

AGRL: An Agentic GraphRAG Framework with Reinforcement Learning for Database Configuration Debugging

CHENGYANG LUO, Zhejiang University, China

QING LIU, Zhejiang University, China

YUJIE GUO, Zhejiang University, China

KAI WANG, Shanghai Jiao Tong University, China

WENJIE ZHANG, University of New South Wales, Australia

YUNJUN GAO, Zhejiang University, China

Database configuration debugging aims to identify suitable parameter (knob) settings to resolve performance or stability issues. This task remains challenging even for experts due to the complexity and interdependencies of modern database settings. Recent work has explored vector-based retrieval-augmented generation (RAG) to assist large language models (LLMs) in automatic configuration debugging by retrieving relevant database knowledge. However, this approach relies solely on surface-level semantic similarity, missing the deeper conceptual relationships essential for root cause diagnosis, which limits its effectiveness. To overcome this, we propose AGRL, an agentic GraphRAG framework empowered by reinforcement learning for database configuration debugging. AGRL has two core designs. First, a synergistic component co-design integrates: (1) a hybrid graph constructor utilizing historical Q&A pairs as semantic bridges to enrich database manuals; (2) a reinforced graph retriever featuring a mixture-of-experts (MoE) encoder and a dual-stage decision mechanism to extract precise subgraphs; and (3) a multi-round debugging agent that decomposes queries and performs an iterative "Reason-Act-Reflect" workflow while regulating retrieval breadth and depth. Second, AGRL employs a progressive training strategy, which combines decoupled initialization with iterative joint refinement, to overcome the cold-start interdependency between the agent and the retriever. Extensive experiments on real-world datasets demonstrate that AGRL significantly outperforms state-of-the-art baseline methods.

CCS Concepts: • **Information systems** → **Autonomous database administration**; • **Computing methodologies** → *Semantic networks*.

Additional Key Words and Phrases: Database configuration debugging; Graph retrieval-augmented generation; Large language models; Reinforcement learning

ACM Reference Format:

Chengyang Luo, Qing Liu, Yujie Guo, Kai Wang, Wenjie Zhang, and Yunjun Gao. 2026. AGRL: An Agentic GraphRAG Framework with Reinforcement Learning for Database Configuration Debugging. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 254 (September 2026), 26 pages. <https://doi.org/10.1145/3837092>

Authors' Contact Information: Chengyang Luo, luocy1017@zju.edu.cn, Zhejiang University, Hangzhou, China; Qing Liu, qingliucs@zju.edu.cn, Zhejiang University, Hangzhou, China; Yujie Guo, yujie1.25@intl.zju.edu.cn, Zhejiang University, Hangzhou, China; Kai Wang, cskaelwang@gmail.com, Shanghai Jiao Tong University, Shanghai, China; Wenjie Zhang, wenjie.zhang@unsw.edu.au, University of New South Wales, Sydney, Australia; Yunjun Gao, gaoyj@zju.edu.cn, Zhejiang University, Hangzhou, China.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/9-ART254

<https://doi.org/10.1145/3837092>

1 Introduction

Database configuration debugging aims to diagnose performance issues or system errors and recommend appropriate parameters (often called knobs) to resolve them [3, 5]. This process is essential for ensuring the performance and reliability of modern database management systems (DBMSs). However, a gap remains between the way users describe problems and the way debugging tools operate. In real-world scenarios, users typically describe symptoms in natural language (e.g., "*I recently migrated to a...*" in Figure 1), rather than specifying concrete system metrics. Traditional tuning methods [4, 5, 7, 15, 18, 27, 31–33] cannot handle natural language (NL) input. Thus, database administrators (DBAs) must manually analyze the NL question by consulting manuals and historical records, and then provide appropriate knob recommendations based on their own debugging experience to resolve the issue.

The emergence of large language models (LLMs) shows promise for end-to-end configuration debugging, as they can accept natural language queries and suggest actionable knobs. Nevertheless, relying solely on LLMs is insufficient as they lack specific domain knowledge, often leading to generic or inaccurate knobs recommendations. To address this, Chen et al. [3] propose the Andromeda framework, which uses a vector-based retrieval-augmented generation (RAG) approach. It converts database manuals and historical Q&A pairs into embeddings, retrieves the snippets most similar to the user's question, and uses them as context for the LLM to generate a response. Figure 1 illustrates a MySQL performance issue in which TPS drops by 40% after migration to a 64-core server. The relevant knob information is shown on the right side of the figure. Andromeda retrieves some snippets that show excessive concurrent threads can cause severe resource contention. Consequently, the LLM recommends executing `SET innodb_thread_concurrency = 64`. However, this advice is ineffective. The user has already disabled the concurrency limit by setting this parameter to 0. Furthermore, the absence of significant row locks indicates that thread contention is not the root cause, making further concurrency limits futile.

In fact, tuning-relevant database entities, such as configuration knobs commonly used in practice (e.g., `innodb_table_locks`), performance metrics (e.g., *Avg Trans per Group*), and workload patterns (e.g., *High Concurrency Write*), are intricately interconnected, reflecting database's runtime behavior and internal mechanisms. Effective configuration debugging necessitates deep reasoning on these entities and their structural dependencies. However, vector-based RAG **neglects structural relationships**, focusing on retrieving isolated snippets based on semantic similarity. Since there is a significant semantic gap between the user's query and the underlying database mechanisms, the retrieved context may superficially resemble the query but lack the causal or logical connections required for accurate problem-solving. This leads to suboptimal or incorrect LLM responses.

Recently, Graph-based RAG (GraphRAG) [6, 9, 35] has emerged as a promising paradigm to address this limitation, as graphs provide a natural and effective way to model relationships between entities. By explicitly modeling tuning-relevant database entities and their interactions as a graph, GraphRAG enables the LLM to reason over complex relational structures which are difficult for purely vector-based methods to capture. For example, the subgraph in Figure 1 captures the interactions under *Double 1* persistence settings, a strict configuration that flushes logs to disk upon every transaction commit for maximum data safety. Under such settings, a *High Concurrency Write* workload, characterized by massive simultaneous data modifications, frequently triggers severe *Binary Log Mutex Wait* during the *Binary Log Group Commit* process. To improve efficiency, the system could increase the *Avg Trans per Group* by bundling more transactions into a single flush operation. This can be achieved by adjusting `binlog_group_commit_sync_delay`, a parameter that controls the wait time before synchronizing the binary log to disk. By extending this delay, the leader thread can alleviate the *Binary Log Mutex Wait* and reduce IOPS pressure. Thus, the graph-based insight

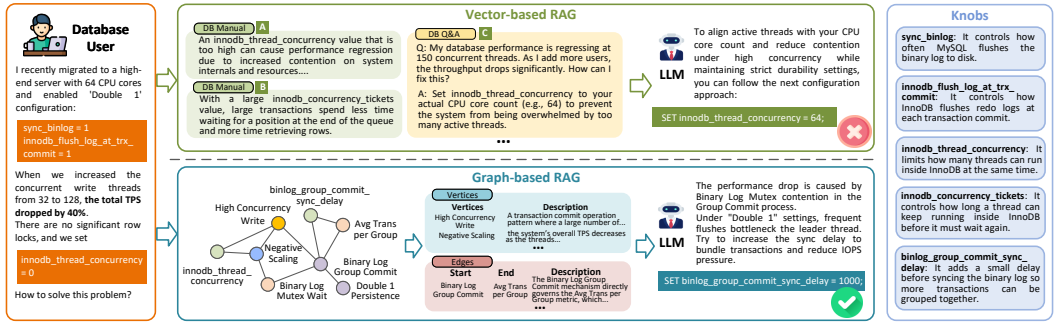


Fig. 1. A motivating example in MySQL

helps the LLM recommend a useful fix for the bottleneck: SET binlog_group_commit_sync_delay = 1000.

Challenges. However, applying GraphRAG techniques to DBMS configuration debugging faces several challenges.

C1. How to construct an effective graph from heterogeneous sources? While manuals and historical Q&A pairs are essential for configuration debugging, naively using LLMs to extract a global graph from these sources is both costly and ineffective. The inherent noise and informality in Q&A discussions severely contaminate the structural dependencies required for diagnosis. Empirically, this naive approach increases construction time by an order of magnitude (Exp-2 in Section 6) and degrades average accuracy by 13.3% (Exp-10, Table 3).

C2. How to adaptively retrieve subgraphs for specialized configuration questions? Since debugging queries demand highly varied information, accurate diagnosis requires dynamically tailored subgraphs. However, existing GraphRAG methods [6, 9, 11, 19] employ rigid global or fixed-hop retrieval strategies that fail to accommodate this query diversity. These non-adaptive approaches inevitably compromise precision: broad retrieval introduces excessive noise [16], while narrow retrieval omits crucial context. Consequently, they suffer at least a 20% accuracy degradation (Exp-1, Table 2). Our ablation study further confirms this necessity, demonstrating significant performance drops when adaptive search modules are removed (Exp-10, Table 3).

C3. How to solve complex configuration debugging tasks? Difficult debugging is an iterative, investigative process. Existing methods act as passive generators with single-round retrieval, which can lead them to commit to a diagnosis based on incomplete initial context and limits their ability to reflect on intermediate results. This lack of a sequential decision-making mechanism [28] severely limits effectiveness. Our experiments confirm this: expanding exploration rounds significantly boosts debugging performance (Exp-4), whereas restricting the system to a single-round retrieval causes a 30.89% drop in average accuracy (Exp-10, Table 3).

Our Solutions. To address these challenges, we propose **AGRL**, a novel agentic GraphRAG framework empowered by reinforcement learning for DBMS configuration debugging. AGRL has two core system-level designs. First, AGRL employs a **synergistic component co-design** that integrates a *hybrid graph constructor*, a *reinforced graph retriever*, and a *multi-round debugging agent* into a unified diagnostic workflow. Second, it introduces a **progressive training strategy** using reinforcement learning to optimize the retriever and the agent, addressing their interdependence.

In this synergistic design, the three components form an interactive closed loop to resolve challenges C1 through C3. (1) **Hybrid Graph Constructor (for C1).** AGRL constructs a base graph from heterogeneous knowledge sources. It incorporates entities extracted from manuals and

integrates historical Q&A pairs as explicit *bridge vertices*. This design improves graph quality while reducing computational overhead. (2) **Reinforced Graph Retriever (for C2)**. Powered by a Shared-Specific Mixture-of-Experts (SS-MoE) encoder to identify query intents, this module employs a dual-stage decision mechanism. A *Scope Controller* dynamically scales the search radius, while a *Vertex Selector* prioritizes relevant entities, facilitating the extraction of query-adaptive subgraphs. (3) **Multi-round Debugging Agent (for C3)**. Serving as the cognitive core, the agent decomposes problems and autonomously triggers multiple retrieval rounds. By reflecting on intermediate retrieval results, it adaptively controls the search target, breadth, and exploration depth, thereby supporting a rigorous diagnostic process.

AGRL introduces a *progressive training strategy* with three decoupled phases to overcome the cold-start interdependency dilemma between the agent and the retriever. Specifically, **Phase I** pre-trains the retriever via PPO [23] to establish fundamental graph traversal capabilities. **Phase II** freezes the retriever to align the debugging agent via GRPO [8, 24], enabling it to master multi-step reasoning and instruction generation. Finally, **Phase III** implements iterative joint refinement, co-training both components under a shared task-success reward. This allows their strategies to co-evolve, ultimately improve end-to-end diagnostic accuracy.

Contributions. Overall, our main contributions are as follows.

- We propose AGRL, a novel agentic GraphRAG framework for DBMS configuration debugging. Empowered by reinforcement learning, AGRL is able to perform multi-round reasoning and adaptive graph retrieval to solve complex configuration issues.
- We formulate a synergistic component co-design integrating a hybrid graph constructor to handle heterogeneous documents, a reinforced graph retriever for adaptive subgraph extraction, and a multi-round debugging agent that autonomously modulates search depth and breadth.
- We design a novel progressive training strategy to jointly optimize the LLM-based agent and the retriever through decoupled phases, effectively overcoming their interdependency and cold-start dilemmas.
- We conduct extensive experiments on real-world database debugging datasets, and the comprehensive empirical results demonstrate that AGRL achieves superior effectiveness compared to baseline methods.

2 Related Work

► **DBMS Diagnosis and Tuning.** DBMS diagnosis and tuning aim to detect performance bottlenecks, identify root causes, and optimize configurations to ensure reliable system operation. Existing studies generally fall into two main categories.

▷ (1) Traditional Approaches. Traditional DBMS diagnosis and tuning techniques mainly rely on empirical rules (e.g., ADDM [4], iSQUAD [18], and DBSherlock [31]), statistical feature selection (e.g., OtterTune [27] and Tuneful [7]), Bayesian optimization (e.g., iTuned [5] and ResTune [33]), and machine learning (e.g., CDBTune [32], Qtune [15]) to automatically optimize system performance based on workload metrics. However, these methods rely on concrete system metrics as input and optimize predefined objectives such as latency or throughput. They do not support natural language input and cannot generate interpretable, issue-specific configuration recommendations.

▷ (2) LLM-based methods. Recent work applies LLMs to DBMS diagnosis, falling into three categories based on optimization goals and interface design. (1) The first uses LLMs to process DB domain knowledge to assist with configuration tuning. GPTuner [13] uses LLMs to extract and structure knowledge from DBMS manuals, which then guides a coarse-to-fine Bayesian optimization process to generate high-performance database configurations. However, GPTuner does not

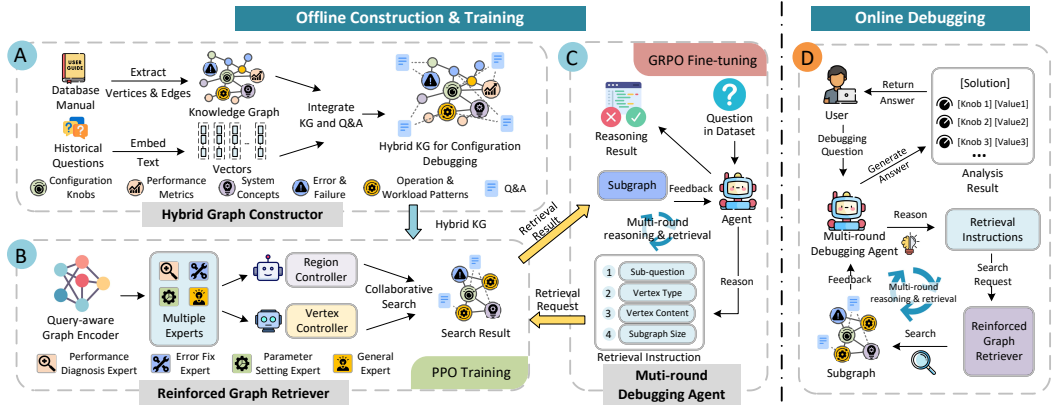


Fig. 2. The design of the AGRL framework

use the LLM to directly generate configurations or respond to user questions. (2) The second uses LLMs as diagnostic advisors that provide textual troubleshooting guidance. Panda [25] takes natural language questions as input and employs retrieval-augmented generation to provide text-based troubleshooting analysis. Similarly, D-Bot [30] is triggered by system alerts to generate diagnosis reports identifying root causes and solutions. Note that Panda and D-Bot provide textual advice rather than executable knob settings. (3) The third uses LLMs to generate explicit knob configuration recommendations based on user's question. Andromeda [3] accepts natural language questions as input and utilizes vector-based RAG techniques to retrieve domain knowledge from manuals and historical records. Its final output explicitly recommends specific DBMS configuration knob settings.

AGRL studies the same problem as Andromeda, aiming to provide concrete knob suggestions based on users' natural language inputs. However, AGRL goes beyond Andromeda's static vector retrieval approach. By introducing an agentic GraphRAG framework, AGRL captures deep structural dependencies among database entities and supports multi-round reasoning and retrieval.

► **GraphRAG.** GraphRAG has emerged as an effective approach for enhancing LLMs with external knowledge structured as graphs. Unlike vanilla RAG, which retrieves isolated text chunks, GraphRAG leverages graph structures to capture relationships among entities and support multi-hop reasoning. Existing GraphRAG techniques can be grouped according to their retrieval granularity and underlying graph types. (1) The first retrieves either hierarchical tree nodes or entity-linked passages (e.g., RAPTOR [22], KGP [29]). (2) The second category leverages knowledge graphs to perform retrieval at the entity, relation, or path/subgraph level (e.g., HippoRAG [12], G-retriever [10], ToG [26], DALK [14]). (3) The third goes further by constructing textual or enriched knowledge graphs that incorporate explicit descriptions, community structures, and hierarchical organizations, thereby enabling multi-level semantic retrieval (e.g., GraphRAG [6], LightRAG [9], ToG2 [19], and KET-RAG [11]). These methods follow a shared workflow involving graph construction, index generation, and retrieval-driven prompting.

However, existing GraphRAG methods are not specifically designed for configuration debugging. They rely on rigid retrieval strategies and lack tight integration with agents, which limits their effectiveness in addressing complex debugging tasks.

3 Preliminary

In this section, we introduce key concepts related to the database configuration debugging.

Definition 3.1 (Debugging Question). Let D denote a database management system (DBMS). A debugging question is a user-posed natural language query that reports unexpected configuration issues and requires diagnostic procedures over D .

A debugging question q may contain a variety of configuration debugging purposes, including but not limited to: (1) **Performance Anomalies**: cases where a database operation exhibits long latency or abnormal resource consumption (e.g., CPU, memory, or disk I/O). (2) **Runtime Errors**: failures that occur during DBMS execution, often reported as explicit error messages. (3) **User-specific Configuration**: requests that involve modifying default parameter settings or customizing system behaviors.

A well-structured debugging query usually includes specific elements: (1) existing knob configurations or prior tuning attempts; (2) performance metrics; (3) error logs; (4) workload characteristics like SQL patterns and concurrency; and (5) environmental context, including schema design and hardware details. These elements are optional, yet richer context tends to improve diagnostic accuracy. Furthermore, when user-provided information is incomplete or inaccurate, AGRL explores alternative scenarios to deduce viable solutions, as detailed in the technical report [17].

Definition 3.2 (Configurable Knobs). For a DBMS, let $\mathcal{K}$ denote the set of configurable knobs. Each knob $\kappa \in \mathcal{K}$ corresponds to a controllable system parameter and is associated with a value domain $\mathcal{D}_\kappa$. A value ω is a valid setting of knob κ if and only if $\omega \in \mathcal{D}_\kappa$.

Modern DBMSs typically contain hundreds of such knobs, each governing a specific aspect of system behavior. For example, MySQL 8.0 contains more than one thousand knobs (such as `wait_timeout` or `unique_checks`), while PostgreSQL 12.0 provides several hundred knobs (such as `trace_locks`). These knobs play essential roles in performance tuning, behavior customization, and troubleshooting. For example, MySQL's `innodb_thread_concurrency` (Figure 1) limits the number of threads that can run concurrently inside InnoDB, helping to reduce contention and improve performance under heavy load. Next, we present the problem.

Problem 1 (DBMS Configuration Debugging [3]). Given an NL debugging question q over a DBMS D , the goal of DBMS configuration debugging is to:

- (1) Identify a subset $\mathcal{K}' \subseteq \mathcal{K}$ of knobs that are relevant to the issues described in q ;
- (2) Recommend for each knob $\kappa \in \mathcal{K}'$ an appropriate value $\omega \in \mathcal{D}_\kappa$ that can resolve the reported problem.

4 AGRL Framework

Figure 2 illustrates AGRL, comprising offline training and online debugging phases. These phases operate through a system-level co-design of three key components: the *hybrid graph constructor* prepares a DBMS-specific debugging knowledge graph, the *reinforced graph retriever* extracts query-adaptive subgraphs, and the *multi-round debugging agent* directs retrieval during diagnosis. Following offline graph construction and model training, the online agent processes user queries by issuing explicit search instructions to the retriever. Utilizing the retrieved subgraphs, the agent iteratively refines its reasoning to either initiate further searches or formulate a final diagnostic suggestion. This synergistic process effectively resolves diverse DBMS configuration issues. Pseudocodes and proofs are deferred to our technical report [17].

4.1 Hybrid Graph Constructor

► **Overview.** For the DBMS configuration debugging problem, two primary sources of knowledge are available: official database manuals and historical debugging Q&A pairs posted by users on community forums. The manuals provide authoritative descriptions of configuration parameters and

operational principles. Historical debugging Q&A pairs are real-world discussions from database community forums, each consisting of an actual user question and its corresponding solution.

Q&A pairs often contain informal or nonstandard language. Directly extracting fine-grained details from both manuals and Q&A pairs would introduce noise and high computational cost. To address this, AGRL constructs a hybrid knowledge graph by modeling Q&A pairs as explicit bridge vertices that connect relevant entities from the manuals. This graph structure serves as the foundational knowledge base for downstream retrieval and reasoning tasks.

► **Graph Modeling.** A Knowledge Graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. Each vertex $v \in \mathcal{V}$ is denoted by $v = (l_v, d_v)$, where l_v is its label (i.e., the corresponding entity name) and d_v is its textual description. Each edge $e \in \mathcal{E}$ is represented as a triplet $e = (u, v, d_e)$, where $u, v \in \mathcal{V}$ are the connected vertices, and d_e is the textual description of the relationship. To make environmental discrepancies and potentially outdated fixes explicit, we incorporate DBMS versions into manual-derived elements and timestamps into Q&A vertices. Rather than storing this metadata separately, we include it directly in textual descriptions so that it appears in the retrieved context. This provides explicit version and temporal context for the debugging agent when assessing version compatibility and the timeliness of forum solutions, without relying on hard-coded filtering rules.

We use h_v to denote the embedding of the textual description for vertex v , and let $N(v)$ represent the set of neighbors of v . Similarly, $h_{(u,v)}$ denotes the embedding of the edge connecting u and v . The embeddings are obtained using a pre-trained language model, such as Sentence-BERT [21] or Qwen3-Embedding [34]. We define six vertex schemas for the configuration debugging task.

❶ **Configuration Knobs Vertex.** Configuration knobs are controllable parameters that directly govern DBMS behavior (e.g., resource allocation, concurrency/locking policies, and security). Each knob vertex corresponds to a canonical configuration parameter. For example, the parameter `innodb_thread_concurrency` in Figure 1 is a knob vertex that controls the maximum number of concurrent threads. During debugging, these vertices provide explicit adjustment targets, allowing the agent to identify which DBMS parameters should be inspected or modified.

❷ **Performance Metrics Vertex.** Performance metrics are quantitative runtime indicators of system health and workload efficiency, such as latency (average execution time), IOPS (I/O capacity), and CPU utilization (resource contention). In Figure 1, *Avg Trans per Group* is a metric vertex that represents the average number of transactions processed per group. Tracing paths from metric vertices helps identify the factors that impact performance. Together with knob and workload vertices, these metric vertices provide structural context for analyzing configuration-related performance changes.

❸ **System Concepts Vertex.** System concepts vertices represent key DBMS components and mechanisms, such as the InnoDB buffer pool, WAL, and SSL/TLS subsystems. In Figure 1, *Binary Log Group Commit* and *Double 1 Persistence* are concept vertices that represent key mechanisms in transaction and durability management. They act as semantic hubs, linking knobs, metrics, and other entities to enable multi-hop reasoning. This abstraction further connects low-level settings with higher-level behaviors that may otherwise appear unrelated.

❹ **Error & Failure Vertex.** Errors denote explicit malfunction manifestations observed during execution, such as *PKIX path building failed*, *Too many connections*, or *Disk full errors*. In Figure 1, *Negative Scaling* is an error vertex. Modeling such vertices enables systematic tracking and root-cause analysis of failures. These vertices also preserve concrete diagnostic symptoms that can serve as entry points for subsequent causal reasoning.

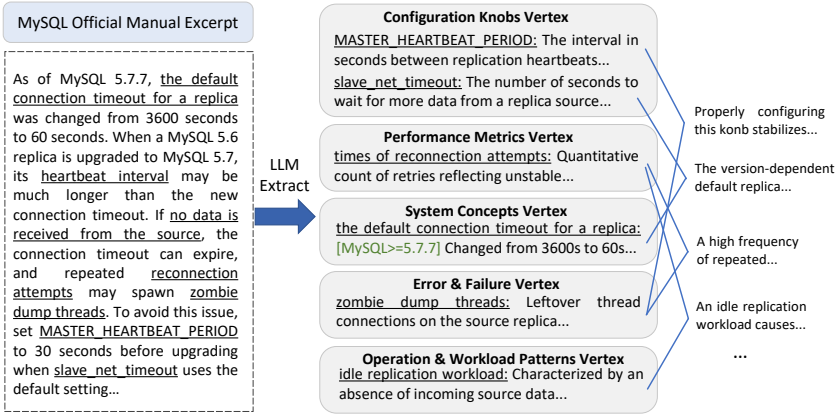


Fig. 3. An extraction example

5 Operation & Workload Patterns Vertex. Operation and workload pattern vertices capture the execution context in which a DBMS operates. In Figure 1, *High Concurrency Write* is modeled as a workload vertex, representing a scenario characterized by numerous concurrent write operations. These vertices enable context-aware retrieval, as the effectiveness of configuration knobs is highly dependent on the specific workload regime. Explicit workload context further helps the retriever identify evidence relevant to the current operating scenario.

6 Q&A Vertex. Q&A vertices encapsulate concrete debugging scenarios and their corresponding solutions, drawn from community discussions. Unlike the static, formal definitions found in official manuals, each Q&A vertex represents a real-world question-and-answer instance. Specifically, every Q&A vertex corresponds to a single Q&A pair.

► **Construction Workflow.** The construction of the hybrid knowledge graph proceeds in two sequential stages: (1) *Manual-based Graph Backbone Construction* and (2) *Q&A-based Enrichment*.

First, to build the graph backbone, AGRL employs an LLM to extract entities and relations from official DBMS manuals, forming the base graph $\mathcal{G}_{base} = (\mathcal{V}_{base}, \mathcal{E}_{base})$. The vertices in $\mathcal{V}_{base}$ belong to the first five schema types defined above and represent system operation specifications. The prompt and the pseudocode of extraction is provided in the technical report [17].

Extraction Example. As illustrated in Figure 3, the LLM extracts and categorizes entities in the MySQL manual into predefined vertices: MASTER_HEARTBEAT_PERIOD and slave_net_timeout as *Configuration Knobs*, “times of reconnection attempts” as *Performance Metrics*, “the default connection timeout for a replica” as *System Concepts*, “zombie dump threads” as *Error & Failure*, and “idle replication workload” as *Operation & Workload Patterns*. The extracted *System Concepts* vertex is additionally enriched with version metadata (e.g., [MySQL ≥ 5.7.7]). Moreover, the LLM extracts the relationships among these vertices as edges in the graph.

Second, AGRL captures empirical experience by modeling each historical Q&A pair as a vertex v_{qa} . Formally, for a Q&A pair with embedding $\mathbf{h}_{qa}$, we calculate its semantic similarity with existing vertices in $\mathcal{V}_{base}$. We select the set of top- k_{QA} most relevant vertices from the current graph for subsequent enrichment, i.e. $V_{target} = \text{Top-}k_{QA} \ v \in \mathcal{V}_{base} (\text{sim}(\mathbf{h}_{qa}, \mathbf{h}_v))$. AGRL then adds the new vertex v_{qa} to the graph and establishes edges between v_{qa} and every vertex in V_{target} .

This construction process offers significant advantages. By instantiating Q&A pairs as independent vertices, AGRL creates semantic “shortcuts” between manual concepts that may not be explicitly linked in the documentation (e.g., linking a specific *Error* directly to a *Knob* via a shared case). Furthermore, this approach enables efficient incremental updates. New Q&A data can be

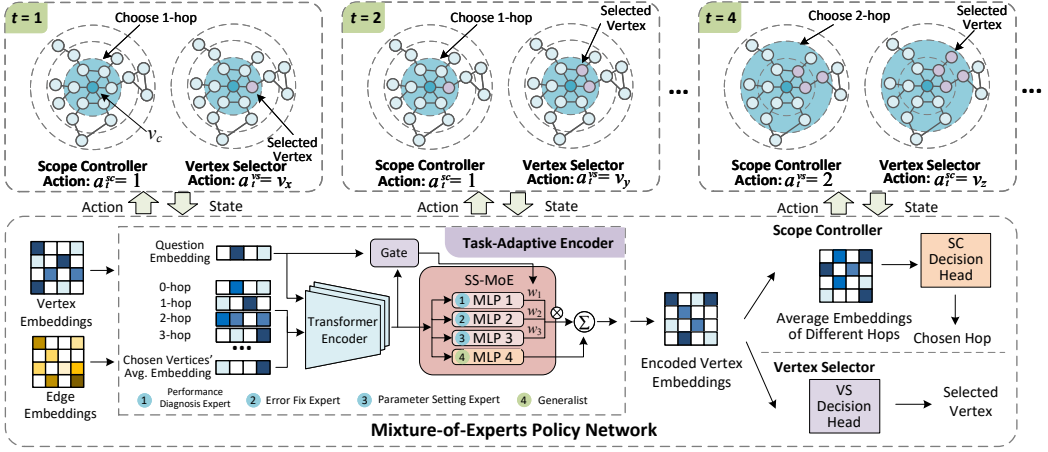


Fig. 4. The reinforced graph retriever

continuously incorporated by simply adding new vertices and edges, without requiring costly LLM-based extraction.

4.2 Reinforced Graph Retriever

► **Overview.** The Reinforced Graph Retriever serves as the intelligent retrieval engine of AGRL. It is designed to execute the Multi-hop Debugging Agent’s high-level retrieval instructions within the database knowledge graph. Unlike conventional RAG retrievers that rely on static vector similarity, this module functions as an adaptive navigator. It can capture intricate structural relationships among database entities, and dynamically identify the most useful information in response to the given question.

However, navigating such a large-scale graph to find relevant diagnostic context is difficult due to the massive search space. To address this combinatorial challenge, we formulate the retrieval process as a sequential Markov Decision Process [20] (MDP), denoted as $\mathcal{M}(\mathcal{G}, q, v_c) = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$. Here, q represents the specific question decomposed by the agent, and v_c denotes the starting center vertex identified as the semantic anchor. Within this framework, the retriever incrementally constructs the diagnostic subgraph $\mathcal{C}$ by transitioning between states $\mathcal{S}$ via discrete actions $\mathcal{A}$, guided by a reward signal $\mathcal{R}$ that reflects both semantic relevance and task utility. As illustrated in Figure 4, this MDP is solved through a sophisticated Perception-Decision framework:

- **Task-Adaptive Perception.** To handle the complex database graph, we propose a *Task-Adaptive Graph Encoder* using a *Shared-Specific Mixture-of-Experts* (SS-MoE) network. This module dynamically adjusts vertex representations according to the query, producing context-aware embeddings for decision-making.
- **Dual-Stage Graph Navigation.** Based on these embeddings, the retriever performs a sequential search using a *Dual-Stage Decision Mechanism*. This mechanism iteratively expands the diagnostic subgraph by first determining a search radius (Scope Controller) and then selecting the most valuable vertex (Vertex Selector) within that region.

► **Task-Adaptive Graph Encoder.** Standard graph neural networks (GNNs) suffer from over-smoothing problem [1, 2] and fail to capture the divergent semantic spaces of heterogeneous database entities. To overcome this, we design a hierarchical encoding process that synergizes heterogeneous context integration with a mixture-of-experts policy network.

▷ Heterogeneous Context Encoding. We construct a state-aware input sequence to capture the full decision context. First, since a candidate vertex should be evaluated in relation to the subgraph retrieved so far, we initialize its feature vector $\mathbf{x}_v$ by fusing its textual embedding $\mathbf{h}_v$ with structural information from edges linking it to the current retrieved subgraph C :

$$\mathbf{x}_v = \frac{1}{2} \left(\mathbf{h}_v + \frac{1}{|C \cap N(v)|} \sum_{v' \in C \cap N(v)} \mathbf{h}_{(v,v')} \right) \quad (1)$$

where $\mathbf{h}_{(v,v')}$ denotes the embedding of edge (v, v') . Next, a comprehensive input sequence S_v is formed by aggregating multi-source context. Specifically, let ${}^k\bar{\mathbf{x}}_v$ denote the average feature representation of the k -hop neighborhood of vertex v . The sequence is defined as: $S_v = [{}^0\bar{\mathbf{x}}_v, {}^1\bar{\mathbf{x}}_v, \dots, {}^K\bar{\mathbf{x}}_v, \mathbf{h}_q, \bar{\mathbf{x}}_H]$. This sequence comprehensively reflects the entire retrieval state: hop-wise summaries (${}^k\bar{\mathbf{x}}_v$) encode local structural contexts, the query embedding ($\mathbf{h}_q$) dictates the task intent, and the global state ($\bar{\mathbf{x}}_H$) encapsulates the retrieved context. We compute ${}^k\bar{\mathbf{x}}_v$ via neighbor averaging to accommodate unordered and variable-sized graph neighborhoods. This mean aggregation produces an efficient, permutation-invariant initial summary. Notably, this averaging is utilized solely to construct the input sequence rather than to establish the final representation. The sequence S_v is projected into the model's hidden dimension using a linear transformation $\mathbf{W}_0 \in \mathbb{R}^{d \times d_m}$, followed by the addition of positional encodings.

$$\mathbf{H}_v^{(0)} = S_v \mathbf{W}_0 + \mathbf{P} \quad (2)$$

We subsequently feed $\mathbf{H}_v^{(0)}$ into an L -layer Transformer encoder. To mitigate the over-smoothing issue inherent in traditional GNNs, AGRL abandons recursive message passing. By discretizing the $0 \sim K$ -hop neighborhoods into separate sequence tokens, the Transformer selectively attends to these structural contexts. This design helps preserve hop-specific information and mitigates continuous dilution caused by GNNs' layer-wise averaging. Each layer comprises a multi-head self-attention block followed by a position-wise feed-forward network. Both components utilize residual connections and layer normalization.

$$\begin{aligned} \tilde{\mathbf{H}}_v^{(l)} &= \text{MHA}(\mathbf{H}_v^{(l)}) + \mathbf{H}_v^{(l)} \\ \mathbf{H}_v^{(l+1)} &= \text{FFN}(\tilde{\mathbf{H}}_v^{(l)}) + \tilde{\mathbf{H}}_v^{(l)} \end{aligned} \quad (3)$$

Upon processing through the final layer, we extract the final state of the first token (corresponding to the original vertex v) as the aggregate vertex embedding $\mathbf{z}_v = {}^0\mathbf{H}_v^{(L)}$. This vector integrates the intrinsic features of v with structural information gathered from its multi-hop neighborhood.

▷ Shared-Specific MoE Architecture. Database configuration debugging involves processing highly distinct semantic spaces. For instance, diagnosing performance anomalies relies on analyzing quantitative metrics (e.g., CPU load) and their correlations, whereas resolving runtime errors hinges on matching discrete log patterns. A monolithic policy network struggles to model these divergent feature distributions. To address this, we propose a *Shared-Specific Mixture-of-Experts* (SS-MoE) architecture that separates universal knowledge from domain-specific reasoning. The network comprises three *Specific Experts* (MLP_k) and one *Shared Expert* (MLP_{sh}), collaborating to generate the final representation:

- **① Performance Diagnosis Expert:** Interprets numerical system metrics together with related configuration signals (e.g., innodb_buffer_pool_size vs. IOPS) to model quantitative dependencies and resource bottlenecks.

- **② Error Fix Expert:** Matches user-reported symptoms to discrete failure patterns (e.g., "PKIX path building failed") for root-cause identification in unstructured logs.
- **③ Parameter Setting Expert:** Models logical constraints among knobs (e.g., `max_connections` limiting `open_files_limit`) to ensure theoretically valid recommendations.
- **④ Shared Expert (Generalist):** Encodes universal diagnostic principles and entity relationships applicable across all tasks.

The three specialized experts are designed to address the primary configuration debugging intents, which are derived from extensive DBA community summaries and previous work [3]. This architecture is theoretically driven by the task nature: diverse debugging scenarios span distinct semantic spaces yet share foundational database concepts. Isolating this universal knowledge into a Shared Expert enables Specific Experts to focus exclusively on unique diagnostic features. This decoupling prevents negative interference among heterogeneous tasks and substantially enhances representational capacity. Consequently, our query-conditioned routing overcomes the limitation of GNNs, empowering AGRL to effectively capture the divergent semantic spaces of heterogeneous entities.

To enforce task-adaptivity, we employ a *Gating Network* that learns to route the latent representation z_v to the most relevant experts based on the query h_q . The routing weights $g \in \mathbb{R}^3$ are computed as:

$$g = \text{Softmax}(\mathbf{W}_g[z_v, h_q]) \quad (4)$$

The final output feature $\mathbf{o}_v$ is obtained via a weighted fusion of the specific experts, augmented by the shared expert's output:

$$\mathbf{o}_v = \text{MLP}_{sh}(z_v) + \sum_{k=1}^3 g_k \cdot \text{MLP}_k(z_v) \quad (5)$$

LEMMA 4.1 (GRADIENT DECOUPLING). *The gating mechanism scales parameter updates proportional to gate values. For task-irrelevant experts ($g_k \rightarrow 0$), the gradient norm vanishes ($\|\nabla_{\theta_k} \mathcal{L}\| \rightarrow 0$), effectively preventing negative transfer.*

This analysis supports SS-MoE specialization. By isolating updates, it guides experts (e.g., Error Fix vs. Performance) toward task-specific subspaces with less cross-task interference. The proof of Lemma 4.1 is provided in the technical report [17].

► **Dual-Stage Decision Mechanism.** Equipped with the task-adaptive embeddings from the encoder, the retriever executes the graph traversal. As outlined in our MDP formulation, this traversal is explicitly anchored by the search entry point (v_c) and constrained by a maximum subgraph size (n_{max}), both dynamically designated by the Debugging Agent (Section 4.3). Starting from v_c , the retriever incrementally expands the diagnostic subgraph C at each step t through a *Dual-Stage Decision Mechanism*. This mechanism breaks down the complex retrieval task into two sequential components: the *Scope Controller* and the *Vertex Selector*.

▷ *Scope Controller.* The Scope Controller specifies the k -hop neighbors of v_c as the candidate vertices. Its decision components are defined as follows:

- *State S^{sc} .* The state at step t includes the embedding of the question h_q , the current subgraph C , and embeddings of all vertices and edges in the graph. The state is therefore:

$$s_t^{sc} = (h_q, C, \{h_v\}_{v \in \mathcal{V}}, \{h_e\}_{e \in \mathcal{E}}) \quad (6)$$

- *Action $\mathcal{A}^{sc}$.* The action is a positive integer a_t^{sc} with $1 \leq a_t^{sc} \leq K$, indicating that the controller selects the a_t^{sc} -hop neighbors as the candidate scope. Specifically, the scope controller aggregates

the center vertex's representations across different hops, $\mathbf{O}_{v_c} = \{\{^{(k)}\bar{\mathbf{o}}_{v_c}\}_{k=0}^K\}$. Here, $\{^{(k)}\bar{\mathbf{o}}_{v_c}\}$ is the mean-pooled vector of the outputs ($\mathbf{o}_v$) for all vertices located exactly k hops away from v_c . Note that because the initial entity feature $\mathbf{x}_v$ (Eq. 1) dynamically fuses the current subgraph C , this state C is intrinsically encoded within the upstream outputs and $\mathbf{O}_{v_c}$. The controller projects $\mathbf{O}_{v_c}$ to a probability distribution $\mathbf{p}_{sc} \in \mathbb{R}^{K+1}$:

$$\mathbf{p}_{sc} = \text{Softmax}(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{O}_{v_c})) \quad (7)$$

Here, $\mathbf{W}_1$ and $\mathbf{W}_2$ denote the learnable weight matrices.

▷ Vertex Selector. The Vertex Selector chooses one vertex from the scope determined by the Scope Controller.

- *State $\mathcal{S}^{vs}$* . Compared to the scope controller, the state at step t for the vertex selector involves an additional element, i.e., the executed action a_t^{sc} :

$$s_t^{vs} = (\mathbf{h}_q, C, \{\mathbf{h}_v\}_{v \in \mathcal{V}}, \{\mathbf{h}_e\}_{e \in \mathcal{E}}, a_t^{sc}) \quad (8)$$

- *Action $\mathcal{A}^{vs}$* . The action is a vertex $a_t^{vs} = v'$ such that $v' \in \mathcal{V}_{a_t^{sc}}$, where $\mathcal{V}_{a_t^{sc}}$ is the set of unvisited candidate vertices located exactly a_t^{sc} hops away from v_c . This means that v' is selected and added to the result subgraph C . Specifically, the vertex selector takes embeddings of candidate vertices as input, $\mathbf{O}_{\mathcal{V}_{a_t^{sc}}} = \{\mathbf{o}_v\}_{v \in \mathcal{V}_{a_t^{sc}}}$. This head outputs a probability distribution vector $\mathbf{p}_{vs} \in \mathbb{R}^{|\mathcal{V}_{a_t^{sc}}|}$:

$$\mathbf{p}_{vs} = \text{Softmax}(\mathbf{W}_3 \text{ReLU}(\mathbf{W}_2 \mathbf{O}_{\mathcal{V}_{a_t^{sc}}})) \quad (9)$$

▷ Reward & State Transition. To drive sequential retrieval and guide policy learning, we define the reward signals and environment dynamics:

- *Reward $\mathcal{R}$* . As intermediate search radii lack direct labels, the scope selection a_t^{sc} yields a step-wise reward of 0. A delayed reward is strictly issued after the vertex selection. We formulate a sequence-level hybrid reward r_t^{vs} to simultaneously evaluate the local semantic relevance and global task utility of the retrieved subgraph, which is detailed in Eq. 10 (Section 5).
- *Transition $\mathcal{T}$* . The transition is deterministic. Executing vertex selection a_t^{vs} adds the chosen vertex v' to the subgraph ($C_{t+1} = C_t \cup \{v'\}$). This update modifies the subgraph in Eq. 1, requiring candidate embeddings to be recomputed for the next state $\mathcal{S}_{t+1}$.

LEMMA 4.2 (VARIANCE BOUND TIGHTENING). *Factorizing the policy into scope and conditional vertex selection tightens the worst-case gradient variance bound by decoupling the search radius decision from entity selection, ensuring optimization stability.*

In a unified action space, dense neighborhoods statistically overshadow sparse ones. This factorization ensures the actor evaluates each scope's relevance fairly, preventing decisions from being biased solely by candidate volume. The proof of Lemma 4.2 is provided in the technical report [17].

► **Retrieval Workflow.** Algorithm 1 summarizes the complete workflow of the reinforced graph retriever. The algorithm begins by initializing the subgraph with the center vertex v_c and computing the required embeddings (lines 1–3). It then constructs query-aware representations for K-hop neighbors of v_c , incorporating multi-hop neighborhood information and the current subgraph context (lines 4–13). These representations are encoded by the Transformer layer and routed through the multi-expert module according to the query type. Next, the Scope Controller determines the appropriate hop size (lines 14–17), after which the Vertex Selector identifies a vertex from the resulting candidate set (lines 18–20). The selected vertex is added to the growing subgraph (line 20), and the process repeats until the predefined retrieval size is reached.

Algorithm 1: Subgraph Retrieval**Input:** Knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, question q , center vertex v_c , hop limit K , the number of vertices n_{max} **Output:** Retrieved subgraph C

```

1  $C \leftarrow \{v_c\}; t \leftarrow 0$ ; Compute question embedding  $\mathbf{h}_q$ ;
2 Compute initialized embeddings  $\mathbf{x}_v$  for  $v \in \mathcal{V}$ ; // Eq. (1)
3 while  $|C| \leq n_{max}$  do
4   for  $v \in \mathcal{N}_K(v_c)$  do
5     Compute current subgraph embedding  $\tilde{\mathbf{x}}_H$ ;
6      $\mathbf{S}_v \leftarrow \{^0\tilde{\mathbf{x}}_v, ^1\tilde{\mathbf{x}}_v, \dots, ^K\tilde{\mathbf{x}}_v, \mathbf{h}_q, \tilde{\mathbf{x}}_H\}$ ;
7      $\mathbf{H}_v^{(0)} \leftarrow \mathbf{S}_v \mathbf{W}_0 + \mathbf{P}$ ; // Eq. (2)
8     for  $l = 0, \dots, L - 1$  do
9        $\tilde{\mathbf{H}}_v^{(l)} = \text{MHA}(\mathbf{H}_v^{(l)}) + \mathbf{H}_v^{(l)}$ ; // Eq. (3)
10       $\mathbf{H}_v^{(l+1)} = \text{FFN}(\tilde{\mathbf{H}}_v^{(l)}) + \tilde{\mathbf{H}}_v^{(l)}$ ; // Eq. (3)
11       $\mathbf{z}_v \leftarrow ^0\mathbf{H}_v^{(L)}$ ;
12      // Mixture-of-Expert
13       $\mathbf{g} = \text{Softmax}(\mathbf{W}_g[\mathbf{z}_v, \mathbf{h}_q])$ ; // Eq. (4)
14       $\mathbf{o}_v = \text{MLP}_{sh}(\mathbf{z}_v) + \sum_{k=1}^3 g_k \cdot \text{MLP}_k(\mathbf{z}_v)$ ; // Eq. (5)
15      // =====Stage 1: Scope Controller=====
16       $\mathbf{O}_{v_c} \leftarrow \{^{(k)}\tilde{\mathbf{o}}_{v_c}\}_{k=0 \text{ to } K}$ ;
17       $\mathbf{p}_{sc} \leftarrow \text{Softmax}(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{O}_{v_c}))$ ; // Eq. (7)
18       $a_t^{sc} \leftarrow \arg \max_k \mathbf{p}_{sc}[k]$ ; // Action of SC
19       $\mathcal{V}_{a_t^{sc}} \leftarrow a_t^{sc}$ -hop neighbors of  $v_c$ ;  $\mathcal{V}_{a_t^{sc}} \leftarrow \mathcal{V}_{a_t^{sc}} - C$ ;
20      // =====Stage 2: Vertex Selector=====
21       $\mathbf{p}_{vs} \leftarrow \text{Softmax}(\mathbf{W}_3 \text{ReLU}(\mathbf{W}_2 \mathbf{O}_{\mathcal{V}_{a_t^{sc}}}))$ ; // Eq. (9)
22       $a_t^{vs} \leftarrow \arg \max_i \mathbf{p}_{vs}[i]$ ; // Action of VS
23       $C \leftarrow C \cup a_t^{vs}; t \leftarrow t + 1$ ;
24 return  $C$ ;

```

▷ *Time Complexity.* The retrieval cost is dominated by the *Task-Adaptive Encoder* and *SS-MoE Policy* operating on the local candidate frontier $\mathcal{N}_K(v_c)$. For each candidate, the inference involves an L -layer Transformer with hidden dimension d . Since the input sequence length is negligible compared to d , the per-vertex computation is bounded by $O(L \cdot d^2)$. Consequently, the total complexity is $O(n_{max} \cdot |\mathcal{N}_K(v_c)| \cdot L \cdot d^2)$.

4.3 Multi-round Debugging Agent

► **Overview.** Acting as the system's decision-making core, the *Multi-round Debugging Agent* emulates a human DBA's workflow by replacing passive, single-pass generation with an active "Reason-Act-Reflect" loop. This iterative approach allows the agent to drive the diagnosis through three key mechanisms:

- *Step-wise Decomposition.* It breaks down complex, symptom-based user queries into fine-grained, actionable sub-questions to guide the reasoning process.
- *Schema-Aware Graph Anchoring.* Leveraging the database-specific schema, the agent explicitly defines the search entry point by specifying the target vertex type (e.g., *Configuration Knobs*

Prompt for Debugging Agent

You are a database assistant who uses the graph retrieval tool to answer complex questions and recommend database knobs (parameters). You are encouraged to decompose the complex question into easier parts and perform search step by step.

INSTRUCTIONS:

- 1. Before you start a new retrieval request, include your reasoning process using `<reasoning>` and `</reasoning>` tags.
- 2. Each time you answer, you can only initiate one retrieval request.
- 3. After receiving the retrieval result for round:
 - If the results are sufficient, formulate your final answer;
 - If more information is needed, perform additional retrievals.
- 4. The graph retrieval tool will search relevant information on a knowledge graph. To use this tool, respond with a simple JSON in this format:


```
{
  "sub_question": "The sub-question for retrieval",
  "start_vertex_type": "The type of starting vertex
    (Choose one from six options.):
    (1). Configuration Knobs Vertex.
    (2). Performance Metrics Vertex.
    (3). System Concepts Vertex.
    (4). Error & Failure Vertex.
    (5). Operation & Workload Patterns Vertex.
    (6). Q&A Vertex."
  "start_vertex_content": "Content of the desired starting vertex."
  "subgraph_size": "Number of vertices in the retrieved subgraph"
}
```
- 5. You may perform up to 5 retrieval iterations before a final answer.
- 6. In the final answer, you must recommend some related database knobs and their values for solving the question using `<knob>` and `</knob>` tags, such as


```
<knob>[knob1: value1, knob2: value2, ...]</knob>.
```

QUESTION: {question}

Fig. 5. Prompt for the multi-round debugging agent

Vertex or *Error & Failure Vertex*) and expected content. This structural constraint directs the retriever to initiate traversal from highly relevant entities, narrowing the search space.

- **Adaptive Search Scope Control.** The agent dynamically regulates the search scope based on task complexity. It controls the overall retrieval **breadth** by specifying the macro subgraph size (n_{max}) for each query (which serves as the global termination threshold for the retriever's step-wise Scope Controller detailed in Section 4.2), and manages reasoning **depth** by autonomously determining the number of iterative rounds needed.

These mechanisms enable the agent to acquire task-relevant structural context iteratively from the knowledge graph, providing the necessary insights to resolve complex DBMS configuration issues. This decision-making process is further optimized via *Reinforcement Learning* (see Section 5).

► **Instruction Design.** To operationalize these autonomous behaviors, we design a system prompt (Figure 5) as a rigorous interaction protocol. Bypassing unstructured dialogue, the prompt mandates the agent to output a standardized JSON object, from which AGRL parses four critical parameters to configure the retriever:

- **sub_question (Search Intent):** This field captures the decomposed sub-question (e.g., "Check implicit defaults for connection timeouts"). It isolates the specific diagnostic goal for the current step, preventing the retrieval of irrelevant context associated with the broad initial user query.

Algorithm 2: Agentic Configuration Debugging

Input: User's question q , knowledge graph $\mathcal{G}$, reinforced graph retriever $\mathcal{T}$, multi-round debugging agent $\mathcal{A}$, round limit r_{max}

Output: Final answer for question q

```

1  $nextRound \leftarrow True$ ; // Signal for next round
2  $i \leftarrow 0$ ; // Iteration counter
3  $ctx \leftarrow$  Initial prompt for  $q$ ; // Prompt context for agent
4  $ans \leftarrow \emptyset$ ; // Final answer
5 while  $nextRound$  and  $i \leq r_{max}$  do
6    $s \leftarrow \mathcal{A}.GENERATEANSWER(ctx)$ ; // Agent reasoning
7    $(q_{sub}, t_{sv}, c_{sv}, n) \leftarrow$  parse the following JSON fields: sub_question, start_vertex_type,
   start_vertex_content, and subgraph_size;
8   if  $(q_{sub}, t_{sv}, c_{sv}, n)$  is empty then
9      $ans \leftarrow s$ ; break;
10   $v_c \leftarrow \arg \max_{v \in \mathcal{V}_{t_{sv}}} (\text{sim}(\mathbf{h}_{c_{sv}}, \mathbf{h}_v))$ ;
11   $C \leftarrow \mathcal{T}.\text{RETRIEVAL}(\mathcal{G}, q_{sub}, v_c, n)$ ; // Algorithm 1
12   $ctx \leftarrow ctx \cup C$ ;
13   $i \leftarrow i + 1$ ;
14 if  $ans$  is empty then
15    $ctx \leftarrow ctx \cup$  "Give the final answer.";
16    $ans \leftarrow \mathcal{A}.GENERATEANSWER(ctx)$ ;
17 return  $ans$ ;
```

- **start_vertex_type & start_vertex_content (Graph Anchors):** These two fields jointly determine the search entry vertex (v_c) by strictly selecting from six pre-defined vertex schemas (e.g., *Configuration Knobs Vertex*) and specifying target content.
- **subgraph_size (Breadth Control):** This parameter specifies the number of vertices to retrieve in the subgraph. It acts as a "breadth control," allowing the agent to dynamically expand the search scope for broad exploration or contract it for precise verification.

► **Debugging Workflow.** The agent operates within the "Reason-Act-Reflect" loop (Algorithm 2). The agent begins by initializing the control flags, the iteration counter, and the context used to interact with the LLM-based reasoning module (lines 1–4). It then enters an iterative reasoning–retrieval loop, where in each round the agent generates a reasoning step wrapped in `<reasoning>` tags and produces a JSON-formatted retrieval request when additional graph information is needed (line 6). The request is then systematically parsed to extract the following four control fields needed for graph retrieval: `sub_question`, `start_vertex_type`, `start_vertex_content`, and `subgraph_size` (line 7). If no retrieval request is issued, the agent has collected sufficient information and terminates the loop with the current reasoning output as the final answer (lines 8–9). When a retrieval request is present, the agent first selects the most relevant center vertex based on the sub-question semantics (line 10). It then calls the reinforced graph retriever to obtain a task-relevant subgraph (line 11). The retrieved subgraph is appended to the agent's context to support subsequent reasoning steps (line 12). The iteration counter is incremented, and the loop continues until the maximum number of retrieval rounds is reached (line 13). After the loop, if the agent has not yet produced a final answer, it explicitly instructs the LLM to generate one and extracts the recommended knob settings as required by the prompt design (lines 14–16). The final answer is then returned to the user (line 17).

5 Training Strategy

Reinforcement Learning (RL) bridges static graph architectures and dynamic debugging requirements. By framing retrieval and reasoning as sequential decision-making, RL enables the retriever to adaptively select vertices and empowers the agent to autonomously navigate the diagnostic trajectory. However, directly coupling them introduces a **cold-start interdependency dilemma**: under-trained modules yield poor mutual supervision, generating irrelevant contexts or vague instructions.

To resolve this, we propose a progressive three-phase curriculum strategy. Phase I pre-trains the retriever via *Proximal Policy Optimization* (PPO) [23] to navigate the discrete traversal space. Phase II fine-tunes the agent using *Group Relative Policy Optimization* (GRPO) [8, 24], which efficiently aligns the LLM for multi-step operations without the prohibitive memory cost of a critic model. Finally, Phase III executes *iterative joint refinement*, co-training both modules through a shared task-success reward to promote mutual adaptation and improve overall debugging performance.

5.1 Phase I: Graph Retriever Initialization

Phase I aims to initialize the reinforced graph retriever's fundamental graph traversal capabilities across diverse query types, rather than directly learning final debugging solutions.

► **Topic-Aware Task Generation.** To align the training data with the SS-MoE architecture of reinforced graph retriever, we explicitly construct a labeled dataset that maps sub-questions to expert domains. We employ an LLM to decompose each historical user query q into a set of labeled sub-questions $S_q = \{(q_i^{sub}, c_i)\}$. Here, $c_i \in \{\text{Performance, Error, Parameter}\}$ corresponds to the expert domains defined in Section 4.2.

► **Rewards.** To guide the learning process, we design a hybrid reward for reinforced graph retriever.

$$r_t^{vs} = \begin{cases} \text{Sim}(\mathbf{h}_{v'}, \mathbf{h}_q), & |C_t| \bmod 5 \neq 0, \\ \text{Sim}(\mathbf{h}_{v'}, \mathbf{h}_q) + \text{LLM}(C_t, q), & |C_t| \bmod 5 = 0, \end{cases} \quad (10)$$

where $\mathbf{h}_{v'}$ is the textual embedding of the selected vertex $v' \in \mathcal{V}_{\mathcal{A}_t^{\text{sc}}}$ (chosen from the dynamic candidate scope defined in Section 4.2), C_t denotes the current retrieved subgraph constructed up to step t , $\text{Sim}(\mathbf{h}_{v'}, \mathbf{h}_q)$ represents the cosine similarity, and $\text{LLM}(C_t, q) \in [1, 10]$ is the LLM-generated utility score for whether the current subgraph helps solve question q . This hybrid reward is shared by the Scope Controller and Vertex Selector, triggered solely upon vertex selection to jointly optimize the dual-stage mechanism.

This hybrid design concurrently balances training stability and task utility. The cosine similarity supplies a dense, continuous signal to maintain semantic alignment at every step. Conversely, the LLM feedback offers a sparse, high-level evaluation, ensuring retrieved subgraphs genuinely resolve downstream issues. Crucially, executing this LLM evaluation conditionally via the 'mod 5' operator constitutes a vital efficiency trade-off. Invoking the LLM at every step introduces prohibitive computational overhead during RL training, whereas periodic evaluation reliably steers long-term utility without bottlenecking training throughput.

► **Optimization via PPO.** We optimize the policy using the PPO clipped objective to maintain training stability:

$$\mathcal{L}_{PPO} = \mathbb{E}_t [\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)] \quad (11)$$

where $r_t(\theta)$ denotes the probability ratio and ϵ is the clipping parameter. This clipping mechanism prevents degenerate traversal loops and keyword over-optimization, ensuring steady policy convergence.

Training progresses by sequentially unlocking model components: (1) **Generalist**: Deactivating specific experts and gating, we exclusively train the shared backbone (encoder and shared expert) to establish universal feature alignment. (2) **Specialists**: Freezing the shared parameters, we activate specific experts via label-based hard routing to capture domain-specific nuances as feature residuals. (3) **Gating**: Unfreezing the entire encoder, we activate the gating network. Guided strictly by retrieval rewards, the router learns to dynamically dispatch queries to the most effective expert.

5.2 Phase II: Debugging Agent Alignment

Following Phase I, the reinforced retriever acts as a frozen environment. Phase II then fine-tunes the debugging agent for effective interaction, utilizing a composite reward and GRPO to optimize the policy for accurate reasoning and retrieval actions.

► **Rewards**. We design a composite reward function for debugging agent: $R = R_{task} + R_{format}$.

- **Format Reward** (R_{format}). To ensure the agent generates executable search instructions and parsable answers, we introduce a strict binary format reward. Let o_q denote the agent's output for a question q , and $\mathbb{I}_{valid}(o_q)$ be an indicator function that equals 1 if o_q strictly adheres to the required protocol (i.e., a valid JSON schema for retrieval requests and properly formatted `<knob>` and `<reasoning>` tags in the final answer), and 0 otherwise. Thus, the format reward is given by:

$$R_{format}(o_q) = \mathbb{I}_{valid}(o_q) \quad (12)$$

This binary signal effectively filters out invalid structural failures, guiding the agent to adhere to the correct interface protocols before optimizing for solution accuracy.

- **Success Reward** (R_{task}). This outcome-oriented reward measures the utility of the final recommendation for a specific training query q . Let $\hat{\mathcal{K}}_{o_q}$ denote the set of knobs predicted by the agent in its final answer, and $\mathcal{K}_q^*$ denote the ground-truth knob set associated with q . We calculate the instance-level F1-score as:

$$R_{task}(o_q) = F1(\hat{\mathcal{K}}_{o_q}, \mathcal{K}_q^*) = \frac{2 \cdot |\hat{\mathcal{K}}_{o_q} \cap \mathcal{K}_q^*|}{|\hat{\mathcal{K}}_{o_q}| + |\mathcal{K}_q^*|} \quad (13)$$

where the reward evaluates to 0 if the denominator is 0. This sparse feedback implicitly guides the agent to optimize its complete reasoning trajectory, encompassing strategic decisions on the number of search iterations, query formulation, and execution termination, thereby enhancing the quality of the final solution. Notably, explicit knob values are excluded from this reward formulation. Because the exact execution environments for many historical forum issues are ambiguous, establishing reliable gold-standard values for all questions is impractical.

► **Optimization via GRPO**. Optimizing LLMs with standard PPO incurs prohibitive computational overhead, as it requires a value network matching the policy model's size. To resolve this, we employ GRPO, which completely eliminates explicit critics.

For each query q , we sample a group of M independent trajectories $\{y_1, y_2, \dots, y_M\}$ from the current policy π_θ . We then compute the advantage A_m for each trajectory y_m by normalizing its reward against the group statistics: $A_m = \frac{r_m - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r}) + \epsilon}$, where $\mathbf{r} = \{R(y_1), \dots, R(y_M)\}$ is the set of composite rewards for the group. The policy is updated by maximizing the GRPO objective:

$$\mathcal{J}_{GRPO} = \mathbb{E}_q \left[\frac{1}{M} \sum_{m=1}^M \left(\sum_t A_m \frac{\pi_\theta(y_{m,t}|q, y_{m,<t})}{\pi_{ref}(y_{m,t}|q, y_{m,<t})} - \beta_{KL} D_{KL} \right) \right] \quad (14)$$

Table 1. Dataset statistics

Datasets	# Queries	Knowledge Base		Mode
		# Manual	# Q&A Pairs	
MySQL SO	174	3506	1632	NL
MySQL Forum	49	3506	1632	NL
PG SO	57	756	402	NL
MySQL Run	100	3506	1632	Runnable
MySQL Val	100	3506	1632	Value-aware

5.3 Joint Optimization

Having stabilized the modules individually, we advance to the *Joint Optimization* phase. This step transitions the isolated components into a fully coordinated framework, fostering mutual adaptation for complex reasoning.

► **Mutual Adaptation.** Unfreezing the retriever enables dynamic interactions under the shared task-success reward $R_{task}(q)$ (Eq. 13). This unified objective shifts the retriever from matching local semantics to maximizing global diagnostic utility. Concurrently, the agent refines its querying strategy to exploit the retriever’s updated capabilities, establishing a robust closed-loop system.

► **Asynchronous Updates.** To mitigate non-stationarity during co-training, we employ an asynchronous schedule: the agent updates at every iteration, whereas the retriever updates only every T_{ret} iterations. Treating the retriever as a slow-evolving environment introduces a structural asymmetry that stabilizes the agent’s observation space, reduces destructive policy oscillation, and encourages more stable cooperative updates.

6 Experiments

► **Datasets.** We conduct experiments on real-world datasets under three evaluation modes. The statistics of these datasets are provided in Table 1. MySQL has 582 configurable knobs, while PostgreSQL has 202. More results can be found in the technical report [17].

▷ **NL Mode.** The NL mode is designed to evaluate the accuracy of knob recommendations. We utilize three datasets for this mode: **MySQL StackOverflow (MySQL SO)**¹, **MySQL Forum**², and **PostgreSQL StackOverflow (PG SO)**³. A data instance in dataset is represented as $\langle q, \mathcal{K}_q \rangle$, where q is a natural language question and $\mathcal{K}_q$ is the ground-truth knob configuration extracted from accepted answers or comments by previous SOTA work [3]. For each test question q , the method outputs a predicted knob set $\mathcal{K}_a$, which is compared against the ground truth $\mathcal{K}_q$. We evaluate configuration debugging using Precision (P), Recall (R), and F1-score: $F1 = \frac{2 \cdot P \cdot R}{P + R} = \frac{2 \cdot |\mathcal{K}_a \cap \mathcal{K}_q|}{|\mathcal{K}_a| + |\mathcal{K}_q|}$.

▷ **Runnable Mode.** The Runnable mode assesses if predicted configurations resolve reproducible runtime issues on a local testbed (MySQL 5.7). The **MySQL Run** dataset consists of 100 queries carefully selected by three database experts from MySQL SO and MySQL Forum based on two criteria: (1) explicit problem and solution descriptions, and (2) reproducible DBMS runtime environments. It covers diverse anomalies: I/O Bottlenecks (28%), Lock/Deadlocks (22%), Connections (18%), Memory/OOM (14%), Replication (11%), and Latency (7%). We first establish a *healthy baseline* by applying the ground-truth configurations to record the optimum recovered performance. The evaluation rigorously follows four steps: (1) **Fault Injection:** Initialize flawed settings and execute the workload. (2) **Knob Application:** Apply LLM’s predictions (using SET GLOBAL for

¹MySQL StackOverflow dataset: <https://www.kaggle.com/datasets/stackoverflow/stackoverflow>

²MySQL Forum dataset: <https://forums.mysql.com>

³PostgreSQL StackOverflow dataset: <https://www.kaggle.com/datasets/stackoverflow/stackoverflow>

Table 2. Accuracy comparison (R: Recall, P: Precision, F1: F1-score)

Model	Method	MySQL SO			MySQL Forum			PG SO			MySQL Run SuccessRate	MySQL Val			Avg +/-
		R	P	F1	R	P	F1	R	P	F1		R	P	F1	
GPT-5	LLM	0.280	0.185	0.196	0.140	0.045	0.051	0.253	0.126	0.131	0.39	0.458	0.341	0.381	-68.14%
	GraphRAG [6]	<u>0.584</u>	0.235	0.359	0.552	0.454	0.416	0.402	0.290	0.366	0.61	0.650	0.535	0.576	-34.31%
	LightRAG [9]	0.518	0.254	0.364	0.492	0.302	0.348	0.373	0.328	0.359	0.51	0.538	0.413	0.459	-42.49%
	ToG-2 [19]	0.562	0.232	0.338	0.524	0.469	0.422	0.426	0.280	0.314	0.58	0.613	0.460	0.512	-37.53%
	KET-RAG [11]	0.530	0.468	0.509	0.546	0.492	0.437	0.424	0.372	0.358	0.66	0.672	0.516	0.568	-27.88%
	Andromeda [3]	0.546	0.449	0.476	0.543	0.471	0.394	0.446	0.381	0.366	0.67	0.691	0.531	0.585	-28.08%
	AGRL-S	0.654	0.569	0.615	0.678	0.620	0.529	0.591	0.484	0.473	0.85	0.837	0.625	0.694	-9.61%
	AGRL	0.712	0.581	0.640	0.735	0.625	0.675	0.720	0.615	0.663	0.87	0.852	0.658	0.721	-
Llama-8B	LLM	0.059	0.022	0.032	0.072	0.019	0.028	0.048	0.022	0.027	0.22	0.325	0.237	0.263	-84.04%
	LLM-SFT	0.097	0.097	0.099	0.120	0.131	0.125	0.080	0.136	0.124	0.29	0.337	0.281	0.300	-73.56%
	GraphRAG [6]	0.269	0.145	0.188	0.292	0.198	0.236	0.249	0.121	0.163	0.37	0.435	0.349	0.379	-60.03%
	LightRAG [9]	0.267	0.179	0.214	0.167	0.107	0.130	0.260	0.178	0.211	0.31	0.401	0.335	0.359	-63.04%
	ToG-2 [19]	0.276	0.168	0.209	0.268	0.158	0.199	0.244	0.149	0.186	0.33	0.409	0.334	0.358	-61.13%
	KET-RAG [11]	0.284	<u>0.288</u>	<u>0.286</u>	0.142	<u>0.244</u>	0.179	0.113	0.149	0.129	0.43	0.477	0.377	0.410	-57.70%
	Andromeda [3]	0.328	0.175	0.210	0.448	0.235	0.259	0.373	0.246	0.268	0.42	0.501	0.404	0.437	-49.40%
	AGRL-S	0.463	0.297	0.354	0.557	0.370	0.366	0.491	0.385	0.372	0.53	0.585	0.456	0.500	-32.20%
	AGRL	0.730	0.496	0.599	0.712	0.581	0.629	0.751	0.588	0.655	0.74	0.729	0.573	0.627	-
Qwen3-8B	LLM	0.023	0.022	0.018	0.039	0.028	0.030	0.044	0.014	0.019	0.19	0.301	0.277	0.282	-85.67%
	LLM-SFT	0.107	0.128	0.114	0.145	0.154	0.145	0.085	0.174	0.132	0.25	0.327	0.264	0.287	-73.10%
	GraphRAG [6]	0.367	0.136	0.198	0.353	0.218	0.269	0.299	0.112	0.163	0.48	0.511	0.424	0.454	-54.43%
	LightRAG [9]	0.350	0.141	0.201	0.333	0.131	0.201	0.300	0.132	0.179	0.35	0.416	0.314	0.349	-61.20%
	ToG-2 [19]	0.352	0.154	0.214	0.392	0.176	0.242	0.218	0.152	0.178	0.40	0.482	0.408	0.432	-56.28%
	KET-RAG [11]	0.279	0.191	0.227	0.155	<u>0.257</u>	0.193	0.125	0.165	0.141	0.31	0.363	0.291	0.314	-64.49%
	Andromeda [3]	0.286	<u>0.252</u>	<u>0.253</u>	0.246	0.187	0.187	0.280	<u>0.214</u>	<u>0.219</u>	0.46	0.532	0.413	0.450	-54.12%
	AGRL-S	0.486	0.335	0.360	0.525	0.419	0.422	0.474	0.375	0.396	0.58	0.595	0.469	0.512	-30.71%
	AGRL	0.636	0.521	0.615	0.695	0.512	0.623	0.786	0.634	0.681	0.83	0.827	0.585	0.666	-
Qwen3-14B	LLM	0.025	0.024	0.017	0.037	0.029	0.030	0.045	0.014	0.020	0.19	0.310	0.261	0.279	-86.46%
	LLM-SFT	0.156	0.152	0.154	0.169	0.171	0.170	0.108	0.187	0.159	0.35	0.442	0.374	0.394	-67.70%
	GraphRAG [6]	0.346	0.141	0.200	0.415	0.198	0.268	0.326	0.122	0.178	0.51	0.537	0.410	0.453	-56.55%
	LightRAG [9]	0.317	0.117	0.171	0.433	0.142	0.214	0.280	0.111	0.159	0.38	0.470	0.370	0.403	-62.28%
	ToG-2 [19]	0.331	0.146	0.201	0.418	0.172	0.244	0.295	0.128	0.177	0.36	0.444	0.345	0.380	-61.26%
	KET-RAG [11]	0.271	0.220	0.250	0.159	0.247	0.197	0.132	0.173	0.150	0.37	0.449	0.356	0.388	-63.37%
	Andromeda [3]	0.301	0.250	0.241	0.275	0.206	0.208	0.294	0.239	0.243	0.42	0.484	0.408	0.433	-56.82%
	AGRL-S	0.524	0.372	0.395	0.566	0.413	0.473	0.487	0.414	0.424	0.63	0.672	0.525	0.576	-30.10%
	AGRL	0.730	0.589	0.693	0.815	0.572	0.673	0.794	0.678	0.686	0.88	0.848	0.607	0.692	-

* For each metric, the best-performing method is highlighted in **bold**. Among all compared baselines, the best result is additionally underlined.

dynamic knobs, and editing `my.cnf` for static ones). (3) **Reset**: Restart the DBMS daemon to ensure state changes take effect. (4) **Replay**: Re-execute the workload. Performance is computed as $\text{SuccessRate} = \frac{1}{|Q_{\text{test}}|} \sum_{q \in Q_{\text{test}}} \text{Solved}(q)$. $\text{Solved}(q) = 1$ if (1) the predicted configuration removes the target error or (2) it meets the query’s target performance or the healthy-baseline tolerance (i.e., $\geq 90\%$ throughput or $\leq 110\%$ latency).

▷ **Value-aware Mode**. This mode evaluates the correctness of predicted *values* using **MySQL Val**, which uses the same 100 queries as **MySQL Run**. **Annotation Protocol**: Three database experts independently annotated valid values or ranges for the ground-truth knobs. Detailed annotation guidelines are provided in the technical report [17]. To ensure reliability, values were only accepted as “ground truth” if they successfully resolved the issue (meeting the four-step criteria of the Runnable mode). Disagreements are resolved through discussion and majority voting. A predicted knob-value item is correct if its name matches a ground truth knob and its value satisfies one of the following: (1) **Unit-aware Exact Match**: Numerically equivalent after conversion (e.g., 1G equals 1024M). (2) **Range Match**: Falls within the validated continuous interval. (3) **Categorical Match**: Matches a valid category after case and alias normalization (e.g., row equals ROW). Precision, Recall, and F1-score are then computed over these knob-value items.

► **Competitors**. We compare AGRL against the following methods: a state-of-the-art vector-based approach for DBMS configuration debugging and several representative GraphRAG-based methods. (1) *Andromeda* [3] is a vector-based RAG framework that retrieves domain knowledge from manuals and historical records using contrastive learning-aligned embeddings, augmented with telemetry analysis for anomaly detection. (2) *GraphRAG* [6] builds an LLM-derived entity

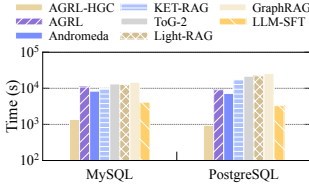


Fig. 6. Offline Graph Construction and Training Time

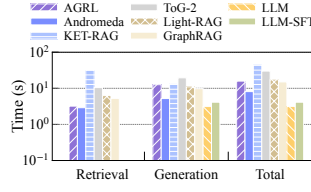


Fig. 7. Online Debugging Time on MySQL SO Dataset

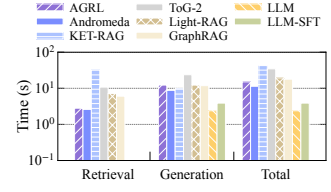


Fig. 8. Online Debugging Time on PostgreSQL SO Dataset

graph with hierarchical community detection and performs query-focused summarization across multi-level communities to support reasoning. (3) *LightRAG* [9] enhances retrieval with a dual-level scheme that indexes entities and relations via an LLM, capturing both local semantics and global structure. (4) *ToG-2* [19] enables multi-round deep reasoning through iterative alternation between KG-guided graph retrieval and document-based context retrieval. (5) *KET-RAG* [11] extracts task-specific key entities as retrieval anchors and dynamically prioritizes entity-conditioned evidence for grounded reasoning over large corpora. (6) *LLM* uses a large language model directly to recommend configuration knobs from a task instruction and failure description, without retrieval or graph reasoning. (7) *LLM-SFT* fine-tunes an LLM on instruction-answer pairs from the training set, where answers are ground-truth knob lists.

► **Implementation.** All experiments are executed on an Ubuntu 20.04.6 LTS server with two Intel Xeon Platinum 8368 CPUs, 1 TB RAM, and eight NVIDIA A100-SXM4-80GB GPUs. For AGRL, we set the graph construction connectivity to $k_{QA} = 5$, cap reasoning at 5 rounds, and retrieve the top-1 subgraph per iteration. Baselines retain their default configurations. We locally deploy open-source models (Llama-8B⁴, Qwen3-8B⁵, Qwen3-14B⁶), alongside GPT-5⁷ as a closed-source reference. Qwen3-8B serves as the default backbone for all efficiency evaluations.

► **Challenge Validation Roadmap.** Our evaluation explicitly revisits Challenges C1–C3 in the Introduction.

C1 is validated through graph construction cost in Exp-2 and graph quality in Exp-10, especially the ablation row “w/o Hybrid Graph (use Global Graph)” in Table 3.

C2 is validated through overall retrieval/debugging accuracy in Exp-1 and retrieval-module ablations in Exp-10, especially the rows “w/o MoE (Shared-only)”, “w/o Shared Expert”, “w/o Dual-stage Decision”, and “w/o task encoder (GAT)” in Table 3.

C3 is validated by Exp-4’s reasoning-round study and Exp-10’s “w/o Agent (AGRL-S)” ablation in Table 3.

6.1 Effectiveness Evaluation

We evaluate AGRL against several baseline methods and AGRL-S, a simplified variant that uses single-round retrieval.

Exp-1. Overall Accuracy Comparison. As shown in Table 2, AGRL delivers strong results across all three evaluation modes. Under the NL Mode, it consistently achieves higher F1-scores on the three corresponding datasets. Evaluated in the Runnable Mode, AGRL effectively resolves reproducible runtime anomalies, attaining a **SuccessRate of 0.88** on the MySQL Run dataset using the Qwen3-14B backbone. This substantial lead extends into the Value-aware Mode on the

⁴Llama-3.1-8B-Instruct model: <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>

⁵Qwen3-8B model: <https://huggingface.co/Qwen/Qwen3-8B>

⁶Qwen3-14B model: <https://huggingface.co/Qwen/Qwen3-14B>

⁷GPT-5 model: <https://openai.com/gpt-5>

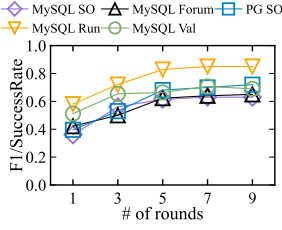


Fig. 9. Effect of Maximum Multi-round Reasoning Limit

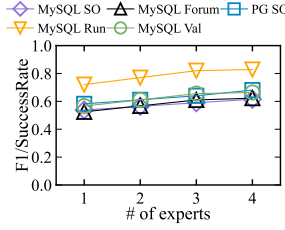


Fig. 10. Effect of Activated Shared-Specific MoE Experts

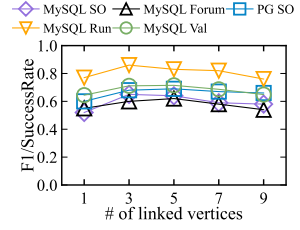


Fig. 11. Effect of Q&A Vertex Connectivity Degree

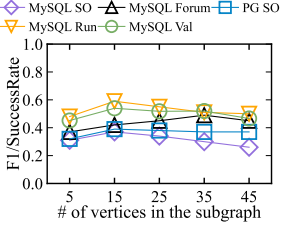


Fig. 12. Effect of Maximum Retrieved Subgraph Size

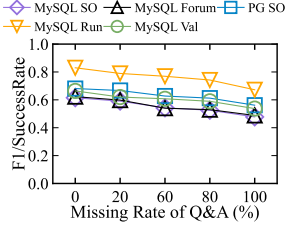


Fig. 13. Effect of Missing Historical Q&A Ratio

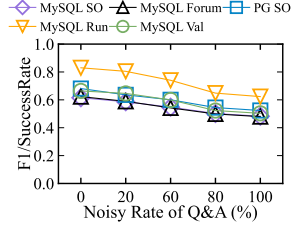


Fig. 14. Effect of Injected Noisy Q&A Ratio

MySQL Val dataset. Since MySQL Val shares the exact queries of the highly curated MySQL Run dataset, metrics here naturally exceed those obtained on historical forum like MySQL SO. However, baselines still fail to predict exact configuration values. AGRL uses graph-guided causal reasoning to accurately infer both the target knobs and their specific value settings, outperforming all baseline methods.

Baseline comparisons emphasize the importance of explicitly modeling structural dependencies. Although Andromeda is specifically optimized for configuration debugging, standard GraphRAG methods frequently outperform it, indicating that purely vector-based RAG lacks essential structural awareness. Nonetheless, graph-based baseline approaches rely on rigid search strategies that either introduce excessive noise or omit crucial context. AGRL mitigates these issues through its MoE-based encoder and dual-stage mechanism, supporting C2's need for adaptive subgraph retrieval. Supported by specialized experts, even the single-round variation (AGRL-S) tends to prioritize task-relevant vertices and yield more useful subgraphs than baseline retrievers. The full AGRL framework builds upon this by tightly coupling a multi-round agent with a reinforced retriever. This architecture supports joint optimization and iterative reasoning, allowing the system to refine search scopes via dynamic feedback and identify causal factors that static methods may miss.

6.2 Efficiency Evaluation

Exp-2: Offline Construction Time. Figure 6 reports the total time for index construction and model training. Because MySQL SO, Forum, and Run share an identical corpus, we present unified results. AGRL's hybrid graph construction (AGRL-HGC) achieves the lowest cost across all datasets. The graph-based baselines construct their graphs or indexes over the entire corpus, including both manuals and Q&A records. In contrast, AGRL restricts LLM extraction to official manuals and injects Q&A pairs only as bridge vertices, validating C1 by preventing prohibitive computational overhead. Even when factoring in model training, AGRL remains highly efficient, substantially outpacing graph-based baselines (e.g., GraphRAG, LightRAG, and ToG-2) and saving approximately 8,000 seconds on MySQL compared to GraphRAG. AGRL also demonstrates efficiency comparable to Andromeda, a vector-based approach burdened by extensive contrastive embedding training.

Table 3. Ablation Study of AGRL (R: Recall, P: Precision, F1: F1-score)

Method	MySQL SO			MySQL Forum			PG SO			MySQL Run	MySQL Val			Avg +/-
	R	P	F1	R	P	F1	R	P	F1	SuccessRate	R	P	F1	
AGRL (Full Model)	0.636	0.521	0.615	0.695	0.512	0.623	0.786	0.634	0.681	0.83	0.827	0.585	0.666	-
w/o Hybrid Graph (use Global Graph)	0.542	0.421	0.510	0.602	0.425	0.531	0.675	0.540	0.588	0.71	0.739	0.561	0.623	-13.30%
w/o Version/Timestamp Metadata	0.601	0.495	0.582	0.658	0.485	0.591	0.745	0.602	0.648	0.79	0.785	0.558	0.635	-5.06%
w/o MoE (Shared-only)	0.565	0.452	0.558	0.612	0.448	0.545	0.695	0.562	0.602	0.74	0.748	0.560	0.625	-10.44%
w/o Shared Expert	0.570	0.483	0.576	0.629	0.471	0.557	0.734	0.598	0.611	0.75	0.772	0.564	0.629	-7.75%
w/o Dual-stage Decision	0.586	0.472	0.576	0.630	0.466	0.561	0.715	0.583	0.615	0.74	0.766	0.606	0.659	-7.44%
w/o task encoder (GAT)	0.553	0.445	0.537	0.597	0.432	0.524	0.673	0.546	0.580	0.69	0.699	0.553	0.603	-13.66%
w/o LLM Guidance in Reward	0.567	0.440	0.559	0.617	0.447	0.547	0.706	0.569	0.619	0.75	0.728	0.576	0.630	-9.96%
w/o Format Reward (R_{format})	0.597	0.486	0.600	0.651	0.477	0.586	0.733	0.589	0.632	0.77	0.769	0.613	0.669	-5.11%
w/o RL LLM Training (Base LLM)	0.469	0.356	0.447	0.512	0.365	0.456	0.598	0.469	0.530	0.61	0.653	0.515	0.560	-24.09%
w/o Agent (AGRL-S)	0.486	0.335	0.361	0.525	0.419	0.422	0.474	0.375	0.396	0.58	0.601	0.464	0.512	-30.89%

Exp-3: Online Debugging Time. Figures 7 and 8 detail the online retrieval, generation, and total debugging times on the MySQL SO and PG SO datasets. AGRL demonstrates remarkable retrieval efficiency, significantly outpacing complex GraphRAG baselines like ToG-2 and KET-RAG, which suffer from extensive traversal or intricate indexing latency. Its retrieval speed even rivals Andromeda, whose vector search is offset by slow temporal analysis overhead. Overall, AGRL maintains a competitive total latency of approximately 10 seconds. Despite utilizing multiple reasoning iterations, each individual round executes swiftly, ensuring that the aggregate temporal overhead remains minimal.

6.3 Parameter Analysis

Exp-4: Effect of Reasoning Rounds (r_{max}). Figure 9 reports the impact of varying the round limit r_{max} of the debugging agent from 1 to 9. Accuracy improves significantly from 1 to 5 rounds, confirming C3’s need for iterative reasoning in complex configuration issues. However, performance plateaus beyond 5 rounds. Five iterations provide sufficient diagnostic context, whereas excessive rounds merely incur unjustified temporal overhead without yielding further accuracy gains.

Exp-5: Effect of Number of Experts. Figure 10 shows performance variations when scaling the MoE network from 1 to 4 experts. These experts are sequentially activated: (1) Shared Expert (Generalist), (2) Performance Diagnosis, (3) Error Fix, and (4) Parameter Setting. Accuracy consistently rises with each addition, validating our Shared-Specific MoE architecture. Integrating these diverse modules effectively captures heterogeneous graph features, substantially enhancing the retriever’s task-adaptive perception.

Exp-6: Effect of Q&A Connectivity Degree (k_{QA}). Figure 11 evaluates the influence of varying k_{QA} . Performance steadily improves and peaks near $k_{QA} = 5$. This optimal connectivity creates vital semantic shortcuts between Q&A vertices and manual entities, smoothly uncovering implicit dependencies like distant error-knob relations. Conversely, excessive connections (e.g., $k_{QA} \geq 7$) degrade accuracy. Over-linking morphs Q&A vertices into noisy super-hubs, diluting path specificity and misdirecting the agent toward irrelevant graph regions.

Exp-7: Effect of Subgraph Size (n_{max}). To isolate retrieval robustness, we evaluate AGRL-S (omitting multi-round reasoning) across varying subgraph sizes n_{max} . The complete MySQL and PostgreSQL graphs contain 7,309 and 2,141 vertices, respectively. Figure 12 demonstrates accuracy peaking near $n_{max} \in [15, 25]$. Undersized subgraphs ($n_{max} = 5$) lack sufficient context, whereas larger scopes ($n_{max} \geq 35$) trigger mild lost-in-the-middle degradation. This minimal drop highlights the retriever’s robust noise suppression, efficiently prioritizing relevant entities even within expanded searches.

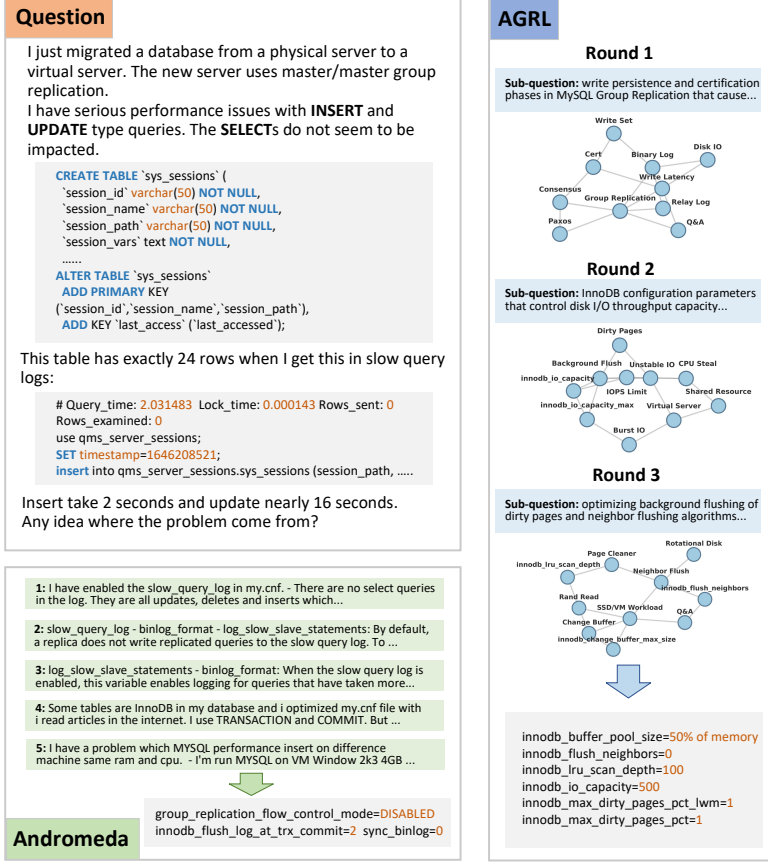


Fig. 15. Case study

Exp-8: Effect of Missing Q&A Ratio. To simulate data scarcity for new systems, we progressively remove historical Q&A vertices. As shown in Figure 13, the performance declines smoothly. Even at a 100% missing rate (relying solely on manuals), AGRL still retains 80.2% of its original performance. The manual-centric graph backbone and the agent's multi-hop reasoning remain useful without Q&A vertices in this setting.

Exp-9: Effect of Noisy Q&A Ratio. To quantify robustness against imperfect forums, we inject synthetic "noisy Q&A vertices" containing misleading configurations but high keyword overlaps. Figure 14 shows that AGRL remains relatively robust to this structural noise. Even at a 100% noise ratio, it preserves 76.6% of its original accuracy. The RL-driven retriever can learn to assign lower selection probabilities to noisy vertices and favor more reliable paths.

6.4 Ablation & Case Study

Exp-10: Ablation Study. Replacing the Hybrid Graph with a Global Graph (−13.30%) supports **C1**, showing the effectiveness of AGRL's hybrid construction strategy over naive whole-corpus graph extraction. Removing version and timestamp metadata (−5.06%) further shows the benefit of retaining version and temporal cues. Ablating the specific and shared experts (−10.44% and −7.75%), the dual-stage decision mechanism (−7.44%), and the task-adaptive encoder (−13.66%) supports **C2**

by showing the need for intent-adaptive perception. Omitting the LLM-guided reward (−9.96%) or format reward (−5.11%) degrades the training optimization efficiency. The largest declines stem from abandoning GRPO fine-tuning (−24.09%) or removing the multi-round framework (w/o Agent, −30.89%), confirming the importance of specialized policy optimization and the iterative exploration required by C3.

Exp-11: Case Study. A representative case is shown in Figure 15. In this case, a user reports severe write latency (UPDATE taking 16s) on a tiny table after migrating a Master/Master Group Replication setup to a virtual server. Andromeda is misled by surface-level keywords and incorrectly attributes the issue to protocol-level flow control. It then recommends dangerous fixes, such as disabling `group_replication_flow_control_mode` or relaxing ACID durability by setting `sync_binlog=0`. These suggestions only mask the symptoms while compromising data consistency.

In contrast, AGRL utilizes its hybrid knowledge graph to conduct structured causal analysis. The agent executes a multi-hop traversal, progressing methodically from abstract mechanisms to concrete configurations. Specifically, it anchors on the *Group Replication* vertex to identify the synchronous persistence bottleneck, and then navigates through *Environmental Context* nodes to diagnose virtual server I/O limitations. Ultimately, it converges on highly relevant *Optimization Pattern* vertices, such as SSD neighbor flushing. Guided by this trajectory, AGRL recommends exact, context-aware settings (e.g., `innodb_io_capacity=500` and `innodb_flush_neighbors=0`), effectively resolving I/O saturation while preserving replication stability.

7 Conclusion

In this paper, we presented AGRL, an agentic GraphRAG framework for database configuration debugging. Through a synergistic component co-design, AGRL integrates hybrid graph construction, adaptive retrieval, and multi-round reasoning into a closed-loop diagnostic workflow. Its progressive RL training strategy further resolves the cold-start interdependency between the retriever and the debugging agent through decoupled initialization and iterative joint refinement. Extensive experiments demonstrate that AGRL significantly outperforms existing RAG and GraphRAG baselines, highlighting the promise of combining agentic reasoning with RL for intelligent database troubleshooting.

Acknowledgments

This work was supported in part by the NSFC under Grants No. (U23A20296, 62025206, and 62302444), and Zhejiang Province's "Lingyan" R&D Project under Grant No. 2024C01259. Yunjun Gao is the corresponding author of the work.

References

- [1] Uri Alon and Eran Yahav. 2020. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205* (2020).
- [2] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 3438–3445.
- [3] Sibe Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, et al. 2025. Automatic database configuration debugging using retrieval-augmented language models. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.
- [4] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. 2005. Automatic Performance Diagnosis and Tuning in Oracle.. In *Cidr*. 84–94.
- [5] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.
- [6] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitan-sky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused

- summarization. *arXiv preprint arXiv:2404.16130* (2024).
- [7] Ayat Fekry, Lucian Carata, Thomas Pasquier, Andrew Rice, and Andy Hopper. 2020. To tune or not to tune? in search of optimal configurations for data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2494–2504.
 - [8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
 - [9] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779* (2024).
 - [10] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems* 37 (2024), 132876–132907.
 - [11] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1003–1012.
 - [12] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems* 37 (2024), 59532–59569.
 - [13] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2025. GPTuner: An LLM-Based Database Tuning System. *ACM SIGMOD Record* 54, 1 (2025), 101–110.
 - [14] Dawei Li, Shu Yang, Zhen Tan, Jae Baik, Sukwon Yun, Joseph Lee, Aaron Chacko, Bojian Hou, Duy Duong-Tran, Ying Ding, et al. 2024. DALK: Dynamic Co-Augmentation of LLMs and KG to answer Alzheimer’s Disease Questions with Scientific Literature. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2187–2205.
 - [15] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.
 - [16] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranajpe, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.
 - [17] Chengyang Luo, Qing Liu, Yujie Guo, Kai Wang, Wenjie Zhang, and Yunjun Gao. 2026. AGRL: An Agentic GraphRAG Framework with Reinforcement Learning for Database Configuration Debugging (Technical Report). <https://github.com/Lcy77/AGRL-Technical-Report/blob/main/AGRL-Technical-Report.pdf>
 - [18] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.
 - [19] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2024. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. *arXiv preprint arXiv:2407.10805* (2024).
 - [20] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.
 - [21] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).
 - [22] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.
 - [23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
 - [24] Zhihong Shao, Peiyi Wang, Qihao Z hu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).
 - [25] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gangadharaiyah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents.(2024). (2024).
 - [26] Jiashuo Sun, Chengjin Xu, Luminyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697* (2023).
 - [27] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

- [28] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345. doi:10.1007/s11704-024-40231-1
- [29] Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 19206–19214.
- [30] Shirley Wu, Kaidi Cao, Bruno Ribeiro, James Zou, and Jure Leskovec. 2024. GraphMETRO: Mitigating Complex Graph Distribution Shifts via Mixture of Aligned Experts. *Advances in Neural Information Processing Systems* 37 (2024), 9358–9387.
- [31] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 international conference on management of data*. 1599–1614.
- [32] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 international conference on management of data*. 415–432.
- [33] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuwei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 international conference on management of data*. 2102–2114.
- [34] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176* (2025).
- [35] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. *arXiv preprint arXiv:2503.04338* (2025).

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026