IEEE International Conference on Services Computing (SCC), Honolulu, HI, USA, 25–30 June 2017; pp. 35–43.
35. Jiang, G.; Chen, H.; Yoshihira, K.; Saxena, A. Ranking the importance of alerts for problem determination in large computer systems. In Proceedings of the 6th International Conference on Autonomic Computing, Barcelona, Spain, 15–19 June 2009; pp. 3–12.
36. Lin, D.; Raghu, R.; Ramamurthy, V.; Yu, J.; Radhakrishnan, R.; Fernandez, J. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 1630–1639.
37. Chen, J.; Wang, P.; Wang, W. Online summarizing alerts through semantic and behavior information. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 1646–1657.
38. Dang, Y.; Lin, Q.; Huang, P. Aiops: Real-world challenges and research innovations. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montréal, QC, Canada, 25–31 May 2019; pp. 4–5.
39. Levin, A.; Garion, S.; Kolodner, E.K.; Lorenz, D.H.; Barabash, K.; Kugler, M.; McShane, N. AIOps for a cloud object storage service. In Proceedings of the 2019 IEEE International Congress on Big Data (BigDataCongress), Los Angeles, CA, USA, 9–12 December 2019; pp. 165–169.
40. Li, Y.; Jiang, Z.M.; Li, H.; Hassan, A.E.; He, C.; Huang, R.; Zeng, Z.; Wang, M.; Chen, P. Predicting node failures in an ultra-large-scale cloud computing platform: An aiops solution. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2020**, *29*, 1–24. [CrossRef]
41. Mudgal, P.; Wouhaybi, R. An assessment of ChatGPT on log data. In Proceedings of the International Conference on AI-generated Content, Shanghai, China, 25–26 August 2023; Springer: Berlin/Heidelberg, Germany, 2023; pp. 148–169.
42. Le, V.H.; Zhang, H. Log parsing: How far can chatgpt go? In Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, 11–15 September 2023; pp. 1699–1704.
43. Zhang, M.; Chen, J.; Liu, J.; Wang, J.; Shi, R.; Sheng, H. Logst: Log semi-supervised anomaly detection based on sentence-bert. In Proceedings of the 2022 7th International Conference on Signal and Image Processing (ICSIP), Suzhou, China, 20–22 July 2022; pp. 356–361.
44. Wang, H.; Wu, Z.; Jiang, H.; Huang, Y.; Wang, J.; Kopru, S.; Xie, T. Groot: An event-graph-based approach for root cause analysis in industrial settings. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 419–429.
45. Shetty, M.; Bansal, C.; Upadhyayula, S.P.; Radhakrishna, A.; Gupta, A. Autotsg: Learning and synthesis for incident troubleshooting. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 18 November 2022; pp. 1477–1488.
46. Li, P.; Wang, X.; Zhang, Z.; Meng, Y.; Shen, F.; Li, Y.; Wang, J.; Li, Y.; Zhu, W. Llm-enhanced causal discovery in temporal domain from interventional data. *arXiv* **2024**, arXiv:2404.14786.
47. Goel, D.; Husain, F.; Singh, A.; Ghosh, S.; Parayil, A.; Bansal, C.; Zhang, X.; Rajmohan, S. X-lifecycle Learning for Cloud Incident Management using LLMs. In Proceedings of the Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, Porto de Galinhas, Brazil, 17–19 July 2024; pp. 417–428.