DBAIOps: A Reasoning LLM-Enhanced Database Operation and Maintenance System using Knowledge Graphs

Wei Zhou
Shanghai Jiao Tong University
weizhoudb@sjtu.edu.cn

Peng Sun
Baisheng (Shenzhen) Technology Co., Ltd.
sunpeng@dbaiops.com

Xuanhe Zhou*
Shanghai Jiao Tong University
zhouxuanhe@sjtu.edu.cn

Qianglei Zang
Baisheng (Shenzhen) Technology Co., Ltd.
zangqianglei@dbaiops.com

Ji Xu
Baisheng (Shenzhen) Technology Co., Ltd.
xuji@dbaiops.com

Tieying Zhang
Bytedance
tieying.zhang@bytedance.com

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Fan Wu
Shanghai Jiao Tong University
fwu@cs.sjtu.edu.cn

ABSTRACT

The operation and maintenance (O&M) of database systems is critical to ensuring system availability and performance, typically requiring expert experience (e.g., identifying metric-to-anomaly relations) for effective diagnosis and recovery. However, existing automatic database O&M methods, including commercial products, cannot effectively utilize expert experience. On the one hand, rule-based methods only support basic O&M tasks (e.g., metric-based anomaly detection), which are mostly numerical equations and cannot effectively incorporate literal O&M experience (e.g., troubleshooting guidance in manuals). On the other hand, LLM-based methods, which retrieve fragmented information (e.g., standard documents + RAG), often generate inaccurate or generic results.

To address these limitations, we present DBAIOps, a novel hybrid database O&M system that *combines reasoning LLMs with knowledge graphs* to achieve DBA-style diagnosis. First, DBAIOps introduces a heterogeneous graph model for representing the diagnosis experience, and proposes a semi-automatic graph construction algorithm to build that graph from thousands of documents. Second, DBAIOps develops a collection of (800+) reusable anomaly models that identify both directly alerted metrics and implicitly correlated experience and metrics. Third, for any given anomaly, DBAIOps employs an automatic graph exploration mechanism that explores the relevant paths over the graph and dynamically explores potential gaps (missing paths) without human intervention. Based on the explored diagnosis paths, DBAIOps leverages reasoning LLM (e.g., DeepSeek-R1) that inputs the relevant pathways, identifies root causes, and generates clear diagnosis reports for both DBAs and common users. Our evaluation over four mainstream database systems (Oracle, MySQL, PostgreSQL, and DM8) demonstrates that DBAIOps outperforms state-of-the-art baselines, 34.85% and 47.22% higher in root cause and human evaluation accuracy, respectively. DBAIOps supports 25 database systems and has been deployed in 20 real-world scenarios, covering domains like finance, energy, and healthcare (*https://www.dbaiops.com*).

PVLDB Reference Format:
Wei Zhou, Peng Sun, Xuanhe Zhou, Qianglei Zang, Ji Xu, Tieying Zhang, Guoliang Li, and Fan Wu. DBAIOps: A Reasoning LLM-Enhanced Database Operation and Maintenance System using Knowledge Graphs. PVLDB, 19(6): 1319 - 1331, 2026.

*Xuanhe Zhou is the corresponding author.

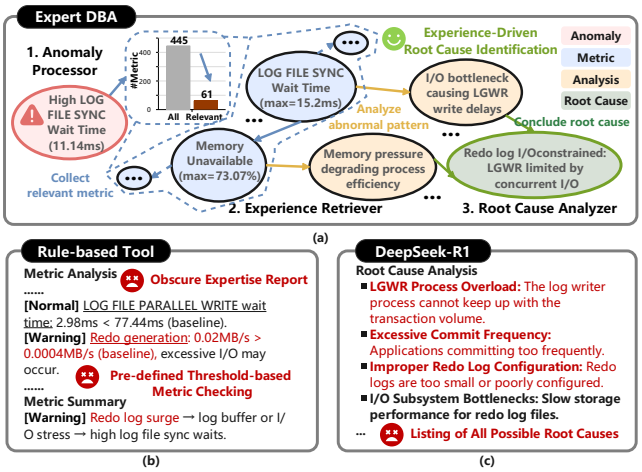


Figure 1: Automatic database O&M is challenging - (a) Expert DBA needs to analyze diverse information from triggered anomalies. (b) Empirical O&M may apply misleading rules (caused by incorrect thresholds). (c) LLMs may lack O&M experience and fail to diagnose *even with sufficient relevant metric information.*

doi:10.14778/3797919.3797937

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/weAIDB/DBAIOps.

1 INTRODUCTION

Database operation and maintenance (O&M) aims to detect, analyze, and resolve various anomalies that arise in target database instances. It is of great importance to meet the rigorous requirements during the online usage of these instances, such as high availability (e.g., less than 52.6 minutes of downtime per year for critical services such as financial and e-commerce systems [27]) and performance (e.g., service-level agreements (SLAs) enforced by cloud service

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797937

providers [10, 21, 46]). For example, the outage of the NOTAM (Notice to Air Missions) database, a critical system managed by the Federal Aviation Administration (FAA) that provides safety alerts to pilots and airlines, led to over 10,000 flight delays and more than 1,300 cancellations, resulting in economic losses amounting to millions of U.S. dollars due to the disruption [8, 13].

Therefore, to ensure high availability and performance, many companies hire senior DBAs (with decades of experience) or purchase costly manual maintenance services [12, 16, 18]. For instance, as shown in Figure 1, diagnosing anomalies such as `LOG_FILE_SYNC` traditionally relies on experienced DBAs to implicitly associate the alert with relevant metrics (e.g., *LOG FILE PARALLEL WRITE Wait Time*) and construct causality chains to identify the root cause (e.g., "I/O bottleneck limiting the LGWR process"). It also involves filtering out large volumes of irrelevant metrics (e.g., selecting 61 relevant metrics from a total of 445), which could otherwise lead to incorrect conclusions. However, this human-intensive O&M process is time-consuming, difficult to scale, and becomes particularly inefficient when repeatedly applied to recurring anomalies.

Existing methods leverage empirical rules [19, 24] or even large language models (LLMs) [38, 39, 44, 48, 49, 51, 52] to automate some tasks in database O&M (see Table 1). However, they do not support flexible DBA experience integration and demonstrate significant limitations. For instance, in Figure 1, rule-based methods such as [19, 24] integrate limited anomaly diagnosis rules with fixed thresholds (e.g., attributing the anomaly to a `Red_log_surge` merely because a preset threshold was exceeded). By contrast, LLM-based methods can infer subtler root causes by reasoning over system metrics and leveraging external knowledge [51]. However, they have two main problems, causing them hardly applicable in real scenarios. First, they have relatively low accuracy in matching relevant expert experience, which (1) naively chunk documents without capturing implicit relationships [38], and (2) rely solely on vector-based similarity for semantic matching [20]. Second, the retrieved experience is often fragmented, making it difficult for LLMs to perform accurate step-by-step root cause analysis. For instance, as shown in Figure 1, DeepSeek-R1 lists multiple possible causes but fails to deliver a concrete, actionable diagnosis, such as identifying the exact root causes (e.g., Redo log I/O bottlenecks) or suggesting concrete recovery actions (e.g., optimizing Redo log placement).

To bridge the gap between experience-based O&M practices of expert DBAs and the limited capabilities of existing methods, there are three main challenges.

C1: How to effectively characterize and integrate O&M experience? While extensive database O&M experience exists across technical notes, scripts, and incident reports (mainly from large database companies [9, 10]), it is often fragmented and distributed in heterogeneous formats (e.g., informal textual documentation, unstructured log entries, and isolated SQL scripts). *There lacks an effective way to represent O&M experience*, hindering their use in guiding O&M tasks and supporting accurate database diagnosis (e.g., correlating recurring `LOG_FILE_SYNC` wait events with storage I/O latency anomalies identified in historical incidents).

C2: How to capture implicitly correlated factors for diverse anomalies? Effective analysis of database anomalies often necessitates a global view of system, log, and trace metrics (e.g., redo log generation rate, I/O subsystem latency, and transaction commit frequency). However, most existing methods focus primarily on metrics that exhibit abnormal patterns (e.g., changes in temporal trends or short-term fluctuations [29, 30, 33]). As a result, metrics that are *implicitly relevant but do not exhibit noticeable abnormal patterns* are frequently overlooked, leading to significant diagnosis errors. For instance, a normal log buffer hit ratio during `LOG_FILE_SYNC` waits may cause DBAs to overlook I/O bottleneck, only focusing on spiking sync wait times.

C3: How to adaptively explore potential O&M diagnosis paths? Existing diagnosis systems often rely on fixed rule-based methods, which follow predefined decision paths and show limited adaptability over diverse scenarios [19, 24]. In addition, many of them either cannot generate informative reports that include root causes and promising recovery solutions [25, 34, 40, 50], or generate overly technical outputs, such as exhaustive lists of metric values, that are difficult for common users to interpret and act upon [32, 33, 37].

To address these challenges, we design `DBAIOps`, an experience-centric database O&M system with five key components: (1) *Heterogeneous O&M Graph Model* (`ExperienceGraph`) for integrating the diverse and mostly-textual O&M experience. This graph represents diagnosis paths through interconnected vertices and edges, enabling the structured organization and incremental enrichment of O&M experience for precise diagnosis **(for C1)**; (2) *Correlation-Aware Anomaly Models* (`AnomalyModel` and `AnomalyProcessor`) for identifying correlated factors associated with input anomaly. Each model incorporates statistical multi-metric correlation analysis, frequency control, and low-code tools to support the discovery of implicitly correlated metrics **(for C2)**; (3) *Two-Stage Graph Exploration with LLM Reasoning* (`ExperienceRetriever` & `RootCauseAnalyser`) that adaptively traverses the graph to collect relevant diagnosis information (e.g., abnormal metrics); then utilizes in-context learning to prompt LLMs for generating comprehensive diagnosis reports that include detailed root cause analysis and practical recovery solutions **(for C3)**.

Contributions. We make the following contributions.

- We design a database operation and maintenance (O&M) system for diagnosing real-world anomalies. *To the best of our knowledge, this is the first database O&M system that integrates knowledge graph with reasoning LLMs to identify root causes and provide recovery solutions* (see Section 3).
- We propose a graph-based experience model to represent O&M experience in graph paths. *Current experience model (the knowledge graph) comprises over 2,000 vertices and 800+ anomaly scenarios for 25 different database systems* (see Section 4 & *https://www.dbaiops.com*).
- We propose a correlation-aware anomaly model to capture implicit correlations across metrics and real-world anomalies, which can trigger more accurate graph exploration during online diagnosis (see Section 5).
- We introduce a two-stage graph exploration mechanism that adaptively explores possible diagnosis paths for different anomalies. We prompt LLM to reason over these diagnosis paths and generate diagnosis reports with specific recovery solutions (see Section 6).
- Extensive experiments and case studies show that `DBAIOps` outperforms both rule and LLM-based baselines in root cause accuracy (34.85% higher) and human evaluation accuracy (47.22% higher).

2 BACKGROUND AND RELATED WORK

Database O&M refers to the process of maintaining and optimizing database systems, which typically involves (1) the collection of necessary O&M factors (e.g., system metrics, logs, and traces) and (2) root cause diagnosis and recovery. As shown in Table 1, we classify existing database O&M methods into three main categories:

► ***Rule-based Methods.*** Methods in this category rely on human experts to incorporate their maintenance knowledge as rules into the diagnosis process, such as by defining a set of diagnosis paths for different types of anomalies [24, 25, 40]. ADDM [24] performs root cause diagnosis in a time graph based on rules (e.g. "exploring all child nodes when a node's time is abnormal"). DBSherlock [40] encodes domain knowledge into rules and uses these rules to filter out predicates ($Attr > k$) that reflect secondary symptoms. ADTS [25] builds an expert system containing 175 rules in the form of "Expression-Result" statements to diagnose root causes.

However, rule-based methods require specialized expertise to design and implement, and are generally limited to specific database systems. For instance, ADDM is only applicable to Oracle database, and extending it to other systems requires considerable manual efforts (e.g., incorporating new rules). In addition, the reliance on pre-defined rules and the absence of external knowledge integration reduces flexibility, making it difficult to adapt to new anomalies.

► ***ML-based Methods.*** Methods in this category use machine learning algorithms or models to enhance the root cause analysis accuracy of rule-based methods. CauseRank [33] employs a Bayesian Network Structure algorithm and expert rules to construct a causal graph of anomalies. DBMind [50] employs an LSTM-based encoder to encode data into anomaly vectors for matching the root cause. iSQUAD [34] employs the Bayesian Case Model to extract the key features of SQLs, while PinSQL[32] employs an ML-based clustering algorithm to group SQLs according to their historical execution trends for root cause diagnosis. RCRank [37] trains a multi-modal machine learning model to extract features from four types of data (SQL, log, plan, and metric) to rank the root causes of slow queries.

However, since ML-based methods are typically built on top of rule-based systems, they inherit similar limitations. Moreover, ML models typically have poor generalization ability due to their strong dependence on training data [31], making them effective only for certain anomaly diagnosis. For example, iSQUAD [34] and PinSQL [32] are designed to diagnose slow SQLs of limited types.

► ***LLM-based Methods.*** Methods in this category leverage the comprehension and reasoning capabilities of LLMs to improve diagnosis accuracy and adaptability. These methods utilize both the LLM 's internal knowledge (e.g., general understanding of different database systems) and external resources (e.g., historical anomaly cases). For example, D-Bot [51] empowers LLM to perform diagnosis with prompts generated with matched document knowledge and retrieved tools and conduct multi-step root cause analysis using the tree-search-based algorithm. ChatDBA [7] leverages a decision tree structure to retrieve relevant information and instruct LLM-driven diagnosis. Panda [38] and GaussMaster [47] utilize LLM agents to specialized diagnosis modules or expert roles for collaborative diagnosis. Andromeda [20] employs Sentence-BERT and seasonal-trend-based metric analysis to enable LLM to leverage information from metrics, historical questions, and diagnosis manuals to generate configuration tuning suggestions.

Table 1: Comparison of Database O&M Methods.

Category	Method	Diagnosis Numeric	Diagnosis Text	Experience Integration	Experience Exploration	New Anomaly Support
Rule-based	ADDM [24]	✓	×	×	×	×
	DBSherlock [40]	✓	×	×	×	×
	ADTS [19]	✓	×	×	×	×
ML-based	CauseRank [33]	✓	×	×	×	×
	DBMind [50]	✓	×	×	×	×
	iSQUAD [34]	✓	×	×	×	×
	PinSQL [32]	✓	×	×	×	×
	RCRank [37]	✓	×	×	×	×
LLM-based	D-Bot [51]	✓	✓	✓	×	✓
	Panda [38]	×	✓	✓	×	✓
	ChatDBA [7]	×	✓	✓	×	✓
	Andromeda [20]	×	✓	✓	×	✓
	GaussMaster [47]	✓	✓	✓	×	✓
	DBAIOps	✓	✓	✓	✓	✓

Although LLM-based methods offer high generalization ability and can generate flexible diagnosis outputs, they have several limitations. First, they prompt LLMs using some general documents only (e.g., basic O&M concepts), based on which LLMs (even equipped with advanced techniques like tree of thought [51]) easily yield generic results or meet diagnosis failures (e.g., analyzing over non-existent metrics). Second, while the LLM+RAG approach allows dynamic document knowledge retrieval [20], typical RAG paradigm conducts top-k matching of the separated knowledge chunks, which destroys the original knowledge relations (e.g., a diagnosis path involving multiple steps) and causes inaccurate or incomplete diagnosis. Besides, similarity-based RAG may return irrelevant knowledge and negatively affect diagnosis (e.g., misleading diagnosis under the guidance of irrelevant ones).

Therefore, we need to develop an experience-enhanced LLM framework that can (1) systematically integrate O&M experience without missing the original relations, and (2) support new root causes and recovery solutions for effective and extensible database O&M.

3 DBAIOPS OVERVIEW

DBAIOps is the first database O&M system that integrates reasoning LLMs with knowledge graphs to deliver expert-level diagnosis. In contrast to existing LLM-based methods, which often retrieve fragmented information [51] and produce generic results [7, 38], DBAIOps proposes a structured graph model that captures diverse diagnosis information and dynamically retrieves anomaly-related information for LLMs through five innovative components.

Architecture. DBAIOps is composed of five key components (Figure 2). ❶ `ExperienceGraph` encodes expert O&M experience into a heterogeneous graph model, where vertices denote O&M information (e.g., metrics), and edges capture relations involved in multi-step anomaly analysis; ❷ `AnomalyModel` performs anomaly detection (using equations derived from metric-anomaly correlation analysis) based on the fine-grained metric hierarchy (e.g., raw data → aggregated data) and descriptive anomaly metadata (e.g., symptom illustration); ❸ `AnomalyProcessor` extracts relevant anomaly analysis information by leveraging both the `AnomalyModel` outputs and implicitly correlated metrics; ❹ `ExperienceRetriever` automatically explores anomaly analysis paths through a two-stage graph exploration strategy (i.e., proximity-based graph expansion and statistical graph clipping) to accumulate relevant experience;

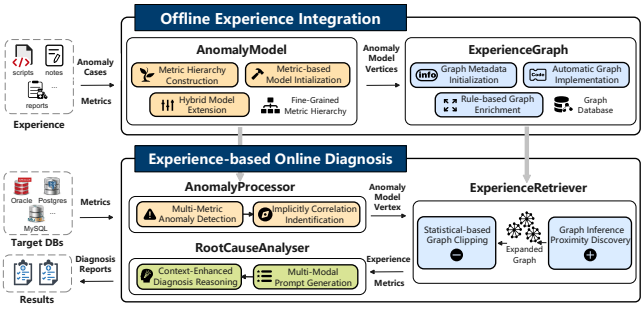


Figure 2: System Overview of DBAIOps.

❺ RootCauseAnalyser employs reasoning LLMs to simulate DBA-style diagnosis (producing accurate and actionable reports) based on the graph-augmented experience.

Note that, with the above components (e.g., multi-metric correlation, graph-based O&M experience encoding), DBAIOps operates effectively using general reasoning LLMs [4, 22] (see Section 7), eliminating the need for specialized LLM training.

Offline Stage. Given historical anomaly cases and metrics, DBAIOps first extracts raw vertices and edges using LLMs, and anchors each case with an AnomalyModel vertex (built from multi-metric relations). Then human experts validate and enrich the graph by linking relevant metrics and tools, producing a structured ExperienceGraph. The graph is stored in Neo4j via Cypher queries, with database labels added for cross-system reuse. Finally, the graph is enriched by connecting vertices that share identical or synonymous tags to improve connectivity and diagnosis coverage.

Online Stage. Upon receiving diagnosis requests, the AnomalyProcessor takes anomaly-related metrics (numeric) as input and outputs triggered vertices of AnomalyModel with alert descriptions. The ExperienceRetriever then processes these vertices and metrics to output diagnosis information from graph paths. Finally, the RootCauseAnalyser takes this retrieved information as input and generates comprehensive diagnosis reports (text) identifying root causes and recovery solutions.

DBAIOps can generalize to unseen root causes via three mechanisms: **(1) Tag-Anchored Subgraph Linking:** DBAIOps connects subgraphs with different diagnosis experience using heuristic strategies (e.g., connecting vertices with the same tag values). It then performs two-stage graph exploration to dynamically retrieve relevant diagnosis information for any input anomaly (Section 6.1); **(2) Two-Stage Graph Exploration:** After retrieving diagnosis information from the graph, DBAIOps exploits the reasoning capabilities of LLMs for graph-augmented diagnosis. It utilizes structured prompts to clearly organize different information, enabling it to infer unseen anomalies (Section 6.2); **(3) Incremental Experience Integration:** By inputting basic information of new experience, DBAIOps maps information like tools and descriptions into vertices and their relations into edges, and automatically generates a Cypher query to incrementally add these vertices and edges into the original experience graph (Section 4.2).

Additionally, it is worth noting that DBAIOps incurs negligible overhead on the databases being diagnosed. It operates on physically separate machines and retrieves runtime metrics through non-intrusive system queries (e.g., for pre-aggregated metrics in Oracle). These metrics are essential to enterprise-grade monitoring and incur less than 2% of the database's total CPU utilization.

Table 2: Vertex and Edge Types in O&M Graph Model.

Type	Description
Anomaly Model Vertex	Specific anomaly scenario defined by monitoring rules or patterns.
Metric Vertex	Quantitative data from database reflecting its runtime status.
Experience Vertex	Diagnosis knowledge, including context, root causes, and solutions.
Tag Vertex	Semantic categories that group related vertices.
Tool Vertex	Executable scripts with specific diagnosis steps.
Auxiliary Vertex	Supplemental attributes of metrics, such as sampling frequency.
Containment Edge	Inclusion of a vertex within an Anomaly Model or Metric vertex.
Relevance Edge	Classification between a vertex and a Tag vertex.
Diagnosis Edge	Utilization of a Metric or Tool vertex by a vertex during diagnosis.
Synonym Edge	Semantic equivalence between two differently expressed Tag vertices.
Association Edge	General logical connection for custom vertex relationships.

4 GRAPH MODEL FOR O&M EXPERIENCE CHARACTERIZATION

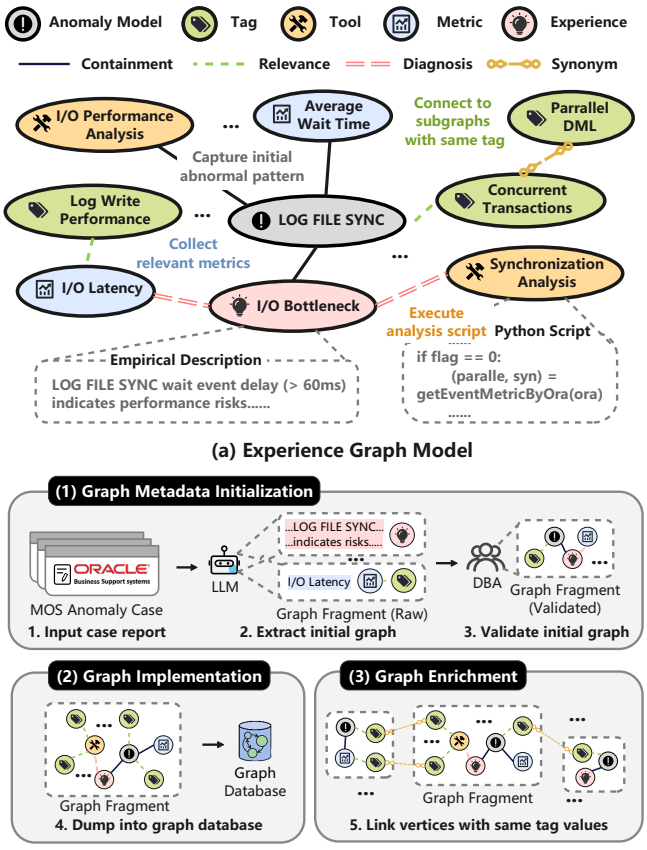
While extensive O&M knowledge exists in technical notes [42, 43, 45, 46], and resolved anomaly case reports [1–3, 6], existing methods fail to effectively capture and represent this diverse diagnosis information for automated analysis [38, 51]. To bridge this gap, we propose ExperienceGraph, a heterogeneous graph model that systematically structures database O&M expertise. Unlike rule-based systems with numeric thresholds or ML methods with simplified causal models, DBAIOps utilizes a semi-automatic construction algorithm to transform unstructured textual experience into executable graph paths, supporting over 25 database systems.

4.1 Experience Graph Model

To integrate O&M experience, existing approaches either rely on rules with predefined numeric metric thresholds [24, 40] or basic RAG strategies where LLMs diagnose by loosely connected document chunks [20, 38, 47]. They fail to capture complex relations that require considering heterogeneous information in O&M experience. To address this, DBAIOps proposes the first O&M-specific heterogeneous graph model (ExperienceGraph) that can be easily utilized by both LLMs and human DBAs.

Formally, we design ExperienceGraph as a directed heterogeneous graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of directed edges. As shown in Table 2, DBAIOps includes six vertex types and five edge types.

Vertex Modeling. The vertices encapsulate essential diagnosis information from diverse sources. *(1) Anomaly Model* vertices serve as the entry points, initiating graph traversal when specific patterns are detected; *(2) Metric* vertices provide quantitative evidence that measures anomalies through statistical indicators; *(3) Experience* vertices encode expert knowledge with detailed anomaly explanations, analysis steps, and recovery solutions; *(4) Tool* vertices automate data collection and in-depth investigation via executable scripts; *(5) Tag* vertices cluster vertices with similar semantics into



(a) Experience Graph Model

(b) Graph Model Construction

Figure 3: Example O&M Graph Model in DBAIOps.

groups; *(6) Auxiliary* vertices enrich metric interpretation with contextual attributes like collection frequency and percentiles.

Edge Modeling. The edges define pathways that guide graph traversal to accumulate diagnosis information across vertices. *(1) Containment* edges organize related components by placing them inside anomaly models or metrics; *(2) Relevance* edges create category links by connecting vertices to their semantic tags; *(3) Diagnosis* edges connect analysis steps to the specific metrics and tools needed for execution; *(4) Synonym* edges combine equivalent terms that have the same meaning in the graph; *(5) Association* edges create flexible links between vertices to help collect related information.

Example 4.1. The topological structure of `ExperienceGraph`*, including the log file synchronization anomaly (detecting excessive wait times during log file synchronization in Oracle databases),* is displayed in Figure 3. A central *Anomaly Model* vertex ("LOG FILE SYNC") acts as the entry point, linking via a *Containment* edge to an Experience vertex of the I/O related diagnosis knowledge ("Empirical Descriptions"). This *Experience* vertex uses *Diagnosis* edges to connect to *Metric* vertex ("I/O Latency") and Tool vertex ("Python scripts of synchronization analysis"). Relevance edges link the *Anomaly Model* and *Metric* vertices to *Tag* vertices, grouping them into meaningful categories ("Concurrent Transactions"). Synonym edges associate equivalent tag values ("Parallel DML"), enhancing graph connectivity for diagnosis knowledge retrieval.

Algorithm 1: Experience Graph Model Construction

Input: Anomaly Cases $\mathcal{A}$, Database Metrics $\mathcal{M}$
Output: Experience Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

```
/* 1. Graph Metadata Initialization */
Initialize G ← (V ← ∅, E ← ∅)
foreach anomaly case a_i ∈ A do
    V_raw, E_raw ← LLM.ExtractFromDoc(a_i)
    v_anomaly ← CreateAnomalyModel(a_i, M)
    V ← V ∪ {v_anomaly}
    V_valid, E_valid ← DBA.Validate(a_i, v_anomaly, V_raw, E_raw)
    V ← V ∪ V_valid, E ← E ∪ E_valid
/* 2. Graph Implementation */
DB ← DB.execute(MapToCypher(G))
/* 3. Graph Enrichment */
foreach vertex pair (v_i, v_j) ∈ V × V do
    if SameTag(v_i, v_j) or SynonymTag(v_i, v_j) then
        E ← E ∪ {(v_i, v_j)}
return G
```

4.2 Graph Model Construction

Given the vast and complex O&M experience pieces, reducing the manual effort required for graph model construction is extremely important [28, 35, 41]. However, existing methods mainly utilize basic ML (e.g., CauseRank [33]) to automatically add graph edges, which rely on simple causal assumptions between vertices and may fail to uncover implicit relations. Instead, we introduce a semi-automatic construction method. This method leverages LLM to generate initial graph sketches of key vertices, which are then manually validated and expanded into a unified graph.

As outlined in Algorithm 1, `DBAIOps` systematically constructs the `ExperienceGraph` in three steps, utilizing diverse sources including anomaly reports such as 15,000 official MOS (My Oracle Support) documents for Oracle databases [6].

• **(1) Graph Metadata Initialization:** For each anomaly case $a_i \in \mathcal{A}$, we employ LLMs to extract initial graph vertices $\mathcal{V}_{\text{raw}}$ (Metric, Experience, and Tag vertices) and edges $\mathcal{E}_{\text{raw}}$ from entities in the documentation, with graph definitions in the prompt. Multiple LLM outputs are aggregated via majority voting for robustness. We then construct an *Anomaly Model* vertex v_{anomaly} using statistical multi-metric relations (Section 5.1) as the central diagnosis entry. Database experts validate the graph starting from this vertex, then organize remaining elements around it: (a) connecting relevant *Metric* and *Experience* vertices to v_{anomaly}, (b) adding Tool vertices with Python scripts when available to offer multiple diagnosis paths, and (c) categorizing each vertex with *Tag* vertices. The final validated vertices $\mathcal{V}_{\text{valid}}$ and edges $\mathcal{E}_{\text{valid}}$ construct graph $\mathcal{G}$.

• **(2) Graph Implementation:** The validated graph $\mathcal{G}$ is transformed into executable Cypher queries via *MapToCypher*($\mathcal{G}$) and stored in Neo4j, enabling efficient graph traversal during online diagnosis. To facilitate knowledge sharing through common tags while eliminating redundant construction efforts, we build a unified

graph for all 25 supported databases by further tagging each vertex with its corresponding database identifier.

• **(3) Graph Enrichment:** We automatically enhance graph connectivity using heuristic strategies by adding edges between vertices that share identical tags or synonymous tags as below.

$$\mathcal{E} \leftarrow \mathcal{E} \cup \{(v_i, v_j) \mid SameTag(v_i, v_j) \vee SynonymTag(v_i, v_j)\}$$

It associates relevant subgraphs across different $v_{anomaly}$ and significantly expands the graph's diagnosis coverage (e.g., generating over 300,000 edges from dozens of Oracle anomaly models).

Example 4.2. As illustrated in Figure 3, the construction of the graph for log file synchronization anomaly proceeds systematically through three stages. Initially, DBAIOps utilizes LLMs to extract raw graph entities from My Oracle Support (MOS) documents [6], including Experience vertices (e.g., "...indicates risks...") and Metric vertices ("I/O Latency"). Database experts then validate the extracted raw graph to construct a central *Anomaly Model* vertex with the trigger condition "wait time > 10ms" and confirm its relations to other vertices. This validated graph is transformed into executable Cypher queries and persisted in the Neo4j graph database to support efficient online retrieval. Finally, the graph is automatically enriched by creating *Synonym* edges between vertices with the semantically equivalent tag values, effectively linking the LOG FILE SYNC anomaly model with other I/O-related anomalies.

In this way, we construct a graph that incorporates DBA expertise in managing 5,000+ databases over the past 10 years, along with 2,000+ historical anomaly cases, and supports new anomalies via automatic graph exploration (see Section 6.1). As shown in Table 3, the number of *Anomaly Model* vertices in the graph varies significantly across different databases, with Oracle, MySQL, and PostgreSQL equal to 82, 91, and 36, respectively. It suggests that DBAIOps provides more granular anomaly detection for these databases. Figure 4 further shows that the top five model categories (SQL Workload, Memory, Logical I/O, Physical I/O, and Active Session) dominate the overall distribution, corresponding to the key performance domains routinely monitored during database anomaly detections.

5 CORRELATION-AWARE ANOMALY MODEL

While existing methods effectively detect anomalies in individual metrics [24, 40], they overlook complex patterns arising from metric correlations. To address this limitation, we introduce a Correlation-Aware Anomaly Model that identifies implicit relations among metrics (e.g., coordinated spikes in log sync delay and parallel write time). Unlike traditional single-metric approaches, our model employs a fine-grained metric hierarchy and statistical multi-metric analysis to uncover complex anomalies.

5.1 Multi-Metric Anomaly Detection

Metrics serve as primary factor to facilitate effective database O&M. However, there exists a high volume of metrics from diverse monitoring sources, and these metrics need to be further processed to derive essential information (e.g., trend changes). To address this issue, DBAIOps first constructs a unified metric hierarchy, and then performs statistical multi-metric correlation analysis to automatically derive effective anomaly detection equations.

Table 3: Statistics of anomaly models in DBAIOps. Variations in numbers arise from differences in available resources.

Database	Oracle	DB2	SQL Server	MySQL	PostgreSQL	OceanBase	GaussDB
Metric	550	927	314	316	645	963	658
Diagnosis Tool	396	10	124	215	148	98	151
Anomaly Model	82	7	25	91	36	34	85

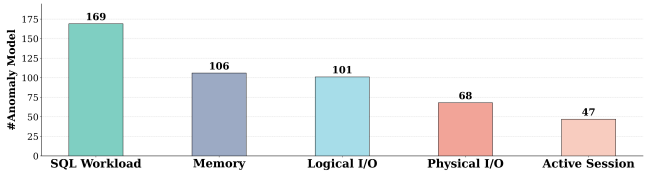


Figure 4: Top-5 Anomaly Model Type in DBAIOps.

Fine-Grained Metric Hierarchy. To provide a comprehensive view of database status, DBAIOps introduces a carefully designed three-level metric hierarchy that organizes thousands of metrics into a tree structure from general domains to specific measurements. This hierarchy is constructed semi-automatically by combining domain expertise from database documentation with metrics collected from real-world deployments. Specifically, the hierarchy comprises *(1) top-level categories* representing broad functional areas (e.g., Performance, Configuration), which are decomposed into *(2) mid-level subcategories* defining specific technical components (e.g., I/O, CPU, Memory under Performance). These subcategories contain *(3) leaf-level metrics* that serve as measurable indicators for anomaly detection, such as *LOG_FILE_SYNC wait time* and *log file parallel write* under the *I/O → Redo Log* path.

These metrics are initially derived as raw data from external tools (e.g., Prometheus [5]), retaining only essential details like category IDs and error messages. Subsequently, additional statistical data (e.g., incremental differences, rolling averages, and histograms) are computed lazily only when needed for diagnosis. For example, DBAIOps collects both long-interval performance statistics from Oracle's Automatic Workload Repository (AWR), such as 30-minute snapshots of CPU and I/O metrics, and short-interval session data from Active Session History (ASH), which captures session activity every second for real-time performance monitoring.

Metric-to-Anomaly Correlation. To effectively capture metric-to-anomaly relation, DBAIOps develops a collection of anomaly models. Each model captures a specific database anomaly based on distinct multi-metric patterns or longitudinal single-metric comparisons. Unlike typical threshold-based methods that detect excessive anomalies with limited accuracy, the anomaly models (1) leverage both established O&M experience and analysis over multiple metrics, and (2) are generated from basic elements. As shown in Function 1, the construction involves four key steps.

• **(1) Relevant Metric Selection:** Given an anomaly case a_i, we first employ multiple LLMs to extract an initial natural language description $Desc_{a_i}$ and a candidate set of metrics $\mathcal{M}_{a_i}$ potentially associated with the anomaly. To improve robustness, we aggregate outputs from different LLMs via a majority voting mechanism facilitated by LLM-as-a-Judge techniques [26]. These preliminary results are subsequently validated by database experts with extensive maintenance experience, yielding a refined description $Desc_{a_i}^{valid}$ and a verified set of causally relevant metrics $\mathcal{M}_{a_i}^{valid} \subseteq \mathcal{M}_{a_i}$.

• **(2) Selected Metric Transformation:** Each validated metric $m \in \mathcal{M}_{a_i}^{valid}$ is transformed via a function Φ to emphasize specific anomaly patterns. Let $\mathbf{m} = [m_1, m_2, \ldots, m_T]$ denote the time series of a metric over T time steps, and $\mathbf{r} = [r_1, r_2, \ldots, r_T]$ be a reference time series. The transformation Φ is defined as:

$$\Phi(\mathbf{m}) = \begin{cases} \Phi_{\text{iden}}(\mathbf{m}) = \mathbf{m}, & \textit{// Identity} \\ \Phi_{\text{shape}}(\mathbf{m}, \mathbf{r}) = DTW(\mathbf{m}, \mathbf{r}), & \textit{// Shape Similarity} \\ \Phi_{\text{cate}}(\mathbf{m}) = Classify(\mathbf{m}) & \textit{// Categorical} \end{cases}$$

where (1) Φ_{iden} returns the original time series values; (2) Φ_{shape} measures the shape similarity between the time series $\mathbf{m}$ and a reference series $\mathbf{r}$ using Dynamic Time Warping (DTW), capturing time-series waveform similarity under temporal distortion; (3) Φ_{cate} maps the metric to a discrete set of categorical labels based on expert-defined rules, enabling multi-level anomaly characterization.

The set of transformed metrics is $\mathcal{M}' = \{\Phi(m) \mid m \in \mathcal{M}_{a_i}^{valid}\}$.

• **(3) Detect Function Construction:** A disjunctive normal form (DNF) detection function f_{detect} is constructed from the transformed metrics. This function includes anomaly-detection equations that use configurable logical expressions over system metrics and statistical patterns (trends). For each $m' \in \mathcal{M}'$, an expression $condition_i$ is defined, typically comparing m' to a category or a threshold θ. These thresholds are initially set by domain experts and later refined automatically via the Anomaly Detector Function (ADF) (introduced in Section 6.1). Conditions may also be composite, combining multiple metrics via conjunctions (AND) and disjunctions (OR). The overall detection function is defined as:

$$f_{detect} = condition_1 \vee condition_2 \vee \ldots \vee condition_n$$

An anomaly is triggered if at least one condition in f_{detect} is true.

• **(4) Anomaly Model Construction:** The final anomaly model is encapsulated as a vertex: $v_{\text{anomaly}} = \langle Desc_{a_i}^{valid},\ \mathcal{M}',\ f_{detect} \rangle$. It integrates the validated anomaly description, the transformed metrics, and the DNF detection logic, forming a complete and executable anomaly representation within the graph model.

Example 5.1. The construction of the LOG_FILE_SYNC *anomaly model begins with multiple LLMs for an initial description ("log file sync wait delays") and candidate metrics. Through majority voting and expert validation, it is refined to a precise description and the verified metric ("average_wait_time"). It undergoes transformation through Φ_{iden} to preserve raw values and Φ_{cate} to classify 10-minute trends into categories like sharp rise. These transformed metrics form a disjunctive normal form detection function $f_{detect} = (\mathbf{METRIC}_{raw} > 60ms) \vee \big((\mathbf{Trend}_{10min} = sharp\ rise) \wedge (\mathbf{METRIC}_{raw} > 6ms)\big)$, where thresholds are initially set by experts and later refined automatically. The complete model is encapsulated as $v_{trigger} = \langle Desc_{a_i}^{valid},\ \{\mathbf{METRIC}_{raw}, \mathbf{Trend}_{10min}\},\ f_{detect} \rangle$, creating an executable anomaly representation that integrates validated description, transformed metrics, and detection logic.*

5.2 Implicitly-Correlated Metric Identification

Many critical database anomalies arise from complex interactions between seemingly unrelated metrics, not from individual threshold violations [47, 51]. These hidden relationships explain performance issues that conventional monitoring misses. To systematically capture these subtle connections, DBAIOps employs three mechanisms to reveal implicit correlations across different metrics.

• **(1) Subgraph Connection via Tags:** DBAIOps utilizes *Tag* vertices in $\mathcal{G}$ to automatocally link disconnected subgraphs through the edge creation function $SameTag(v_i, v_j)$ and $SynonymTag(v_i, v_j)$. For instance, it connects "Physical Read" and "Disk Read" vertices by semantic equivalence, dynamically forming implicit correlation paths from different v_{anomaly} that were not explicitly predefined.

• **(2) Time Series Similarity Analysis:** DBAIOps detects synchronized fluctuation patterns, uncovering hidden dependencies even between metrics from different domains that may seem unrelated. It applies dynamic time warping (DTW) [36] for waveform similarity and cross-correlation to determine which metric leads or lags another. For example, it reveals connections between "Log file sync wait time" and "Disk I/O latency" when their patterns align through Φ_{shape}, uncovering previously overlooked relationships.

• **(3) Two-Stage Graph Exploration:** DBAIOps employs a systematic discovery process that first expands the search scope by traversing neighboring vertices in the graph, then filters out irrelevant metrics using statistical analysis to retain only those with significant abnormal patterns as valid implicit correlations. It enhances the diagnosis coverage as detailed in the following section.

Function 1: CreateAnomalyModel(a_i, $\mathcal{M}$)

```
/* 1. Relevant Metric Selection */
Desc_{a_i}, M_{a_i} ← LLM.ExtractFromDoc(a_i)
Desc^{valid}_{a_i}, M^{valid}_{a_i} ← DBA.Validate(a_i, Desc_{a_i}, M_{a_i})
/* 2. Selected Metric Transformation */
foreach metric m_i ∈ M^{valid}_{a_i} do
    m'_i ← Φ(m_i)        ▷ identity/shape similarity/categorical
    M' ← M' ∪ {m'_i}
/* 3. Detection Function Construction */
f_detect ← ∅                          ▷ Initialize DNF expression
foreach metric m'_i ∈ M' do
    condition_i ← DBA.SetCondition(m'_i)
    f_detect ← f_detect ∨ condition_i   ▷ OR DNF combination
/* 4. Anomaly Model Construction */
v_anomaly ← CreateVertex(Desc_{a_i}, M', f_detect)
return v_anomaly
```

6 SCENARIO-AWARE ANOMALY DIAGNOSIS

While traditional methods usually treat anomalies independently, they often fail to capture relevant issues where one anomaly triggers or worsens another, leading to incomplete diagnoses or false positives that require expert interpretation [37, 38]. To overcome these limitations, we propose Scenario-Aware Diagnosis that combines a two-stage graph exploration strategy with graph-augmented LLM reasoning. Unlike static rule-based systems with limited flexibility or standard LLMs prone to hallucinations, DBAIOps dynamically identifies diagnosis paths during graph exploration and uses

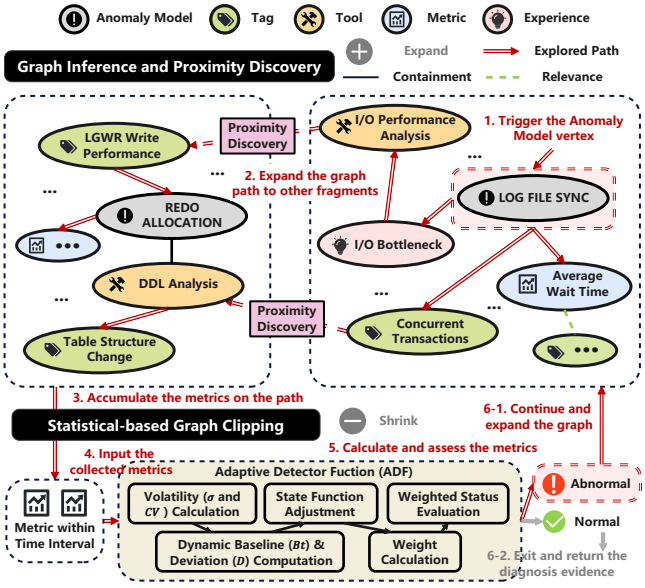


Figure 5: Two-Stage Graph Exploration in DBAIOps.

structured prompts to guide LLMs, ensuring the generated reports contain accurate root causes and recovery solutions.

6.1 Two-Stage Graph Exploration

In real-world scenarios, anomalies are rarely isolated, where a performance issue in one anomaly model may simultaneously trigger or worsen issues in another. However, different anomaly models such as those related to LOG_FILE_SYNC and REDO_ALLOCATION may appear only loosely connected in the initialized graph, which share sparse and fragmented experience (e.g., concurrency-related wait events). To this end, we propose an automatic graph exploration mechanism to dynamically discover and connect such related experience fragments across different anomaly models. As shown in Figure 5, this mechanism comprises two main stages.

• **(1) Graph Inference and Proximity Discovery:** Given the entire graph $\mathcal{G}$, DBAIOps initiates graph exploration through inference using graph query language (Cypher). Starting from *Anomaly Model* vertices v_{anomaly} where detection functions evaluate to true, it traverses connected vertices and edges to collect and aggregate relevant diagnosis information. The exploration follows *Relevance* edges to Metric vertices for statistical analysis and *Containment* edges to *Experience* vertices for diagnosis knowledge collection. Moreover, *Synonym* edges enhance the graph exploration by connecting semantically equivalent *Tag* vertices across different subgraphs, handling complex root causes by associating fragmented experiences. Through iterations within k hops, it yields an explored graph $\mathcal{G}'$ with increased edge density ($|\mathcal{E}'| > |\mathcal{E}|$), forming a more interconnected structure for complex anomalies.

• **(2) Statistical-based Graph Clipping.** After metrics over the explored graph $\mathcal{G}'$ are obtained, DBAIOps proposes Adaptive Detector Function (ADF) to assess whether collected metrics exhibit abnormal patterns and clip the graph accordingly. For a metric sequence $\mathbf{m} = [m_1, m_2, \ldots, m_T]$ over T steps, it operates in five steps.

Step 1. Volatility Calculation. We first calculate the standard deviation $\sigma(\mathbf{m})$ to measure fluctuation amplitude in the metric sequence. Moreover, we compute the coefficient $C_V = \rho_V / \rho_R$, where ρ_V represents the autocorrelation of volatility patterns and ρ_R represents random volatility autocorrelation, to quantify the persistence of fluctuations over time.

Step 2. Dynamic Baseline (B_t) and Deviation (D) Computation. We derives a dynamic baseline B_t for each time interval t. This baseline is updated hourly across different databases to maintain adaptiveness. The deviation is then calculated as $D = |m_t - B_t|$. By design, B_t can incorporate parameterized factors that capture known operational patterns.

Step 3. State Function Adjustment. We introduce a state function $F_{\text{state}}(m_t, B_t)$ which classifies how close m_t is to the baseline:

$$F_{\text{state}}(m_t, B_t) = \begin{cases} 1 - \frac{D}{\sigma}, & \text{if } m_t \text{ is near } B_t \\ \frac{D}{\sigma}, & \text{otherwise} \end{cases}$$

Smaller deviations from B_t yield a larger value under the first case; conversely, large divergences indicate potential anomalies.

Step 4. Weight Calculation. We compute the volatility weight w_1 dynamically based on σ and a threshold θ: $w_1 = \frac{\sigma}{\sigma+\theta}$, $\quad w_2 = 1 - w_1$. When $\sigma > \theta$, DBAIOps assigns more weight to volatility, signifying that larger fluctuations in the metric are more relevant.

Step 5. Weighted Status Evaluation. We compute the final anomaly score $S = w_1 \cdot \sigma(\mathbf{m}) + w_2 \cdot F_{\text{state}}(m_t, B_t)$. The graph exploration employs clipping based on this score: if S exceeds the environment threshold, the metric is marked as abnormal and the graph exploration continues via relevant edges; otherwise, it terminates at the current vertex, ensuring efficient traversal over anomaly regions.

Example 6.1. As illustrated in Figure 5, the graph exploration for the log file synchronization anomaly is conducted in two key steps. First, during **Graph Inference and Proximity Discovery**, DBAIOps identifies the *LOG FILE SYNC* vertex as the starting point and executes Cypher queries to traverse relevant edges, expanding the search to uncover hidden connections such as the *REDO ALLOCATION* anomaly model via the same tag values. Second, in **Statistical-based Graph Clipping**, DBAIOps validates and filters out these expanded vertices with normal patterns. For instance, analyzing the metric (*I/O Latency*) with a dynamic baseline $B_t \approx 15$ and a current deviation $D \approx 43$, it calculates that the anomaly score exceeds the threshold ($\sigma > \theta$); consequently, the relevant *REDO ALLOCATION* subgraph is retained and merged, while statistically normal branches are pruned. In this way, we can explore the graph to extract all relevant information for diagnosis.

6.2 Graph-Augmented LLM Diagnosis

With the paths explored from the graph, several challenges remain for accurate anomaly diagnosis. First, there may be false positives, such as vertices that appear relevant but do not accurately reflect the root cause. Second, experience within those vertices can be incomplete or difficult for general users to interpret. To this end, DBAIOps proposes a prompt-based strategy that guides the reasoning LLM to analyze the experience paths and generate clear, actionable diagnosis reports that include both the identified root causes and corresponding recovery solutions.

Table 4: Common Root Causes Observed in Real-World Usage Across the Four Database Systems.

Database	HIGH DATA SELECT	LOW REDO FILE SIZE	LOW REDO GROUP COUNT	LOG BUFFER SETTING NOT ENOUGH	TABLE INITTRANS NOT ENOUGH	BUFFER BUSY WAIT	ENQ LOCK WAIT	LATCH WAIT	HIGH MEMORY USAGE	HIGH CPU USAGE	BGWRITER PARAMETER PROBLEM	SHARED BUFFER NOT ENGHOU	CHECKPOINT PARAMETER PROBLEM	WAL PARAMETER PROBLEM	TABLE DEAD TUPLE	INDEX PROBLEM	STATISTICS EXPIRED
Oracle	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
DM8	✓	×	×	✓	×	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
Mysql	✓	×	×	×	×	✓	✓	✓	✓	✓	×	×	×	×	×	×	×
PostgreSQL	×	×	×	×	×	×	×	×	×	×	✓	✓	✓	✓	✓	✓	✓

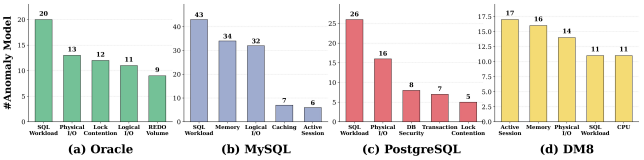


Figure 6: Top-5 Anomaly Model Type vs. Database in DBAIOps.

To address false positives and incomplete coverage, DBAIOps provides LLM with (1) extensive textual analysis experience collected during graph traversal and (2) a collection of accumulated metrics and execution details (e.g., logs, historical performance baselines). When LLM crafts a diagnosis report, it not only refers the *Triggered Vertex* in the graph but also traces relevant edges to other anomalies. It then contextualizes these findings by describing how each anomaly interacts within the broader environment (e.g., "*The concurrency waits grew after I/O latencies exceeded 30 ms, indicating shared resource contention.*"). This synergy between structured graph data and open-ended generative reasoning allows DBAIOps to produce more thorough and comprehensible diagnosis.

Prompting LLM for Structured Report Generation. A core design feature of DBAIOps is the structured prompts to guide the LLM in generating diagnosis reports that are both actionable and easy to understand, which are composed of the following components: Given an observed anomaly, we concatenate five necessary components into a ***prompt*** $= \langle \mathcal{S}^a, \mathcal{S}^l, \mathcal{S}^m, \mathcal{S}^e, \mathcal{S}^o \rangle$, where:

- $\mathcal{S}^a$ (Anomaly) specifies the symptom descriptions (e.g., "CPU usage spiked to 95% at 16:00 on 2023-10-05");
- $\mathcal{S}^l$ (Condition) encodes the anomaly detection condition (e.g., "exceeds 90% for >5 min");
- $\mathcal{S}^m$ (Metrics) records key statistics, e.g., metric name (*CPU Usage, %*), time range (*1684600070–1684603670*), and threshold (*90%*);
- $\mathcal{S}^e$ (Experience) provides contextual facts such as normal load (10 k req/min) and recent maintenance (kernel update 2023-10-04);
- $\mathcal{S}^o$ (Output) prescribes the desired report *components*.

We supply the ***prompt*** for LLM to generate diagnosis reports that includes: (1) Anomaly Validation: determines whether the reported anomaly requires further investigation; (2) Root Cause Analysis: identifying up to five likely causes supported by metrics, logs, or known fault signatures; (3) Recover Solution: suggesting technical adjustments such as configuration changes or query optimizations; (4) Summary: providing a concise assessment of overall system health; (5) SQL Context: including relevant SQL statements or execution plans if the issue involves database operations.

7 EXPERIMENTS

7.1 Experiment Setup

Databases. We test four database systems (i.e., Oracle [15], MySQL [14], PostgreSQL [17], and DM8 [11]). The metrics and logs are collected by adapted tools like Prometheus [5].

Anomalies. Table 4 lists the root causes of the tested anomalies for four databases: Oracle, MySQL, PostgreSQL, and DM8. The total number of tested scenarios is 178, 114, 127, and 139, respectively. These anomalies are grouped into five main categories: (1) Log Synchronization and Management Issues, (2) Resource Contention and Concurrency Issues, (3) SQL Optimization Issues, (4) Hardware and System Resource Bottlenecks, and (5) Database Write Performance Issues. The distribution of fine-grained anomaly model types in DBAIOps across the four assessed databases is shown in Figure 6. Moreover, we ensure the tested anomalies are distinct from those in the graph model; specifically, the graph does not explicitly contain identical root causes or solutions as the test cases. The ground-truth results are derived from expert DBAs' diagnosis reports.

Evaluation Methods. We evaluate methods in Table 1 that can generate complete diagnosis reports with detailed analysis steps. (1) Rule-based Tool + DBA: Utilize pre-defined tools to generate specialized reports, requiring further analysis of an expert DBA to overcome the limitations that traditional methods in Table 1 can not generate comprehensive diagnosis reports; (2) LLM-Only: We evaluate typical LLMs (DeepSeek-R1-32B and DeepSeek-R1-671B) by directly providing them with the necessary diagnosis information (e.g., the monitoring metrics) as the input. (3) ChatDBA [7]: RAG-based approach that incorporates a tree-based structure to support diagnosis over MySQL and PostgreSQL; (4) D-Bot [51]: State-of-the-art LLM-based method that utilizes multi-agent framework (equipped with tree-of-thought algorithm) for diagnosis over PostgreSQL [51]; (5) DBAIOps: We provided the metric data and textual knowledge description from the anomaly model and O&M knowledge graph, with different LLMs as the underlying backbones (i.e., DeepSeek V3 [22], DeepSeek-R1-32B, and DeepSeek-R1-671B [23]);

Evaluation Metrics. We adopt four metrics. First, we utilize two basic metrics (i.e., *Precision* and *F1 Score*) to quantify the effectiveness of different methods in root cause identification. Second, we use the metric *Accuracy* proposed in [51] to quantify the effectiveness of root cause analysis, while also accounting for the presence of a wrong root cause. Finally, we adopt Human Evaluation Accuracy (HEval) to measure the overall diagnosis quality of different methods, strictly adhering to three human-assessed criteria (Root Cause Recall (30%), Theoretical Consistency (30%), and Evidence Authenticity (40%))[1].

Other Settings. The experimental setup includes the following key components: (1) LLM SERVER, utilizing the Ollama framework and equipped with an RTX 3090 GPU, running a 32B distilled model; (2) An operational knowledge graph, constructed based on the KYD Zhiyan platform; and (3) Data collection, performed using the DBAIOps community edition tool. These components collectively provide the necessary technical support for the experiment, ensuring efficient operation and accurate data analysis.

7.2 Overall Performance

Table 5 presents the overall diagnosis performance of different methods over anomalies across diverse database systems. We have the

[1] https://github.com/weAIDB/DBAIOps/blob/master/HEval_criteria.md

Table 5: Overall Diagnosis Performance of Different Methods over Anomalies across Four Database Systems (N/A denotes that diagnosis over the database is not supported by the corresponding method, e.g., D-Bot [51] only supports PostgreSQL).

Method		Oracle				MySQL				PostgreSQL				DM8			
		Precision	F1-Score	Accuracy	HEval	Precision	F1-Score	Accuracy	HEval	Precision	F1-Score	Accuracy	HEval	Precision	F1-Score	Accuracy	HEval
Traditional	Rule-based Tool + DBA	0.88	0.89	0.88	0.88	1.00	0.67	1.00	0.50	1.00	1.00	1.00	0.95	1.00	1.00	1.00	0.90
LLM Only	DeepSeek-R1 32B	0.68	0.70	0.65	0.52	0.84	0.91	0.71	0.85	0.10	0.13	0.83	0.05	0.74	0.72	0.01	0.63
	DeepSeek-R1 671B	0.77	0.83	0.75	0.78	0.67	0.80	0.56	0.70	0.75	0.86	0.63	0.75	0.60	0.60	0.73	0.45
LLM (RAG-based)	ChatDBA	N/A	N/A	N/A	N/A	0.50	0.60	0.45	0.65	0.63	0.56	0.59	0.40	N/A	N/A	N/A	N/A
LLM (Agent-based)	D-Bot (DeepSeek V3)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.50	0.40	0.45	0.35	N/A	N/A	N/A	N/A
	D-Bot (DeepSeek-R1 32B)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.33	0.33	0.27	0.50	N/A	N/A	N/A	N/A
	D-Bot (DeepSeek-R1 671B)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.40	0.36	0.34	0.35	N/A	N/A	N/A	N/A
DBAIOps (DeepSeek V3)		0.50	0.67	0.45	0.66	0.77	0.87	**1.00**	0.88	0.83	0.91	0.75	0.83	**1.00**	**1.00**	0.82	**0.95**
DBAIOps (DeepSeek-R1 32B)		0.94	0.88	0.93	0.87	**0.94**	**0.97**	**1.00**	0.95	**0.87**	**0.93**	**0.93**	0.85	**1.00**	0.95	**0.85**	0.90
DBAIOps (DeepSeek-R1 671B)		**1.00**	**0.95**	**1.00**	**0.91**	0.92	0.96	**1.00**	**0.98**	0.83	0.91	0.91	**0.88**	**1.00**	**1.00**	0.82	**0.95**

following observations. ❶ First, DBAIOps *achieves comparable diagnosis performance under different LLMs, with highest performance over the four database systems.* Specifically, DBAIOps (DeepSeek-R1 32B) and DBAIOps (DeepSeek-R1 671B) obtain the aggregated diagnosis performance of 0.92 and 0.94, which is 61.40% and 34.29% higher than the diagnosis performance of 0.57 by DeepSeek-R1 32B and 0.70 by DeepSeek-R1 671B, respectively. The underlying reason can be attributed to the fact that even with the necessary diagnosis information (e.g., the relevant metrics), LLM can only conclude the root causes based on their general knowledge rather than specific O&M experience in the graph model of DBAIOps. ❷ Second, DBAIOps *can outperform state-of-the-art LLM-based methods or even Rule-based Tool + DBA method, showcasing well-behaved generalization ability across scenarios.* Specifically, DBAIOps outperforms D-Bot and ChatDBA by over 37% and 45% in *HEval*. The average diagnosis performance of DBAIOps with different LLM arrives at 0.89 across database systems, which is comparable with the performance of 0.91 by Rule-based Tool + DBA. Moreover, DBAIOps obtains the average accuracy of 0.94, better than 0.79 of Rule-based Tool + DBA over MySQL anomalies. The underlying reason is that DBAs face challenges in processing large volumes of monitoring data within a limited time. They often rely on a small subset of signals, which can lead to incomplete or conflicting conclusions. For example, in *IO_Latency_MySQL_Critical* anomaly model, the tool generates 14 diagnosis items, making it difficult for DBAs to analyze all relevant data and accurately identify all root causes. In contrast, DBAIOps improves in two aspects: (1) DBAIOps provides LLM with more comprehensive O&M experience essential for accurate diagnosis (e.g., metric statistics and relevant knowledge points), some of which might be left out in the pertaining corpus of LLM; (2) DBAIOps carefully prompts them to reason over the provided information about the anomaly like DBA (e.g., analyze the provided metrics via relevant O&M experience) and exploits the generation capability of LLMs to produce customized diagnosis reports. Thus, DBAIOps can generate more comprehensive and user-friendly (i.e., easier to understand) diagnosis reports (More details in Section 7.4). ❸ Finally, DBAIOps *with the medium-sized reasoning model, i.e.,* DBAIOps *(DeepSeek-R1 32B) can achieve comparable diagnosis accuracy to a large-scale reasoning model, i.e.,* DBAIOps *(DeepSeek-R1 671B).* Specifically, the average diagnosis performance of DBAIOps (DeepSeek-R1 32B) arrives at 0.92 across database systems, comparable to the one of 0.94 by DBAIOps (DeepSeek-R1 671B). The reason is that DBAIOps provides useful information, including in the processed data from the combined use of an anomaly model and O&M knowledge graph, alleviating the difficulty of identifying the correct root causes. Therefore, a medium-sized reasoning model can perform well given the information in the processed data. ❹ Last, DBAIOps delivers consistently higher Human Evaluation Accuracy (*HEval*) across different anomalies. Specifically, DBAIOps outperforms Rule-based Tool + DBA and DeepSeek-R1 32B by 29.00% on the Oracle LOG SYNCHRONIZATION DELAY anomaly and by 48.00% on the PostgreSQL BACKEND PROCESS FLUSHES DIRTY PAGES anomaly. It highlights that DBAIOps 's integrated design (combining structured anomaly models, knowledge-graph-based reasoning, and evidence-bounded LLM analysis) yields substantially more reliable diagnoses than baseline approaches across heterogeneous anomaly types (see detailed analysis in Section 7.4).

Table 6: Diagnosis Performance of DBAIOps Variants.

Component		Oracle				MySQL			
Experience Graph Model		Precision	F1 Score	Accuracy	HEval	Precision	F1 Score	Accuracy	HEval
w/o Experience		0.60	0.50	0.56	0.50	0.60	0.55	0.56	0.20
w/o Tag		0.75	0.67	0.70	0.60	0.80	0.89	0.78	0.50
Correlation-Aware Anomaly Model		Precision	F1 Score	Accuracy	HEval	Precision	F1 Score	Accuracy	HEval
w/ Correlation		0.60	0.75	0.56	0.60	0.60	0.75	0.56	0.60
w/o Correlation		0.20	0.22	0.12	0.15	0.25	0.33	0.18	0.10
Two-Stage Graph Exploration		Precision	F1 Score	Accuracy	HEval	Precision	F1 Score	Accuracy	HEval
Graph Inference	ADF								
×	✓	0.67	0.80	0.63	0.60	0.40	0.44	0.34	0.30
✓	×	0.75	0.86	0.73	0.50	0.60	0.55	0.56	0.20
×	×	0.60	0.75	0.56	0.50	0.60	0.75	0.56	0.40
DBAIOps		**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

We further assess the detailed diagnosis performance of different methods across database anomalies. Figure 7 presents the results over four scenarios. We notice that DBAIOps *showcases stable diagnosis performance improvement of LLMs that present different effectiveness across anomalies.* Specifically, DBAIOps helps achieve promising diagnosis performance of 0.87 on average across different anomalies. In contrast, DeepSeek-R1 32B performs well with the average diagnosis performance of 0.59 over CPU Spikes and performs poorly with the average diagnosis performance of 0.12 over Excessive Dirty Page Writes. It suggests that solely relying on LLMs is ineffective for diagnosing diverse anomalies, emphasizing the need to integrate an O&M knowledge graph (as utilized in DBAIOps) to improve diagnosis accuracy.

7.3 Ablation Study

We experiment with 7 variants for the five DBAIOps components to evaluate their contributions to DBAIOps's effectiveness. **(1) Experience Graph Model (ExperienceGraph):** We assess (a) w/o

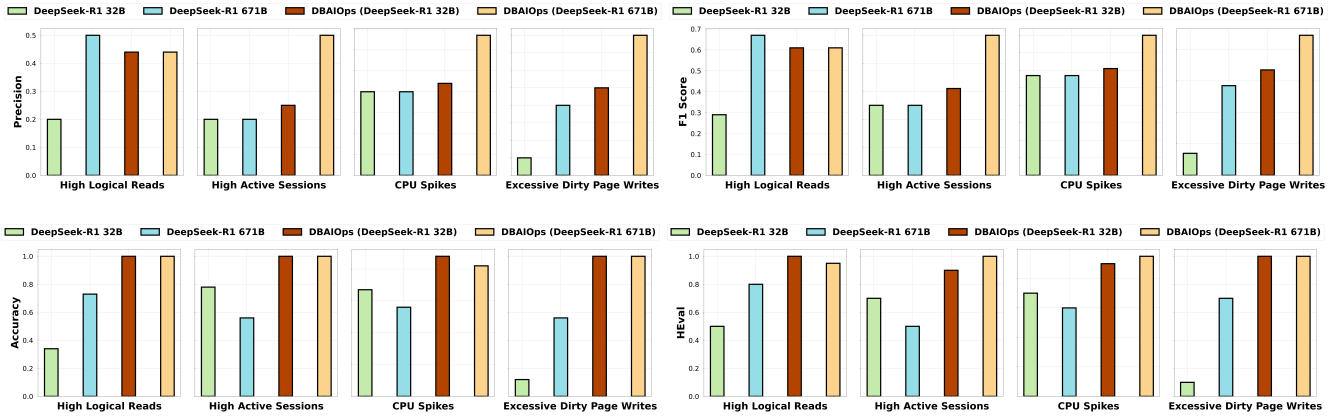


Figure 7: Distribution of Diagnosis Performance across Different Scenarios.

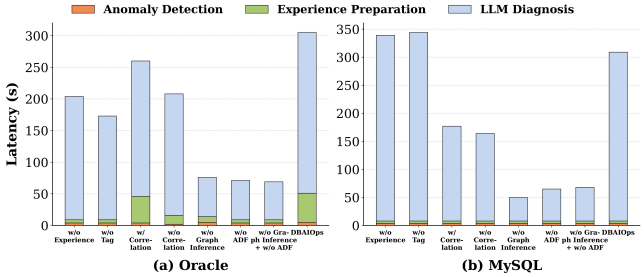


Figure 8: Latency Analysis of DBAIOps Variants.

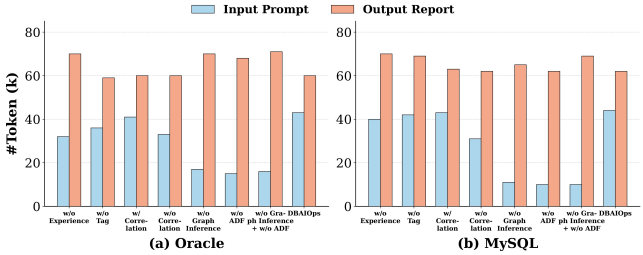


Figure 9: #Token of Input Prompt and Output Report.

Experience: Remove the *Experience* vertices important to instruct LLM diagnosis; (b) w/o Tag: Remove the *Tag* vertices utilized to enhance the subgraph connectivity. As shown in Table 6, both variants lead to a significant diagnosis performance drop with 39.00% on average. For example, in the Oracle memory usage anomaly, removing experience such as memory allocation settings results in a complete loss of diagnosis performance to 0.20 when relying solely on LLM. This highlights the importance of experience augmentation for accurate diagnosis, especially for LLMs with limited inherent knowledge; **(2) Correlation-Aware Anomaly Model (`AnomalyModel` & `AnomalyProcessor`):** We validate (a) w/ Correlation: Utilize anomaly models that infer overlapping root causes to trigger graph exploration for diagnosis; (b) w/o Correlation: Utilize anomaly models that infer completely different root causes. Table 6 displays that DBAIOps maintains reasonable performance with 0.63 on average when anomalies share the same root causes, but its performance significantly drops to 0.19 for irrelevant anomalies. This is due to the anomaly-trigger mechanism that focuses on specific anomaly patterns rather than producing generic conclusions. However, DBAIOps avoids false triggers through correlation-aware anomaly model, and contextual graph exploration, ensuring reliable anomaly detection in production; **(3) Two-Stage Graph Exploration LLM Reasoning (`ExperienceRetriever` & `RootCauseAnalyser`):** We verify: (a) No Graph Inference to collect diagnosis information from related subgraphs; (b) No ADF to identify and filter out normal metrics in the graph; (c) Neither Graph Inference nor ADF. Results in Table 6 demonstrate that removing graph exploration components causes significant performance degradation (i.e., 43.12% on average). For instance, in MySQL CPU anomaly diagnosis scenarios, the absence of the Adaptive Detector Function (ADF) prevents LLM from distinguishing between normal and abnormal metrics. Thus, it must rely solely on its inner knowledge, introducing randomness and hallucinations that severely compromise performance.

We further analyze the latency overhead of DBAIOps and its variants by breaking down and counting the time distribution over three stages: **(1) Anomaly Detection** to trigger the anomaly model for diagnosis; **(2) Experience Preparation** to retrieve and process the diagnosis information from two-stage graph exploration; **(3) LLM Diagnosis** to instruct LLMs to generate diagnosis reports with the collected diagnosis information. For the results in Figure 8, we observe that the majority of the time is spent on LLM Diagnosis, i.e., an average 88.47% (135.75s) on Oracle and MySQL anomalies. In contrast, Anomaly Detection occupies only a small proportion, i.e., 4.01% ($\leq$ 5s), on average, and works completely without LLMs. It indicates that DBAIOps can quickly identify anomalies without waiting for LLM processing, making it reliable for real-world use where fast detection is important. We further analyze the input prompt and output report sizes of DBAIOps and its variants in Figure 9. As displayed in Figure 9, the input prompt sizes decrease with less experience collected from the graph (i.e., 26.2k for variants and 43.5k for DBAIOps on average). In contrast, the output prompt sizes remain stable (i.e., 67.3k for variants and 61.0k for DBAIOps on average) because complete diagnosis reports are well-structured and include all the required components specified in the prompt (e.g., listing of root causes and detailed analysis).

7.4 Real-World Case Analysis

We assess the diagnosis reports generated by DBAIOps and baseline methods for two representative anomalies (i.e., LOG SYNCHRONIZATION DELAY in Oracle, and BACKEND PROCESS FLUSHES DIRTY PAGES in PostgreSQL). As summarized in Table 7, we evaluate two diagnosis reports for each anomaly using the three criteria

Table 7: Case Study of Diagnosis Reports Generated by DBAIOps and Baseline Methods (Complete reports are in our [artifact]).

HEval Criteria		LOG SYNCHRONIZATION DELAY (Oracle Anomaly)		BACKEND PROCESS FLUSHES DIRTY PAGES (PostgreSQL Anomaly)	
		Anomaly Description: The wait occurs during commits or rollbacks while waiting for redo logs to be written to disk, often causing bottlenecks under heavy transactions or poor I/O.		**Anomaly Description:** The abnormal alarm on the metric backend buffer write ratio, which indicates potential misconfigurations in shared buffers or bgwriter/checkpoint settings.	
		Positive Example (HEval = 1.00, by DBAIOps)	**Negative Example (HEval = 0.40, by Tool / DeepSeek-R1 32B)**	**Positive Example (HEval = 1.00, by DBAIOps)**	**Negative Example (HEval = 0.00, by Tool / DeepSeek-R1 32B)**
Root Cause Recall	**Report Content**	✔ **Root Cause:** • (1) Insufficient I/O Performance of REDO Log Storage • (2) Intermittent I/O Pressure Spikes during Log Writing	✘ **Root Cause:** • (1) Log file parallel write anomaly • (2) Redo generation rate anomaly • (3) Checkpoint delay anomaly • (4) Insufficient memory • (5) Control file write anomaly	✔ **Root Cause:** • (1) bgwriter_lru_maxpages too low • (2) I/O latency causing bgwriter failure • (3) bgwriter_lru_multiplier too low	✘ **Root Cause:** • (1) High I/O latency • (2) Misconfigured checkpoints • (3) Insufficient bgwriter • (4) High concurrent writes • (5) Unoptimized SQL
	Comment	• Identify both root causes centered around I/O storage bottleneck	• Only list symptoms and manifestations (i.e., root cause (1),(3), and (5)) • Miss core root cause around I/O bottleneck (i.e., (2) and (4) are not direct root causes)	• Correctly center around root causes of BGWRITER parameters and I/O bottlenecks	• Root cause (4) and (5) describe transaction backlog, irrelevant to REDO logs • Root cause (1) - (3) partially valid but diluted
Theoretical Consistency	**Report Content**	✔ **Reasoning:** • log_file_sync / log_file_parallel_write = 2 → Storage I/O primary factor • Spike at 06:00 with normal OS latency → transient load	✔ **Reasoning:** • Redo surge → log buffer overflow → wait for LGWR • Memory pressure → log buffer insufficient	✔ **Reasoning:** • bgwriter_stop_scan_count >0 → reach bgwriter_lru_maxpages limit → bgwriter stops → backend takes over writes	✘ **Reasoning:** • I/O latency had a maximum value of 5736.96ms and an average value of 827.0ms (normal value should be <10ms) • active sessions had an average of 48 → high concurrency • checkpoint delay had a maximum value of 525,058,688.0ms → checkpoint process severely blocked
	Comment	• Rigorous reasoning chain linking metrics to Oracle and OS knowledge or mechanisms	• Despite root cause errors, reasoning adheres to Oracle principles	• Logically clear, aligned with OS knowledge	• Merely describe abnormal metric alarms rather than establishing causal relations
Evidence Authenticity	**Report Content**	✔ **Evidence:** • Metric 2184301 (log file sync): max=15.2ms, avg=6.0ms • Metric 2184305 (log file parallel write): max=7.09ms, avg=3.0ms	✘ **Evidence:** • Metric 2180503 (checkpoint delay): max=61,660.0ms, avg=61,060.0ms • Metric 2184306 (control file write): max=3.78ms, avg=1.0ms	✔ **Evidence:** • Metric 2300140 (Bgwriter stop scan): max=2.76, avg=1.0 • Metric 3000006 (I/O latency): max=5736.96ms, avg=827.0ms	✔ **Evidence:** • Metric 3000006 (I/O latency): max=5736.96ms, avg=827.0ms • Metric 2300145 (checkpoint delay): max=525,058,688.0ms
	Comment	• All metrics exist in provided data (i.e., no hallucination)	• Cite data that does not exist in provided metrics (i.e., hallucinated metrics)	• Metrics match the provided data (i.e., no hallucination)	• Despite wrong conclusions, metrics exist in provided data (i.e., no hallucination)

(i.e., Root Cause Recall, Theoretical Consistency, and Evidence Authenticity) of Human Evaluation Accuracy (HEval) in Section 7.1.

► ***Root Cause Recall.*** This criterion examines whether the reports correctly identify the direct root causes of anomalies. As shown in Table 7, DBAIOps accurately captures all root causes related to I/O bottlenecks and parameter misconfigurations (e.g., correctly identifying that bgwriter_lru_maxpages is set too low). However, the baseline method mainly lists secondary symptoms rather than direct causes. For instance, in Oracle anomaly, it only reports abnormal patterns of log file parallel write, checkpoint delay, and control file write, missing the actual root cause (i.e., insufficient I/O performance of REDO log storage). This improvement stems from the experience graph and anomaly model in DBAIOps, which effectively characterize and supply LLM with essential knowledge for comprehensive diagnosis (Section 4 and Section 5).

► ***Theoretical Consistency.*** This criterion evaluates whether the reasoning follows established database principles. As displayed in Table 7, DBAIOps consistently grounds its reasoning in theoretical knowledge, forming causal chains that link abnormal metrics to underlying mechanisms. For example, it explicitly relates abnormal bgwriter stop scan counts to bgwriter_lru_maxpages and bgwriter_lru_multiplier settings based on bgwriter and checkpoint principles. In contrast, the baseline method merely enumerates abnormal alarms (e.g., elevated I/O latency and active sessions) without constructing causal relations to identify the root cause (i.e., a misconfiguration of the bgwriter_lru_maxpages parameter). This strength arises from the knowledge-path-based diagnosis strategy in DBAIOps, which augments LLMs with well-organized diagnosis context, and steers them with structured prompts to reason over the path carefully (Section 6.2).

► ***Evidence Authenticity.*** This criterion assesses whether all cited evidence originates from the provided data. As shown in Table 7, DBAIOps exclusively relies on the provided evidence data (e.g., log file parallel write in Oracle and I/O latency in PostgreSQL), while the baseline occasionally cites non-existent or incorrect values, such as a non-existent checkpoint delay metric and an inaccurate maximum of 3.78ms for control file write due to hallucination issues that compromise reliability. Thus, DBAIOps can better constrain the LLM to reason strictly based on the essential evidence identified through the graph and its automatic exploration mechanism (Section 6.1).

8 CONCLUSION

In this paper, we present the first hybrid database O&M system DBAIOps, which combines the benefits of knowledge graphs and reasoning LLMs to support real-world O&M across 25 databases, covering domains such as finance and healthcare. We constructed a heterogeneous graph model that enables the reuse of structured O&M experience across different database systems. We designed a collection of anomaly models from a fine-grained metric hierarchy that captures explicit and implicit metric correlations. We propose a two-stage graph exploration mechanism that adaptively explores diagnosis paths and accumulates experience for newly observed anomalies. We introduced a long-term reasoning mechanism that guides diagnosis through the context of adaptive graph traversal and LLM-based inference. Extensive experiments validated the effectiveness of DBAIOps, demonstrating superior root cause accuracy and report quality compared to traditional and LLM-based methods.

ACKNOWLEDGMENTS

Xuanhe Zhou is the corresponding author. This work was supported in part by National Key R&D Program of China (No. 2023YFB4502400), in part by China NSF grant (No. 62441236, 62372296, 62432007, U25A6024, U25A20437, 62525202, 62232009, 62502304), in part by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM103), in part by Alibaba Group through Alibaba Innovation Research Program, in part by Tencent Rhino Bird Key Research Project, Shenzhen Project (CJGJZD20230724093403007), Zhongguancun Lab, and Beijing National Research Center for Information Science and Technology (BNRist), CCF Populus Grove Fund, ByteDance, Tencent, Ant Group through CCF-Ant Research Fund, Shanghai Jiao Tong University AI for Engineering Initiative.

REFERENCES

[1] [n.d.]. https://dba.stackexchange.com/. Last accessed on 2025-07.
[2] [n.d.]. https://forums.mysql.com/. Last accessed on 2025-07.
[3] [n.d.]. https://learn.microsoft.com/en-us/answers/tags/780/sql-server. Last accessed on 2025-07.
[4] [n.d.]. https://openai.com/index/hello-gpt-4o/. Last accessed on 2024-10.
[5] [n.d.]. https://prometheus.io/. Last accessed on 2025-07.
[6] [n.d.]. https://support.oracle.com. Last accessed on 2025-04.
[7] [n.d.]. http://web.chatdba.com/. Last accessed on 2025-04.
[8] 2023 FAA system outage. *(Wikipedia)*. https://en.wikipedia.org/wiki/2023_FAA_system_outage Last accessed on 2025-07.
[9] Alibaba Database. *(Database)*. https://www.alibabacloud.com/en/product/databases
[10] Amazon Database. *(Database)*. https://aws.amazon.com/cn/free/database
[11] Dameng Database. *(DBMS)*. https://en.dameng.com/
[12] dataVail. *(DBMS)*. https://www.datavail.com/solutions/database-administration/
[13] FAA says 'damaged database file' prompted halt on domestic US flights. *(News)*. https://www.ft.com/content/e65ee681-f242-45f1-b1ab-b5f1b42d8a12 Last accessed on 2025-07.
[14] MySQL. *(DBMS)*. https://www.mysql.com/
[15] Oracle. *(DBMS)*. https://www.oracle.com/database/
[16] Percona. *(DBMS)*. https://try.percona.com/managed-services/
[17] PostgreSQL. *(DBMS)*. https://www.postgresql.org
[18] Rackspace. *(DBMS)*. https://docs.rackspace.com/docs/database-administration-solutions/
[19] Darcy G. Benoit. 2005. Automatic Diagnosis of Performance Problems in Database Management Systems. In *ICAC*. IEEE Computer Society, 326–327.
[20] Sibei Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, and Xiaoyong Du. 2025. Automatic Database Configuration Debugging using Retrieval-Augmented Language Models. *Proc. ACM Manag. Data* 3, 1 (2025), 13:1–13:27.
[21] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. In *SIGMOD Conference*. 666–679.
[22] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437
[23] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948
[24] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. 2005. Automatic Performance Diagnosis and Tuning in Oracle. In *CIDR*. www.cidrdb.org, 84–94.
[25] Dejan Dundjerski and Milo Tomasevic. 2022. Automatic Database Troubleshooting of Azure SQL Databases. *IEEE Trans. Cloud Comput.* 10, 3 (2022), 1604–1619.
[26] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. 2024. A Survey on LLM-as-a-Judge. *CoRR* abs/2411.15594 (2024).
[27] Allan Hirt. 2007. *Pro SQL server 2005 high availability*. Apress. https://www.sqlservercentral.com/wp-content/uploads/2019/05/Hirt_BusinessofAvailability_Apress_780X.pdf
[28] Ihab F. Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed A. Soliman. 2022. Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale. In *SIGMOD Conference*. ACM, 2259–2272.
[29] Prajakta Kalmegh, Shivnath Babu, and Sudeepa Roy. 2017. Analyzing Query Performance and Attributing Blame for Contentions in a Cluster Computing Framework. *CoRR* abs/1708.08435 (2017).
[30] Prajakta Kalmegh, Shivnath Babu, and Sudeepa Roy. 2019. iQCAR: inter-Query Contention Analyzer for Data Analytics Frameworks. In *SIGMOD Conference*. ACM, 918–935.
[31] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nat.* 521, 7553 (2015), 436–444.
[32] Xiaoze Liu, Zheng Yin, Chao Zhao, Congcong Ge, Lu Chen, Yunjun Gao, Dimeng Li, Ziting Wang, Gaozhong Liang, Jian Tan, and Feifei Li. 2022. PinSQL: Pinpoint Root Cause SQLs to Resolve Performance Issues in Cloud Databases. In *ICDE*. IEEE, 2549–2561.
[33] Xianglin Lu, Zhe Xie, Zeyan Li, Mingjie Li, Xiaohui Nie, Nengwen Zhao, Qingyang Yu, Shenglin Zhang, Kaixin Sui, Lin Zhu, and Dan Pei. 2022. Generic and Robust Performance Diagnosis via Causal Inference for OLTP Database Systems. In *CCGRID*. IEEE, 655–664.
[34] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. 2020. Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases. *Proc. VLDB Endow.* 13, 8 (2020), 1176–1189.
[35] Yongli Mou, Li Liu, Sulayman K. Sowe, Diego Collarana, and Stefan Decker. 2024. Leveraging LLMs Few-shot Learning to Improve Instruction-driven Knowledge Graph Construction. In *VLDB Workshops*. VLDB.org.
[36] Cory S. Myers and Lawrence R. Rabiner. 1981. Connected word recognition using a level building dynamic time warping algorithm. In *ICASSP*. IEEE, 951–955.
[37] Biao Ouyang, Yingying Zhang, Hanyin Cheng, Yang Shu, Chenjuan Guo, Bin Yang, Qingsong Wen, Lunting Fan, and Christian S. Jensen. 2025. RCRank: Multimodal Ranking of Root Causes of Slow Queries in Cloud Database Systems. *CoRR* abs/2503.04252 (2025).
[38] Vikramank Y. Singh, Kapil Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Balakrishnan Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance Debugging for Databases using LLM Agents. In *CIDR*. www.cidrdb.org.
[39] Zirui Tang, Weizheng Wang, Zihang Zhou, Yang Jiao, Bangrui Xu, Boyu Niu, Xuanhe Zhou, Guoliang Li, Yeye He, Wei Zhou, Yitong Song, Cheng Tan, Bin Wang, Conghui He, Xiaoyang Wang, and Fan Wu. 2025. LLM/Agent-as-Data-Analyst: A Survey. *arXiv preprint* (2025). https://arxiv.org/abs/2509.23988
[40] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. DBSherlock: A Performance Diagnostic Tool for Transactional Databases. In *SIGMOD Conference*. ACM, 1599–1614.
[41] Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. 2024. A Comprehensive Survey on Automatic Knowledge Graph Construction. *ACM Comput. Surv.* 56, 4 (2024), 94:1–94:62.
[42] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li. 2025. Cracking SQL Barriers: An LLM-based Dialect Translation System. *Proc. ACM Manag. Data* 3, 3 (2025), 141:1–141:26.
[43] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li. 2025. CrackSQL: A Hybrid SQL Dialect Translation System Powered by Large Language Models. *arXiv Preprint* (2025). https://arxiv.org/abs/2504.00882
[44] Wei Zhou, Guoliang Li, Haoyu Wang, Yuxing Han, Xufei Wu, Fan Wu, and Xuanhe Zhou. 2025. PARROT: A Benchmark for Evaluating LLMs in Cross-System SQL Translation. In *NeurIPS*.
[45] Wei Zhou, Chen Lin, Xuanhe Zhou, and Guoliang Li. 2024. Breaking It Down: An In-depth Study of Index Advisors. *Proc. VLDB Endow.* 17, 10 (2024), 2405–2418.
[46] Wei Zhou, Chen Lin, Xuanhe Zhou, Guoliang Li, and Tianqing Wang. 2024. TRAP: Tailored Robustness Assessment for Index Advisors via Adversarial Perturbation. In *ICDE*. IEEE, 42–55.
[47] Wei Zhou, Ji Sun, Xuanhe Zhou, Guoliang Li, Luyang Liu, Hao Wu, and Tianyuan Wang. 2025. GaussMaster: An LLM-based Database Copilot System. *arXiv preprint arXiv* (2025). https://arxiv.org/abs/2506.23322
[48] Wei Zhou, Jun Zhou, Haoyu Wang, Zhenghao Li, Qikang He, Shaokun Han, Guoliang Li, Xuanhe Zhou, Yeye He, Chunwei Liu, Zirui Tang, Bin Wang, Shen Tang, Kai Zuo, Yuyu Luo, Zhenzhe Zheng, Conghui He, Jingren Zhou, and Fan Wu. 2026. Can LLMs Clean Up Your Mess? A Survey of Application-Ready Data Preparation with LLMs. *arXiv preprint* (2026). https://arxiv.org/abs/2601.17058
[49] Xuanhe Zhou, Junxuan He, Wei Zhou, Haodong Chen, Zirui Tang, Haoyu Zhao, Xin Tong, Guoliang Li, Youmin Chen, Jun Zhou, Zhaojun Sun, Binyuan Hui, Shuo Wang, Conghui He, Zhiyuan Liu, Jingren Zhou, and Fan Wu. 2025. A Survey of LLM × DATA. *arXiv preprint arXiv* (2025). https://arxiv.org/abs/2505.18458
[50] Xuanhe Zhou, Lianyuan Jin, Ji Sun, Xinyang Zhao, Xiang Yu, Shifu Li, Tianqing Wang, Kun Li, and Luyang Liu. 2021. DBMind: A Self-Driving Platform in openGauss. *Proc. VLDB Endow.* 14, 12 (2021), 2743–2746.
[51] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-Bot: Database Diagnosis System using Large Language Models. *Proc. VLDB Endow.* 17, 10 (2024), 2514–2527.
[52] Xuanhe Zhou, Wei Zhou, Liguo Qi, Hao Zhang, Dihao Chen, Bingsheng He, Mian Lu, Guoliang Li, Fan Wu, and Yuqiang Chen. 2025. OpenMLDB: A Real-Time Relational Data Feature Computation System for Online ML. In *SIGMOD Conference Companion*. ACM, 729–742.