SuperDRI: Graph-Guided Autonomous Troubleshooting for Cloud Databases

Yiwen Zhu
Microsoft, USA
yiwzh@microsoft.com

Joyce Cahoon
Microsoft, USA
jcahoon@microsoft.com

Qiushi Bai
Microsoft, USA
qiushibai@microsoft.com

Anna Pavlenko
Microsoft, USA
anna.pavlenko@microsoft.com

Divya Vermareddy
Microsoft, USA
dvermareddy@microsoft.com

Meina Wang
Microsoft, USA
meiwa@microsoft.com

Mathieu Demarne
Microsoft, USA
mdemarne@microsoft.com

Wenjing Wang
Microsoft, USA
wenjwang@microsoft.com

Swati Bararia
Microsoft, USA
bararia swati@microsoft.com

Hannah Lerner
Microsoft, USA
hannahlerner@microsoft.com

Katherine Lin
Microsoft, USA
katlin@microsoft.com

Subru Krishnan
Microsoft, Spain
subru@microsoft.com

Miso Cilimdzic
Microsoft, USA
misoc@microsoft.com

ABSTRACT

We demonstrate SuperDRI, an autonomous troubleshooting system for large-scale cloud databases such as Microsoft Fabric Data Warehouse. Incident diagnosis requires expert, multi-step reasoning across heterogeneous telemetry and tools, while much of this knowledge remains scattered across documentation and senior engineers' experience. Rather than relying on manually authored playbooks, SuperDRI learns structured troubleshooting workflows directly from historical incident traces through a multi-agent extraction pipeline, incrementally building InvestiGraph, a typed workflow graph of database problems, diagnostic actions, and causal relationships for each domain. At runtime, an agentic graph-traversal mechanism explores InvestiGraph, generates and executes read-only Kusto (KQL) queries against database telemetry, and iteratively reasons over results to reach mitigations. The system further incorporates real-time feedback learning, allowing database engineers to inject domain knowledge into the graph.

We illustrate four capabilities: (1) exploring a production-scale InvestiGraph learned from over 1,300 real incidents (10,000+ nodes), (2) observing autonomous troubleshooting with adaptive KQL generation and execution against Fabric DW telemetry, (3) editing the graph and observing how feedback immediately improves subsequent runs, and (4) accessing SuperDRI through multiple MCP-compatible clients. SuperDRI is deployed in production at Microsoft for Azure Data incident management.

PVLDB Reference Format:
Yiwen Zhu, Joyce Cahoon, Qiushi Bai, Anna Pavlenko, Divya Vermareddy, Meina Wang, Mathieu Demarne, Wenjing Wang, Swati Bararia, Hannah Lerner, Katherine Lin, Subru Krishnan, and Miso Cilimdzic. SuperDRI: Graph-Guided Autonomous Troubleshooting for Cloud Databases. PVLDB, 19(12): 4678 - 4681, 2026.
doi:10.14778/3827998.3828095

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828095

1 INTRODUCTION

Managing large-scale cloud database services such as Microsoft Fabric Data Warehouse (Fabric DW) [6], Azure Synapse Analytics, and Azure SQL Database requires continuous operational vigilance. When incidents occur (query failures, connectivity degradation, workspace activation errors), on-call engineers (Designated Responsible Individuals, or DRIs) must rapidly diagnose root causes by reasoning over heterogeneous telemetry, executing diagnostic queries against multiple Kusto clusters, and coordinating across teams. This troubleshooting process demands deep expertise and involves multi-step reasoning that is difficult to automate [3, 5].

Current practice relies on manually authored troubleshooting guides (TSGs) and playbooks; however, these are labor-intensive to maintain, quickly become outdated as database systems evolve, and fail to address the long tail of rare failure modes. Recent advances in LLM-based agents [7, 8] have renewed interest in automating these workflows, yet purely prompt-driven approaches lack structural grounding and are brittle when faced with the domain-specific constraints of database operations (e.g., correct KQL syntax, cluster-specific schemas, time-sensitive telemetry windows).

We present SuperDRI, an autonomous troubleshooting system built on a *trace-first* paradigm. Rather than relying on static documentation, it learns executable workflows from historical incident traces: (1) a multi-agent pipeline incrementally constructs InvestiGraph, a typed workflow graph of database problems, diagnostic actions, and causal relationships; (2) an LLM-driven agent traverses the graph, generates and executes KQL queries against telemetry, and reasons over results; and (3) real-time engineer feedback directly enriches the graph.

SuperDRI is deployed in production at Microsoft for Azure Data incident management for database product teams including Synapse

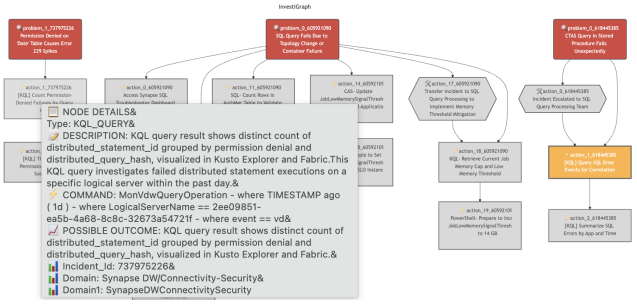


Figure 1: Subgraph of the InvestiGraph.

SQL DW and Azure SQL Database. It has processed over 1,300 incident records spanning one year to build an InvestiGraph containing over 10,000 nodes. In end-to-end evaluation on 61 held-out incidents, the agent reaches mitigation in >90% of cases, with an average of 2.39 matched expert actions per incident.

2 SYSTEM OVERVIEW

SuperDRI is an autonomous troubleshooting system designed for cloud database operations. It is organized into two complementary phases: an *offline construction* phase that distills database incident traces into InvestiGraph, a structured workflow graph, and an *online execution* phase that leverages the graph to guide autonomous, LLM-driven incident diagnosis.

2.1 Offline: Adaptive Graph Construction

The offline pipeline transforms raw database incident traces, specifically IcM (Incident Management) [3] tickets containing engineer notes, diagnostic KQL queries, telemetry outputs, and resolution summaries, into InvestiGraph, a typed workflow graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (see Figure 1). The graph schema defines three core entity types: **Domain** ($\mathcal{D}$, e.g., Connectivity-Security, QueryOptimization, TransactionalStorage), **Problem** ($\mathcal{P}$, e.g., ActivateWorkspace failures, query timeouts), and **Action** ($\mathcal{A}$, e.g., KQL diagnostic queries, CAS commands, service restarts), connected by typed edges: `CAUSES` (problem→problem), `RESOLVES` (action→problem), `LEADS_TO` (action→action), and `BELONGS_TO` (entity→domain).

We construct a multi-agent extraction pipeline:

(1) **Data ingestion and preprocessing**: A document preprocessing agent normalizes heterogeneous incident data (HTML TSGs, Markdown wikis, JSON telemetry, IcM conversation threads).
(2) **Taxonomy extraction**: Specialized agents extract domain, problem definitions, resolution paths, and discrete diagnostic actions (particularly KQL queries that comprise 80–90% of actions).
(3) **Clustering and graph update**: Extracted resolution paths are merged with the existing graph through agglomerative clustering over canonicalized embeddings for entity resolution. The pipeline supports *incremental construction*, integrating new incidents without full graph reconstruction.

InvestiGraph is deployed on Azure Cosmos DB (Gremlin API) for low-latency graph traversal, with vector embeddings indexed in Azure AI Search for semantic and keyword-based retrieval.

2.2 Online: Agentic Graph Traversal

The online engine performs iterative graph traversal using a nested control-flow architecture built on the ENCO orchestration framework [10]. The orchestration layer is modular and can be replaced by any MCP-compatible client [1] (e.g., GitHub Copilot CLI [4], Claude Code [2]).

A **Coordination Agent** operates the main reasoning loop: at each step, it formulates a semantic query based on the investigation context, retrieves a relevant subgraph from InvestiGraph via vector and keyword-based similarity search, and invokes an **Action Planner** to select the most promising diagnostic actions. If no suitable actions are found, the planner signals a pivot, causing the retriever to explore a different region of the graph, mimicking how a human DRI would shift investigation strategies.

Selected actions are dispatched to specialized **Execution Agents**. The **Kusto Agent** is the primary execution agent for read-only KQL queries, equipped with five composable skills:

(1) *KQL Generation* synthesizes context-specific queries parameterized with incident details (region, time window, database identifiers) and informed by graph-persisted KQL hints from prior feedback.
(2) *Cluster Routing* selects the appropriate Fabric DW telemetry cluster based on incident metadata and action node annotations.
(3) *Schema Adaptation* adjusts query syntax for cluster-specific schemas, quoting conventions, and escape characters.
(4) *Self-Correction* detects execution failures (syntax errors, permission issues, empty results, wrong cluster, out-of-memory issues) and automatically retries with corrective modifications [9].
(5) *Result Summarization* distills raw telemetry outputs into actionable findings for the Coordination Agent.

This agent returns inferred data findings, rather than raw query outputs, to the Coordination Agent for iterative refinement until a mitigation is reached. All tool invocations are performed under the requesting user's existing permissions using role-based access control (RBAC). Consequently, the system cannot access data, clusters, or operational resources beyond those already available to the user.

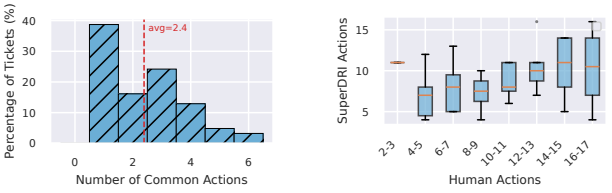
KQL Generation Accuracy. Generating executable KQL from incident context based on SuperDRI is simpler than vanilla NL-to-SQL tasks as it is presented with several historic queries, but it still faces additional challenges: heterogeneous cluster schemas, quoting conventions, and time-varying table availability. In an evaluation of 100 incidents, the Kusto Agent selected and executed 957 queries against live telemetry clusters. Of the executed queries, 735 succeeded on the first attempt (76.8%). Table 1 breaks down the 222 failures by type. The Kusto Agent's self-correction skills substantially recover from these failures: the *Schema Adaptation* skill recovers 49.4% (38/77) of schema-related failures, while the *Empty Result Retry* skill recovers 44.9% (48/107) by adjusting time ranges, filters, or target tables, yielding 86 additional successful queries beyond the baseline.

End-to-End Online Evaluation. We evaluate SuperDRI on 61 held-out Fabric DW incidents by comparing agent-generated actions against human DRI actions. Figure 2(a) shows the distribution of matched expert actions per incident, with an average of 2.39

Table 1: KQL execution failure type distribution.

Failure Type	Count	%	Failure Type	Count	%
Zero rows returned	138	62.2	Column not found	5	2.3
Table/entity not found	36	16.2	Type mismatch	3	1.4
Semantic error	20	9.0	Authentication error	2	0.9
Syntax error	11	5.0	Exceeded limits	1	0.5
Timeout	6	2.7			

matches. Figure 2(b) shows that SuperDRI maintains a consistent action count (median 7–10) regardless of how many actions human DRIs took, suggesting the agent efficiently converges on diagnostic paths without unnecessary steps.



(a) Matched actions distribution. **(b) Agent vs. human DRI actions.**

Figure 2: Online evaluation: (a) distribution of matched expert actions per incident (avg. 2.39), (b) SuperDRI actions vs. human DRI action count.

2.3 Feedback Learning

SuperDRI incorporates three feedback mechanisms that enable continuous learning from database engineer interactions:

Action Hint Collector. The system passively monitors DRI interactions with the action interface, tracking which suggested actions are accepted, skipped, reordered, or modified, and records these behavioral signals as hints on the corresponding InvestiGraph nodes. Subsequent traversals incorporate these hints when ranking candidate actions in the same graph region, progressively aligning suggestions with engineer preferences.

KQL Generation Feedback. Database engineers can edit generated KQL queries before execution or provide free-text hints (e.g., "use the `SessionActivity` table instead of `RequestLog` for this cluster"). The system infers the rationale behind modifications and persists them as feedback on the corresponding action nodes, substantially improving future query generation accuracy for database-specific patterns such as quoting conventions, escape characters, and cluster-specific schemas.

Graph Editor. A dedicated agent connects to the InvestiGraph database and exposes an intuitive form-based interface for database engineers to directly add, modify, or remove nodes and edges, enabling rapid graph maintenance as database systems evolve (e.g., new Fabric DW features, schema changes, or newly discovered failure patterns).

Note that all feedback mechanisms directly inject domain knowledge back to the graph which will be retrieved dynamically at runtime, rather than creating a standalone memory management system or storage.

3 DEMONSTRATION SCENARIOS

We demonstrate SuperDRI through four scenarios to showcase its core capabilities for database operations troubleshooting. The video recording of the demo is available at https://youtu.be/HGnCzaJs5ok.

3.1 Scenario 1: Exploring the Database Workflow Graph

In the first scenario, we present InvestiGraph to demonstrate the learned troubleshooting knowledge from database incident traces. The graph captures domain-specific workflows spanning connectivity diagnostics, authentication failures, query performance degradation, workspace management, and resource provisioning issues. At runtime, subgraphs related to one specific domain will be retrieved.

Interaction. Users can issue natural-language queries (e.g., "*Show me troubleshooting workflows for ActivateWorkspace failures in Fabric DW*") and the system retrieves relevant subgraphs, rendered as interactive Mermaid diagrams (see Figure 1). The investigation is visualized at each step. Users can:

- Navigate from database problem nodes (e.g., "workspace activation failing at high rate") to associated diagnostic action chains, following `RESOLVES` and `LEADS_TO` edges through KQL queries, CAS commands, and mitigation steps.
- Inspect individual node properties, including KQL query templates, target Kusto clusters, preconditions, expected outcomes, and provenance links to the original IcM tickets.
- Click on action nodes to trigger the corresponding diagnostic steps in the next scenario, observing how graph traversal guides the troubleshooting process.

3.2 Scenario 2: Autonomous Database Troubleshooting

In the second scenario, we illustrate SuperDRI autonomously diagnosing a real database incident end-to-end. Given only an incident description, the agent traverses InvestiGraph, selects diagnostic actions, generates and executes KQL queries against telemetry clusters, and iteratively reasons over results to reach a mitigation.

Interaction. Users can provide an incident ID, and the system begins its autonomous investigation. The demonstration highlights:

- **Incident understanding**: The Coordination Agent extracts and parses the incident description and metadata to contextualize the investigation (e.g., identifying the database product, region, failure symptoms).
- **Graph-guided action selection**: The Coordination Agent retrieves a relevant subgraph from InvestiGraph and the Action Planner selects the most promising diagnostic action, displaying its reasoning in the chat interface.
- **Adaptive KQL generation**: When a diagnostic query is needed, the Kusto Agent generates a context-specific KQL query parameterized with incident details (database ID, region, time window). The query is presented via an *adaptive card* (Figure 3) that allows cluster selection, query inspection, editing, and one-click execution against the appropriate Fabric DW telemetry cluster.
- **Iterative reasoning**: After each query execution, the agent incorporates the telemetry results into its investigation context

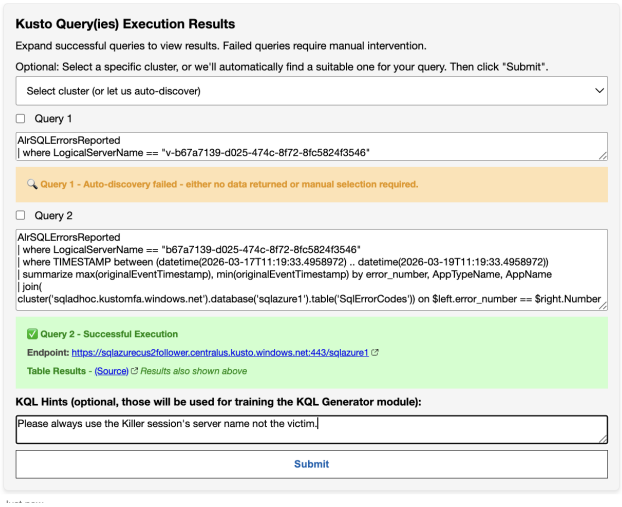


Figure 3: UI for KQL query execution and feedback collection.

and determines the next diagnostic step, potentially pivoting to a different subgraph if the initial approach is unproductive.

- **Error recovery**: If a KQL query fails (syntax error, permission issue, empty result, wrong cluster), specialized skills within the Kusto Agent automatically diagnose the failure and generate corrective actions, such as modifying query syntax for cluster-specific schemas or requesting elevated permissions.

3.3 Scenario 3: Feedback Learning

We illustrate the feedback loops that enable SuperDRI to continuously improve through agent-engineer interaction. This scenario demonstrates both implicit feedback (captured during troubleshooting) and explicit graph editing.

Interaction. Building on Scenario 2, users interact with the feedback mechanisms:

- **Action redirection**: During an autonomous troubleshooting session, users redirect the agent's chosen action (e.g., "check the `SessionActivity` table instead of `RequestLog`"). The Action Hint Collector captures the reasoning and attaches it to the relevant InvestiGraph nodes.
- **Query feedback**: Users edit a generated KQL query in the adaptive card before execution (see Figure 3). The system detects the diff, infers the rationale (e.g., "use single quotes for string literals in this cluster's KQL dialect"), and persists the hint for future query generation.
- **Direct graph editing**: Users open the Graph Editor interface to add new nodes (e.g., a recently discovered KQL query for a new Fabric DW feature), create edges to existing nodes, or annotate nodes with troubleshooting hints.

Key insight. This scenario shows how SuperDRI bridges automated learning and DRI expertise. As Fabric DW evolves (e.g., schema changes or new failure patterns), engineers can incrementally refine the InvestiGraph without re-running the offline pipeline.

3.4 Scenario 4: MCP Integration

SuperDRI is also implemented as a set of MCP-compatible tools [1]. In this scenario, users experience similar troubleshooting capabilities accessed through different MCP clients: GitHub Copilot CLI [4], Claude Code [2], and GitHub Copilot in Visual Studio Code. Each client connects to the shared InvestiGraph and Kusto Agent backend using On-Behalf-Of (OBO) authentication. The graph traversal, KQL generation, and feedback mechanisms are client-agnostic. This architecture enables database engineers to invoke SuperDRI from their preferred development environment (terminal, IDE, or web chat) while maintaining a single, consistent workflow graph and feedback loop across all surfaces.

Additionally, SuperDRI provides an API endpoint for integrations with other internal tools, allowing users to trigger troubleshooting sessions directly to further streamline the workflow.

4 CONCLUSIONS

We demonstrated SuperDRI, an autonomous troubleshooting system that learns executable workflow graphs from historical cloud database incidents. Across four scenarios, attendees explore InvestiGraph, observe adaptive KQL-based diagnosis, refine the graph through real-time feedback, and access these capabilities from multiple MCP-compatible clients. SuperDRI is deployed at Microsoft for Azure Data incident management.

REFERENCES

[1] Anthropic. 2024. Model Context Protocol (MCP). https://modelcontextprotocol.io/. Open standard for connecting AI applications to external systems; enables structured, secure interaction between LLMs and data sources/tools via a universal protocol.

[2] Anthropic. 2025. Claude Code. https://docs.anthropic.com/en/docs/claude-code. Agentic coding tool with built-in MCP client support.

[3] Zhe Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Haidong Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Yingnong Dang, Feng Gao, Pu Zhao, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Michael R. Lyu. 2020. Towards Intelligent Incident Management: Why We Need It and How We Make It. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. https://doi.org/10.1145/3368089.3417055

[4] GitHub. 2026. GitHub Copilot CLI. https://github.blog/changelog/2026-02-25-github-copilot-cli-is-now-generally-available/. MCP-compatible agentic coding assistant for the command line.

[5] Jun-Jie Huang et al. 2024. A Survey of AIOps for Failure Management in the Era of Large Language Models. *arXiv preprint arXiv:2406.11213* (2024).

[6] Microsoft. 2024. Microsoft Fabric Data Warehouse. https://learn.microsoft.com/en-us/fabric/data-warehouse/ Accessed: 2025-03-27.

[7] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[8] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[9] William Zhang, Yiwen Zhu, Yunlei Lu, Mathieu Demarne, Wenjing Wang, Kai Deng, Nutan Sahoo, Katherine Lin, Miso Cilimdzic, and Subru Krishnan. 2025. FLAIR: Feedback Learning for Adaptive Information Retrieval. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25)*. Association for Computing Machinery, Seoul, Republic of Korea. https://doi.org/10.1145/3746252.3761553 Work done while interning at Microsoft.

[10] Yiwen Zhu, Mathieu Demarne, Kai Deng, Wenjing Wang, Nutan Sahoo, Hannah Lerner, Anjali Bhavan, Divya Vermareddy, Yunlei Lu, Swati Bararia, William Zhang, Xia Li, Katherine Lin, Miso Cilimdzic, and Subru Krishnan. 2026. ENCO: Deploying Production-Scale Engineering Copilots. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1* (Republic of Korea) *(KDD '26)*. Association for Computing Machinery, New York, NY, USA, 2606–2616. https://doi.org/10.1145/3770854.3783949