Diagnostic Knowledge Graphs: Automated Benchmark Construction and Deterministic Evaluation for Multi-Step Reasoning Agents

Chuci Chen
chuciche@amazon.com
Amazon
Bellevue, WA, USA

Zhenyu Zhang
zhenyuzh@amazon.com
Amazon
Bellevue, WA, USA

Xinyang Shen
xinyans@amazon.com
Amazon
Bellevue, WA, USA

ABSTRACT

Evaluating multi-step diagnostic reasoning in LLM agents remains an open problem. When cause labels are extracted from resolved operational cases (customer-service tickets, incident reports, clinical notes), the resulting gold standards exhibit extreme vocabulary explosion—5,076 unique cause strings from 2,196 tickets on a single symptom, 92% appearing only once—making LLM-as-judge protocols variance-prone (±2–3pp inter-run) and longitudinal monitoring impossible. We argue that building reproducible diagnostic benchmarks and building effective diagnostic agents are *dual problems solved by the same artifact*—a canonical knowledge structure that normalizes evaluation gold-standards and constrains agent hypothesis spaces simultaneously.

We instantiate this duality as *Diagnostic Knowledge Graphs* (DKGs): hierarchical cause trees with frequency priors, built automatically from resolved tickets via LLM extraction, two-pass BERTopic clustering, and LLM merge, with optional per-node SQL grounding against operational data. The pipeline compresses 5,076 cause strings to 73 canonical clusters (97% coverage) and scales to 284 symptom families from 14,953 tickets without manual curation. A domain-expert validation shows the resulting normalization achieves 92% accuracy—exceeding the LLM judge's 80% agreement with the same expert—while enabling deterministic scoring (σ=0 inter-run variance).

Under a fair semantic-judge protocol (scoring against raw gold chains, so DKG agents gain no vocabulary advantage), the DKG yields a +32pp action-accuracy lift over an unstructured baseline (73% vs. 41%). A five-condition ablation reveals that the cause menu and frequency priors alone—without SQL—account for the dominant share of this gain (78–86% of the total lift), establishing that the evaluation infrastructure itself is the primary source of agent improvement. Pre-validated SQL adds a directionally positive but non-significant benefit at n=100, bounded by data sparsity rather than a method ceiling.

KDD'26 Workshop, August 09, 2026, Jeju, Republic of Korea

ACM ISBN 978-1-4503-XXXX-X/2026/08…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Information systems** → **Clustering**; *Graph-based database models*; • **Computing methodologies** → *Natural language processing*.

KEYWORDS

large language models, LLM agent evaluation, LLM-as-a-judge, root cause analysis, diagnostic knowledge graphs, benchmark construction, text clustering

ACM Reference Format:
Chuci Chen, Zhenyu Zhang, and Xinyang Shen. 2026. Diagnostic Knowledge Graphs: Automated Benchmark Construction and Deterministic Evaluation for Multi-Step Reasoning Agents. In *Proceedings of KDD Workshop on Evaluation and Trustworthiness of Agentic AI (KDD'26 Workshop).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Evaluating multi-step diagnostic reasoning in LLM agents is an open problem with immediate practical consequences. As agentic AI systems are deployed for customer-service triage, IT incident resolution, and medical diagnosis, we need evaluation that is (1) *per-step*: identifying where in the reasoning chain an agent fails, not just whether the final answer is correct; (2) *deterministic*: producing identical scores across repeated evaluations of the same output, enabling longitudinal drift detection; and (3) *faithful*: correlating highly with expert judgment. No existing evaluation approach delivers all three properties simultaneously.

In our domain (logistics customer service), free-text cause labels extracted from resolved tickets exhibit extreme vocabulary explosion: 5,076 unique cause strings from 2,196 tickets about a single symptom, with 92% appearing exactly once. This long-tail distribution is the root cause of evaluation failure: it forces reliance on fuzzy LLM-as-judge scoring, which introduces ±2–3pp inter-run variance and achieves only moderate agreement with domain experts (Cohen's $\kappa = 0.60$).

We argue that this evaluation challenge and the challenge of building effective diagnostic agents are *dual problems solved by the same artifact*: a canonical knowledge structure that simultaneously normalizes evaluation gold-standards and constrains agent hypothesis spaces. We instantiate this as *Diagnostic Knowledge Graphs* (DKGs)—hierarchical cause trees with frequency priors at each node, built automatically from resolved cases. The framework applies wherever resolved cases plus operational data exist: IT incident management, medical triage, financial fraud investigation, manufacturing defect analysis.

The evaluation problem. Three properties make diagnostic evaluation harder than standard NLG evaluation:

- **Multi-step structure**: a correct diagnosis is not a single label but a causal chain (symptom → cause → deeper cause → root cause), requiring per-step comparison to localize errors.
- **Vocabulary explosion**: without normalization, semantically identical diagnoses receive different surface labels (e.g., 5,076 unique strings for 73 underlying mechanisms), and an LLM judge must resolve this on every comparison—introducing variance that masks real performance changes.
- **Absence of deterministic scoring**: unlike code generation (where tests pass or fail), diagnostic output has no natural exact-match protocol unless both sides share a canonical vocabulary.

The DKG solution. A DKG addresses all three by providing: (1) a tree structure that defines evaluation *levels* (each depth is a scoring point), (2) canonical cluster labels that compress thousands of surface forms to a closed vocabulary (5,076 → 73 clusters, 97% coverage), enabling exact-match scoring with zero inter-run variance, and (3) optional per-node SQL queries validated against operational data, providing verifiable evidence at deployment time. The same artifact that normalizes the benchmark also constrains the agent's hypothesis space to historically observed causes—yielding a +32pp action-accuracy lift over unstructured baselines as a direct consequence of the evaluation infrastructure.

Two evaluation modes. We deliberately separate two uses of the DKG for evaluation: (1) *Semantic-judge mode* for fair cross-method comparison—all conditions (including DKG-constrained agents) are scored against *raw* extracted gold chains by an LLM judge, so the DKG agent gains no vocabulary-alignment advantage. (2) *Deterministic exact-match mode* for production monitoring—both gold chains and agent output reference canonical DKG node IDs, reducing evaluation to string equality: judge-free, reproducible, and suitable for detecting performance drift after model updates or API changes.

Contributions.

(1) **An automated pipeline for constructing reproducible diagnostic benchmarks from noisy operational data** (primary contribution). Two-pass BERTopic clustering with LLM merge compresses thousands of surface-form cause strings to canonical clusters (5,076 → 73 with 97% coverage on the largest family), producing normalized golden chains for deterministic per-step scoring. The pipeline scales to *284 symptom families* from 14,953 tickets without manual curation. Domain-expert validation confirms 92% normalization accuracy—exceeding the LLM judge's 80% agreement—establishing the benchmark as more faithful than the evaluation method it replaces.

(2) **A dual-mode evaluation framework with formal error decomposition.** The DKG supports both semantic-judge scoring (for fair cross-method comparison) and deterministic exact-match scoring (for production monitoring and drift detection). We formalize coverage loss as the sum of three diagnosable error types (tree gap, query error, judge error), each targeted by a specific validation gate.

(3) **Empirical validation of the evaluation-agent duality.** A five-condition ablation on three SQL-grounded symptom families (265 evaluation tickets), plus DKG+Tree vs. LLM-only evaluation on three additional families (n=100 each), shows that the cause taxonomy and frequency priors alone—the same artifact that normalizes the benchmark—account for the dominant share of a +32–44pp action-accuracy gain over unstructured baselines (78–86% of total lift). This establishes that investment in evaluation infrastructure *directly* improves agents at inference time.

2 RELATED WORK

LLM-as-judge and evaluation variance. Using LLMs to evaluate free-text outputs has become the default for open-ended generation [18]. However, recent work reveals fundamental limitations: JudgeBiasBench [19] identifies 12 types of systematic biases across 4 dimensions, including position bias and verbosity bias. Beyond systematic biases, LLM judges also exhibit low intra-rater reliability, producing different scores for the same input across runs and making evaluation non-reproducible [**?**]. Our approach sidesteps these issues by constraining both agent output and gold standard to a shared vocabulary, reducing evaluation to exact string matching where applicable.

Benchmark construction from noisy data. Label noise in LLM-generated annotations can substantially bias downstream model evaluation [4], and multi-step reasoning chains compound this with structural noise (ordering, depth, completeness) on top of label noise. BERTopic [6] provides our core clustering mechanism. Reasoning-based refinement of unsupervised clusters [8] uses LLMs to validate clustering output; we adopt a similar idea in our *LLM merge* step (Section 4.3), with the stricter criterion that two clusters merge only if they describe the same cause from the customer's perspective and lead to the same resolution action—catching semantic duplicates that share no tokens but converge to the same mechanism.

Knowledge graphs for fault diagnosis. Knowledge graphs organize domain knowledge as structured entity-relationship networks [7, 9]. In fault diagnosis, KGroot [14] constructs failure event graphs from structured logs; Cloud Atlas [15] synthesizes causal graphs from documentation; KG4Diagnosis [22] combines LLMs with KGs for medical diagnosis; GROVE [1] organizes debugging knowledge into trees for hardware failures. KG-driven diagnosis is a growing paradigm in manufacturing [3, 20] and microservice root cause analysis [13]. However, none connect graph nodes to executable queries against operational databases—our key differentiator. MedRAG [17] integrates hierarchical graphs with health records; SOP-Bench [12] benchmarks agents on industrial procedures. Our DKG is built from *unstructured* ticket text, with each node grounded in a validated SQL query.

LLM-based KG construction. Recent surveys document the shift to LLM-driven KG construction [2, 21]. Our pipeline uses LLMs for extraction (Stage 1) and taxonomy construction (Stage 2), but adds both a normalization layer (Stage 2b) and a SQL grounding loop (Stage 3).

Text-to-SQL and RAG. Spider 2.0 [10] reports 20% accuracy on enterprise schemas, motivating our pre-validation approach. RAG

[11] and GraphRAG [5] retrieve documents to ground responses; KG-enhanced RAG has been applied to customer service QA [16]. Our DKG replaces retrieval-by-analogy with structured tree traversal and SQL verification.

Positioning. The closest work to ours combines knowledge graphs with diagnostic agents (KGroot, GROVE, KG4Diagnosis). We differ in three ways: (1) our DKG is constructed *bottom-up* from noisy ticket text rather than curated ontologies; (2) we explicitly address the *evaluation* problem (vocabulary explosion, inter-run variance) rather than focusing solely on inference; and (3) we demonstrate the evaluation-agent *duality*—that the same artifact solving the benchmark problem also solves the agent problem—a connection absent from prior work.

3 PROBLEM FORMULATION

We formalize the DKG structure, SQL grounding objective, and evaluation metrics. The central design principle is *precision under noise*: both DKG construction (from noisy ticket text, 8% extraction failure) and inference (70–87% per-node SQL accuracy) minimize the risk of confidently wrong diagnoses, at the cost of occasionally over-reporting candidates.

3.1 Diagnostic Knowledge Graph

A DKG is a rooted tree $\mathcal{G} = (V, E, q, a)$ where V is a set of nodes (root = symptom, internal = causes at increasing depth, leaves = root causes); $E \subseteq V \times V$ are directed parent-child edges representing causal decomposition; $q : V \to \mathcal{Q} \cup \{\perp\}$ maps each node to a parameterized SQL query that returns TRUE when the condition holds, or $\perp$ if non-verifiable; and $a : V \to \mathcal{A}$ maps leaf nodes to resolution actions.

Precision-oriented design. Each node is a one-sided detector (TRUE = active; FALSE discarded, not used for routing) and all siblings are evaluated independently. This avoids unrecoverable false-negative routing errors at the cost of broader candidate sets (Section 3.4). Figure 1 shows the full DKG for our running example family ("shipment stuck in receiving"); Table 4 in Appendix A lists each node's verification status and recommended action.

3.2 SQL Grounding Objective

Each query $q(v)$ is parameterized by (customer ID, case ID, reference date). A node is *verified* if $\text{Acc}(v) = \frac{1}{M}\sum_{i=1}^{M} \mathbf{1}[q(v)(x_i) = y_{i,v}] \geq 0.7$ on M=30 ground-truth tickets. The 0.7 threshold accounts for extraction noise ($\delta \approx 0.08$), yielding true accuracy $\geq$0.62. Unverified nodes are retained but flagged as low-confidence.

3.3 Evaluation Metrics

We define two primary metrics. For each level i in the gold path, a per-step LLM judge determines whether the agent's corresponding step describes the same diagnostic mechanism. Full metric suite in Appendix J.

Diagnostic accuracy (M1). Fraction of the gold path's reasoning levels matched anywhere in the agent's derived path:

$$\text{DiagAcc}(\mathbf{g}, \mathbf{d}) = \frac{|\{g_i \in \mathbf{g} : \exists\, d_j \in \mathbf{d},\ \text{sem}(g_i, d_j) = 1\}|}{|\mathbf{g}|} \quad (1)$$

In comparative mode (Section 5), sem is an LLM judge; in production-monitoring mode, it reduces to exact-match over canonical cluster IDs.

Action accuracy (M2). Did the agent recommend the correct resolution action? The metric most directly tied to customer impact.

3.4 Error Analysis Framework

Coverage loss decomposes into three terms: $1 - \text{DiagAcc}(x) \geq \epsilon_{\text{struct}} + \epsilon_{\text{sql}} + \epsilon_{\text{eval}}$, where ϵ_{struct} = tree gap (gold node absent), ϵ_{sql} = query error (node present but SQL wrong), ϵ_{eval} = judge error. The DKG's one-sided design (FALSE discarded, siblings evaluated independently) accepts recoverable false positives to eliminate unrecoverable false negatives. On the running example family, 12/16 failures are SQL misdirection (recoverable) and only 2 are tree gaps.

4 METHOD

Our method proceeds in five stages (Figure 2): extract diagnostic chains, build a cause taxonomy, ground each node with validated SQL, evaluate agents, and augment with new causes. Each stage incorporates a precision-oriented filter: frequency filtering (Stage 2), SQL verification thresholds (Stage 3), and one-sided traversal (Stage 4). The approach generalizes to any domain with resolved cases containing diagnostic narratives.

Assumptions. (1) Majority correctness via frequency filtering ($\geq$5 tickets/node). (2) Extraction fidelity: 92% of contacts yield valid chains; 8% discarded. (3) Verifiability $\neq$ correctness: SQL agreement with ticket text at $\geq$70% validates operationalization, not original diagnosis correctness.

4.1 Stage 1: Diagnostic Chain Extraction

Given N resolved tickets, an LLM extracts a structured chain from each: a **symptom** (status or metric anomaly), a **cause chain** (sequence of causes at increasing depth explaining *why*), and an **action** (reusable resolution procedure). Constraints enforce that causes describe mechanisms (not outcomes), case-specific identifiers are stripped, and actions are reusable. Temperature 0 ensures deterministic extraction. A complete running example tracing one ticket through all five stages is in Appendix B.

4.2 Stage 2: Taxonomy Construction

Stage 2 produces an ungrounded *cause taxonomy*—a hierarchical tree of symptom families and their causes, without SQL queries. This taxonomy becomes a DKG after Stage 3 adds validated queries to each node.

Step 1—Symptom clustering. Unique symptom labels are embedded (a proprietary text embedding model, 512-d) and clustered at cosine similarity $\geq \tau_s$, merging surface variants. **Step 2—Semantic grouping.** An LLM groups top clusters into diagnostic families, catching equivalences embedding similarity misses. Families below 1% frequency are discarded. **Step 3—Tree generation.** Per family, an LLM organizes all observed causes into a hierarchical tree with frequencies. Depth is data-driven (typically 2–4 levels). The output is one cause taxonomy per family. Threshold selection details and sensitivity analysis are in Appendix H.

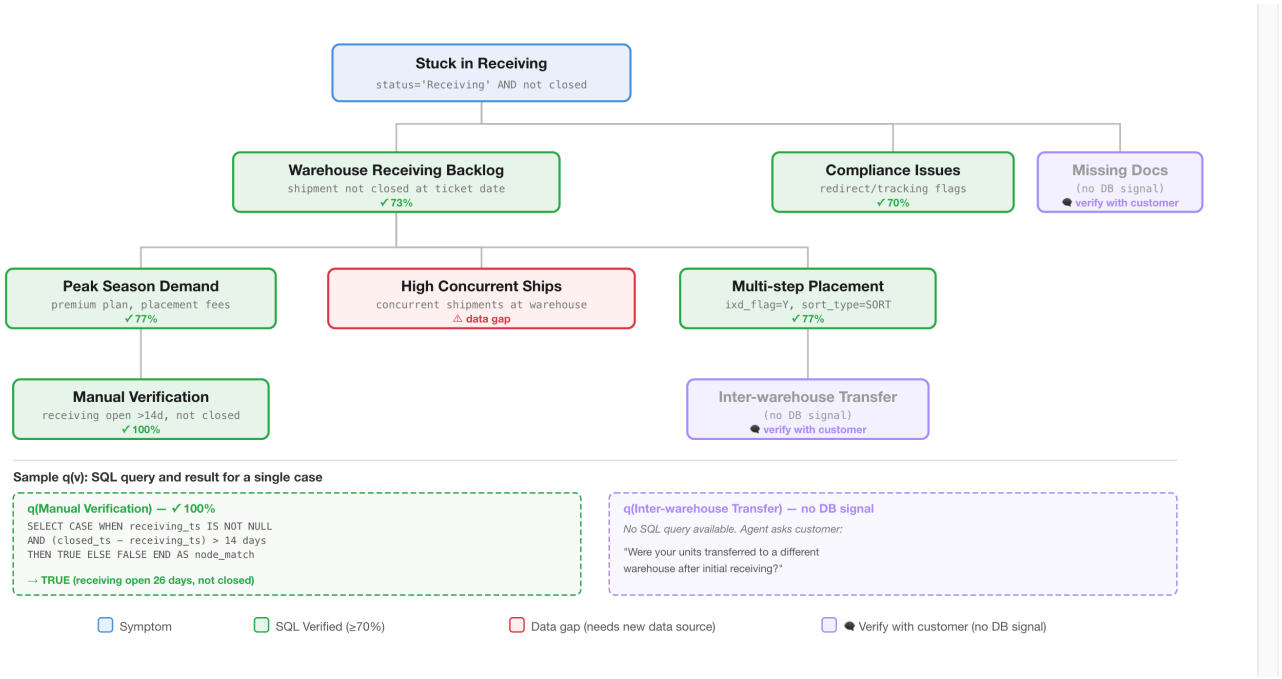

Figure 1: DKG for the running-example symptom family ("shipment stuck in receiving"): 14 cause nodes across three levels, with 8 nodes SQL-verified at 70–83% accuracy on 30 ground-truth tickets each. Colors indicate node status: green = SQL verified; red = known cause with no reliable database signal (*data gap*); purple = unverifiable via SQL (agent asks customer). Bottom panel shows two example node-level queries $q(v)$: a verified SQL check and a customer-facing qualitative check.

4.3 Stage 2b: Cause Normalization and Golden Chain Construction

A critical challenge: from 2,196 tickets about a single symptom family, LLM extraction produces 5,076 unique cause strings—the same underlying cause appearing as dozens of surface variants. Without normalization, evaluation requires expensive semantic comparison for every agent-vs-gold comparison.

Two-pass BERTopic clustering. For each symptom family, all causes across all levels are pooled and clustered in two passes: Pass 1 with HDBSCAN produces main clusters (leaving 30–40% as outliers); Pass 2 re-clusters outliers with relaxed density parameters. The **LLM merge** step then resolves semantic equivalences that no embedding threshold can catch (e.g., "barcode unscanned at intake" ≡ "manual entry required because barcode unreadable"—same root cause, near-zero token overlap). Merge criterion: two clusters merge *only* when they describe the same cause and would yield the same resolution action.

Result. 5,076 unique raw cause strings compress to 73 canonical clusters ($\rho = 70\times$, 97% coverage). Figure 3 visualizes the compression.

Normalized golden chains. Each raw extracted chain is normalized by applying the mappings:

$$(s, k_1, k_2, \ldots, k_d) \xrightarrow{\mu} (\hat{f}, \mu(k_1), \mu(k_2), \ldots, \mu(k_d)) \tag{2}$$

Table 1: Normalization quality across symptom families (5 of 10 shown; full table in supplement). ρ = compression ratio.

Symptom Family	Chains	Raw	Cl.	ρ	Cov.
Shipment qty discr.	2,196	5,076	73	70×	97%
Reimburs. not issued	667	1,412	25	57×	99%
Inventory recall unclear	536	1,203	58	21×	99%
Delivery unverifiable	357	714	8	89×	100%
Multi-SKU discrepancy	192	384	3	128×	100%
Avg (10 families)	**466**	**1,246**	**28**	**57×**	**99%**

where $\hat{f}$ is the matched symptom family and $\mu(k_i)$ maps each raw cause to its canonical cluster via nearest-centroid embedding match:

$$\mu(k) = \arg\min_{v \in V_f} d_{\cos}(\text{emb}(k), \text{centroid}(v)) \tag{3}$$

Multiple raw chains may normalize to the same golden path—this is the desired compression. The normalized chains constitute the evaluation benchmark: clean, canonical, and comparable via exact string match.

Scalability. Table 1 shows that the pipeline produces consistent normalization across the top 10 symptom families without manual curation.

Enabling deterministic evaluation. Once both the golden chain and the agent output reference the same canonical cluster vocabulary, evaluation reduces to string equality—deterministic, reproducible, and free of LLM-judge variance. We emphasize that

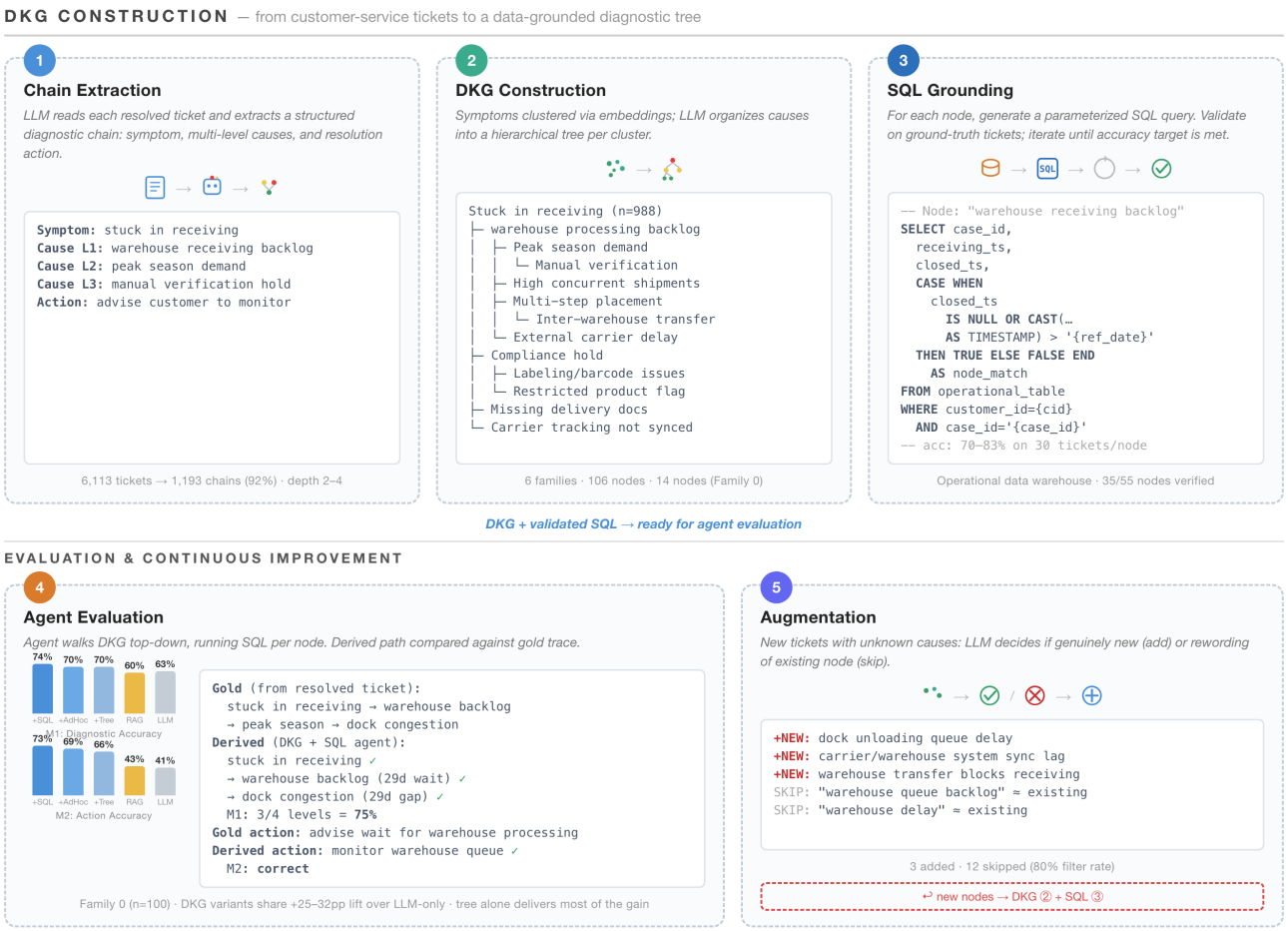

Figure 2: End-to-end pipeline illustrated on the running example ("shipment stuck in receiving," 988 chains). Top row: DKG construction from noisy tickets through chain extraction (Stage 1), taxonomy construction (Stage 2), and per-node SQL grounding (Stage 3). Bottom row: agent evaluation across five conditions with M1/M2 results for the receiving family (Stage 4), and augmentation with new causes triaged against the existing tree (Stage 5). Dashed arrow: new nodes from Stage 5 re-enter Stages 2 and 3.

this exact-match scoring is *not* used in the comparative experiments of Section 5; those experiments use a semantic LLM judge against raw gold chains to avoid giving the DKG-constrained agent a vocabulary-alignment advantage. The two modes serve different purposes: semantic-judge scoring for cross-method comparison, and exact-match scoring for longitudinal production monitoring.

4.4 Stage 3: Query Grounding

The key insight: **resolved ticket text serves as ground truth**. A ticket contains both the diagnosis and the operational details, creating a closed loop for query validation analogous to test-driven development—the ticket is the test case, the SQL query is the code under test. The grounding procedure runs once per node, offline, and produces a parameterized query executable without further LLM calls.

Phase A—Ground truth generation. For each node v, an LLM reads M=30 resolved tickets and labels every node in the family as TRUE/FALSE based on the ticket text, producing a label matrix $\mathbf{T} \in \{0,1\}^{M \times |V|}$. We use stratified sampling to ensure at least 10 positive and 10 negative tickets per node.

Phase B—Iterative SQL generation. For each node, a refinement loop (bounded by K=7 iterations) synthesizes a parameterized SQL template: (1) discriminating-column analysis identifies schema columns that separate TRUE from FALSE tickets; (2) an LLM generates a SQL template using placeholders `{cid}`, `{case_id}`, `{ref_date}`; (3) execution against locally cached data (in-memory DuckDB, ~10 ms per ticket); (4) refinement with failing examples if accuracy < 0.8. Full details in Appendix L.

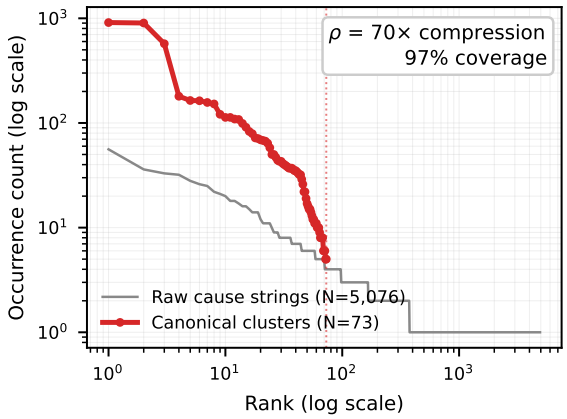

Figure 3: Rank-frequency distribution before and after clustering (largest symptom family, 2,194 tickets). Gray: 5,076 raw cause strings (92% appearing once). Red: 73 canonical clusters absorbing 97% of unique strings ($\rho = 70\times$ compression).

Verification thresholds. The 0.8 *break threshold* and 0.7 *verification threshold* differ deliberately: the break threshold stops overfitting; the verification threshold accounts for label noise ($\delta \approx 0.08$, yielding true accuracy ≥ 0.62). Nodes that fail to reach 0.7 are retained without SQL; the agent defers to children (Algorithm 1).

4.5 Stage 4: Agent Evaluation

The traversal design implements the error asymmetry principle from Section 3.4. Unlike a decision tree that commits to a single path at each split—where a single false negative derails the entire diagnosis—the DKG evaluates every node independently: multiple sibling causes can be simultaneously active. This yields higher expected coverage than greedy tree-walking (details in Appendix K).

Traversal. A deterministic algorithm executes SQL at every node top-down, collecting all TRUE matches; FALSE results are discarded. The matched set (with labels and evidence) is passed to an LLM in a single synthesis call. The LLM narrates the verified facts but does not decide which branches to explore. This design accepts a broader candidate set (more false positives) in exchange for not routing the agent away from the correct cause due to a false negative. The cost is measurable: on the running example family, SQL misdirection accounts for 12% of errors (Section 5), where broad parent queries fire alongside the correct specific cause.

4.6 Stage 5: Augmentation

The DKG evolves as new failure modes emerge. New chains are matched to existing families by embedding similarity; unmatched causes are collected as gap candidates. Frequency filtering (≥ 2 occurrences) removes one-off extraction noise without discarding genuine new low-frequency failure modes; the threshold is set conservatively at 2 because the LLM-triage step that follows is itself a high-precision filter (sensitivity over $\{1, 2, 3\}$ in Appendix H). The triage decides genuinely-new vs. paraphrase, with at most 3 new nodes added per family per round to prevent tree bloat. New nodes enter the Stage 3 SQL grounding loop before becoming active.

5 EXPERIMENTS

5.1 Setup

Data. The corpus consists of resolved customer-support contacts drawn from an internal logistics ticketing system, paired with internal operational warehouse data for the SQL-grounded subset. No personal or account identifiers are used or shown anywhere in this paper; the running-example ticket text in Appendix B is paraphrased and all identifiers are redacted. We use two scopes. The *benchmark construction scope* (Stages 1–2b) processes 14,953 contacts spanning all logistics categories, producing 284 symptom families with normalized golden chains—each a fully constructed DKG with cause taxonomy, frequency priors, and canonical cluster vocabulary usable for both deterministic evaluation and agent inference. The *SQL-grounded agent ablation* (Stages 3–4) uses 6,113 inbound logistics cases with matching data warehouse coverage, split into build (1,292), ground truth (252), and eval (100 per family). Stage 1 extracted 1,193 chains (92% success); Stage 2 produced 6 families surviving a 2% frequency threshold; cause clustering yielded 10–24 clusters per family. The five-condition ablation requires SQL grounding (Stage 3), limiting it to three families with sufficient data warehouse coverage; the DKG construction and normalization results (Tables 1, 3) are validated across all 284 families.

Evaluation families. The five-condition ablation (including SQL conditions) requires operational data warehouse coverage for query grounding; only three families have sufficient coverage: a *receiving* family ("shipment stuck in receiving," n=100), a *quantity-discrepancy* family (n=100), and a *transit* family ("shipment stuck in transit," n=65). We additionally evaluate DKG+Tree vs. LLM-only (no SQL) on three further families to confirm the tree-structural gain generalizes beyond the SQL-grounded subset (Section 5.3). **Models:** LLM-A (mid-capability; per-ticket work) and LLM-B (higher-capability; per-family work). Temperature 0 throughout.

Gold standard. To avoid giving the DKG agent a vocabulary-alignment advantage, gold chains are *raw* Stage 1 extractions (not DKG-normalized). Scoring uses a semantic LLM judge ("same mechanism?") applied identically across all conditions.

5.2 Baselines and Ablations

We evaluate five conditions on the same tickets per family, same LLM-A judge, temperature 0. **Equal-protocol top-3 scoring.** Every condition emits three ranked candidates (one JSON object each, ranked by agent confidence); the judge scores the best match against the gold chain—a generous protocol that mirrors how a real diagnostic system would surface alternatives to a human operator. The same top-3 scaffold is used across all conditions, ruling out an asymmetric "DKG gets 3 attempts, others get 1" confound. The conditions:

- **DKG+SQL (ours)**: DKG tree with frequency priors plus pre-validated SQL evidence (TRUE results only; FALSE discarded).
- **DKG+AdHocSQL**: same tree and priors, but the agent writes SQL at inference time against the operational schema.
- **DKG+Tree**: tree and priors only, no SQL—isolates the structured-hypothesis-space contribution.

Table 2: Main results across three SQL-grounded families (n=100 for stuck-in-receiving and quantity discrepancy; n=65 for stuck-in-transit). M1 = diagnostic accuracy, M2 = action accuracy (%). 95% bootstrap CIs in brackets. DKG+Tree = cause menu + frequency priors only (no SQL).

Family		DKG +SQL	DKG +AdHoc	DKG +Tree	RAG	LLM -only
Stuck in receiving	M1	**74** [67,81]	70 [63,76]	70 [63,77]	60 [55,66]	63 [57,69]
	M2	**73** [64,82]	69 [60,78]	66 [57,75]	43 [33,53]	41 [32,52]
Quantity discrepancy	M1	**70** [64,76]	70 [64,76]	70 [64,76]	58 [52,64]	37 [30,45]
	M2	**55** [45,65]	51 [41,61]	49 [40,59]	26 [18,35]	11 [5,18]
Stuck in transit	M1	58 [51,66]	56 [47,64]	57 [49,65]	56 [49,64]	55 [47,63]
	M2	38 [28,51]	**43** [31,55]	37 [26,49]	41 [29,52]	32 [22,43]

- **RAG**: 20 similar historical cases retrieved by embedding similarity, no tree; agent has schema access and writes SQL at inference time.
- **LLM-only**: issue text only, no cases or tree; agent has schema access and may write SQL.

5.3 Results

Table 2 presents results across all three SQL-grounded families. Structured diagnostic reasoning yields a large and unambiguous gain over unstructured baselines on families with concentrated frequency distributions: every condition that includes the DKG tree (DKG+SQL, DKG+AdHocSQL, DKG+Tree) reaches 49–73% action accuracy on the receiving and quantity-discrepancy families, vs. 11–41% for LLM-only—a +25–38pp lift attributable primarily to the structured hypothesis space and frequency priors. On the harder transit family, all methods converge to 32–43% with overlapping CIs, revealing the boundary condition. Within the structured family, adding pre-validated SQL (DKG+SQL) is not statistically distinguishable from the no-SQL ablation (DKG+Tree) at n=100; their bootstrap CIs overlap.

Generalization without SQL. To confirm that the tree-structural gain extends beyond families with data warehouse coverage, we evaluate DKG+Tree vs. LLM-only on three additional families from a broader ticket corpus, without SQL grounding (Table 3). These families are distinct from those in Table 2 (different ticket pool and pipeline scope). The DKG agent receives only the cause taxonomy and frequency priors; the LLM-only baseline receives the ticket text and database schema and may write SQL at inference time, with no DKG structure (identical to the LLM-only condition in Table 2). As a directional generalization check (n=100 per family), the pattern is consistent: DKG+Tree outperforms LLM-only on all three families—separated clearly on the inventory-recall family and directionally on the other two—with the largest lift on the family with the most concentrated frequency distribution.

Key findings. The five-condition ablation yields three clear conclusions:

- *Tree structure with frequency priors drives most of the gain.* DKG+Tree (cause menu and priors only, no SQL) reaches 66% on the receiving family and 49% on the quantity-discrepancy family—accounting for 78% of the +32pp gain (receiving) and 86% of the +44pp gain (qty) over LLM-only. Constraining the agent to historically observed causes is the dominant lever. This is the same artifact that normalizes the evaluation benchmark, demonstrating the duality thesis empirically.
- *SQL evidence is directionally positive but not statistically distinguishable on top of the tree.* On the receiving family, DKG+SQL (73%) is within DKG+Tree's CI [57,75] and DKG+AdHocSQL's [60,78]; the quantity-discrepancy family's three structured conditions span 49–55% with overlapping CIs. Pre-validated SQL adds a consistent directional benefit that a power calculation indicates would require $n \approx 900$ per condition to confirm at α=0.05.
- *Harder families resist all methods, revealing boundary conditions.* On the transit family (n=65), all five conditions span 32–43% action with overlapping CIs; DKG+Tree reaches only 37% (vs. 32% LLM-only)—a +5pp lift versus +25–38pp on the other two families. The tree-structural gain shrinks when symptom diversity outpaces frequency priors, providing a predictable boundary condition observable from the canonical vocabulary's rank-frequency distribution.

Table 3: DKG+Tree vs. LLM-only on three additional families without SQL (n=100 each, different ticket pool from Table 2). M1 = diagnostic accuracy, M2 = action accuracy (%); best per row in bold. 95% CIs in brackets.

Family		DKG +Tree	LLM -only
Inbound qty discrepancy	M1	**80** [72,88]	73 [64,82]
	M2	**75** [67,83]	72 [63,81]
Reimburs. not issued	M1	**65** [56,74]	45 [35,55]
	M2	**52** [42,62]	47 [37,57]
Inventory recall unclear	M1	**78** [70,86]	47 [37,57]
	M2	**70** [61,79]	35 [26,44]

5.4 Benchmark Quality and Meta-Evaluation

We evaluate two properties critical for benchmark utility: (1) normalization quality (does the clustering preserve diagnostic meaning?) and (2) faithfulness compared to the LLM judge it can replace.

Normalization coverage and compression. On the largest symptom family from the benchmark-construction scope (shipment quantity discrepancy, 2,196 chains; distinct from the receiving family used in agent eval), two-pass BERTopic + LLM merge compresses 5,076 unique raw cause strings to 73 canonical clusters with 97% coverage. Three raw strings extracted from different tickets about the same underlying issue—"warehouse receiving backlog delayed processing," "warehouse processing delay during inbound," "distribution center has not yet processed inbound shipment"—all map to a single canonical cluster, reducing agent-vs-gold comparison from LLM-judged fuzzy match to exact equality. Table 1 confirms this generalizes across the top 10 families (average 57× compression, 99% coverage) and Figure 3 visualizes the compression on the largest family.

Human validation of normalization accuracy. A domain expert reviewed 50 randomly sampled cause-level mappings (raw

extracted string → canonical DKG cluster) and judged each as correct, partially correct, or incorrect. The normalization achieves **92% strict accuracy** (46/50 mappings judged correct), compared to only 80% agreement between the LLM semantic judge and the same expert on 20 sampled diagnostic-pair comparisons (Cohen's κ = 0.60; Appendix I). This 12pp gap demonstrates that the normalized benchmark is *more faithful to expert judgment* than the LLM judge it replaces.

Inter-run variance. Evaluation against the normalized benchmark is deterministic by construction (σ=0 across any number of runs), whereas the LLM judge exhibits ±2–3pp run-to-run variance in M2 on identical inputs (Appendix I). Such variance makes it impossible to detect small (<3pp) performance drifts in longitudinal monitoring—precisely the regime where production teams need signal. Together, these properties—92% expert faithfulness plus σ=0—make the normalized benchmark suitable for longitudinal production monitoring where small performance changes must be reliably detected.

5.5 Error Decomposition and Localization

A key advantage of per-step evaluation is *error localization*: identifying not just *that* the agent failed, but *where* and *why*. We demonstrate this on the 27 failure cases (tickets where DKG+SQL produced incorrect action) on the receiving family.

Error taxonomy. Of 16 analyzed failures on the receiving family (DKG+SQL): *SQL misdirection* (12/16, 75%)—broad parent queries confirm a general cause when the true root cause is more specific (e.g., SQL confirms "warehouse processing delay" when the actual issue is barcode scan failure, a child node); *tree gap* (2/16, 12.5%)—the gold root cause is absent from the DKG, addressable through augmentation (Stage 5); *ticket misclassification* (2/16, 12.5%)—the gold chain itself is incorrect (noise floor from 8% extraction error rate).

Per-level diagnostic patterns. The canonical vocabulary enables pattern-level failure diagnosis:

- *High L1 + low L2*: Agent identifies the correct problem family but fails to reach the specific sub-cause. Addressable by improving L2-level SQL discrimination.
- *Low L1*: Agent mis-identifies the problem entirely. Indicates either a symptom-routing error or a ticket that does not match its assigned family.
- *High M1 + wrong M2*: Agent identifies correct reasoning steps but maps to an incorrect action. Indicates action-mapping errors in the DKG structure, not reasoning failures.

This granularity distinguishes lucky guesses (wrong reasoning, correct action by chance) from sound reasoning that fails on the final step—a distinction only possible with per-step scoring against a canonical vocabulary.

6 DISCUSSION

The evaluation-agent duality. The cause taxonomy constrains the agent to check *whether* each hypothesis holds rather than deciding *what* to check—preventing narrative fabrication (+32–44pp gap). The same constraint eliminates vocabulary explosion for evaluation: outputs from a closed vocabulary enable exact match. This is a structural property, not a methodological convenience.

What SQL contributes. At n=100, no structured condition is statistically distinguishable from DKG+Tree on action accuracy. SQL provides deployment-time reliability (auditable evidence, schema-error containment) rather than diagnostic-accuracy lift. The gap is bounded by data sparsity: 30–43% of nodes per family lack sufficient coverage for SQL grounding.

Boundary conditions. The DKG's effectiveness depends on frequency concentration: when causes are heavily skewed (receiving family), frequency priors provide +32pp lift; when uniformly distributed (transit family), only +5pp. This boundary is predictable from the canonical vocabulary's rank-frequency distribution.

Limitations. (1) Agent evaluation spans 6 families (565 tickets): 3 with full SQL grounding and 5-condition ablation (n=265), 3 additional with DKG+Tree vs. LLM-only (n=100 each); DKG construction and normalization validated on all 284 families. (2) Expert validation at n=50 (normalization) and n=20 (judge); larger studies would strengthen confidence. (3) Production monitoring validated in prerequisites (92% faithfulness, σ=0) but not demonstrated over an actual deployment timeline. (4) Requires resolved cases with diagnostic narratives plus an operational database. (5) 8% normalization error rate from embedding-model biases.

7 CONCLUSION

We presented an automated pipeline transforming noisy resolved tickets into both a clean diagnostic benchmark and a SQL-grounded reasoning scaffold. Two-pass BERTopic clustering with LLM merge compresses extreme vocabulary explosion (5,076 → 73 clusters, 97% coverage) without manual curation, producing normalized golden chains with 92% expert agreement and zero inter-run variance. The evaluation-agent duality is our central finding: the canonical vocabulary that normalizes the benchmark accounts for 78–86% of a +32–44pp action-accuracy gain when used as an inference-time menu. The pipeline scales to 284 families from 14,953 tickets and generalizes to any domain with abundant resolution traces.

REFERENCES

[1] Yunsheng Bai and Haoxing Ren. 2025. Learning to Debug: LLM-Organized Knowledge Trees for Solving RTL Assertion Failures. *arXiv preprint arXiv:2511.17833* (2025).

[2] Haonan Bian. 2025. LLM-empowered Knowledge Graph Construction: A Survey. *arXiv preprint arXiv:2510.20345* (2025).

[3] Chang Cai, Zhengyi Jiang, Hui Wu, Junsheng Wang, Jiawei Liu, and Lei Song. 2024. Research on knowledge graph-driven equipment fault diagnosis method for intelligent manufacturing. *The International Journal of Advanced Manufacturing Technology* 130 (2024), 4649–4662. https://doi.org/10.1007/s00170-024-12998-x

[4] Mohammadreza Chavoshi, Hari Trivedi, Janice Newsome, Aawez Mansuri, Chiratidzo Rudado Sanyika, Rohan Satya Isaac, Frank Li, Theo Dapamede, and Judy Gichoya. 2025. Impact of Label Noise from Large Language Models Generated Annotations on Evaluation of Diagnostic Model Performance. *arXiv preprint arXiv:2506.07273* (2025).

[5] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130* (2024).

[6] Maarten Grootendorst. 2022. BERTopic: Neural Topic Modeling with a Class-Based TF-IDF Procedure. In *arXiv preprint arXiv:2203.05794*.

[7] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. 2022. Knowledge Graphs. *Comput. Surveys* 54, 4 (2022), 1–37.

[8] Tunazzina Islam. 2026. Reasoning-Based Refinement of Unsupervised Text Clusters with LLMs. *arXiv preprint arXiv:2604.07562* (2026).

[9] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications.

IEEE Transactions on Neural Networks and Learning Systems 33, 2 (2022), 494–514.
[10] Fangyu Lei et al. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *arXiv preprint arXiv:2411.07763* (2024).
[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.
[12] Subhrangshu Nandi, Arghya Datta, Rohith Nama, Udita Patel, Nikhil Vichare, et al. 2025. SOP-Bench: Complex Industrial SOPs for Evaluating LLM Agents. *arXiv preprint arXiv:2506.08119* (2025).
[13] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. Root Cause Analysis for Microservice System based on Causal Inference: How Far Are We?. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.
[14] Tingting Wang, Guilin Qi, and Tianxing Wu. 2024. KGroot: Enhancing Root Cause Analysis through Knowledge Graphs and Graph Convolutional Neural Networks. *arXiv preprint arXiv:2402.13264* (2024).
[15] Zhiqiang Xie, Yujia Zheng, Lizi Ottens, Kun Zhang, Christos Kozyrakis, and Jonathan Mace. 2024. Cloud Atlas: Efficient Fault Localization for Cloud Systems using Language Models and Causal Insight. *arXiv preprint arXiv:2407.08694* (2024).
[16] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering. *arXiv preprint arXiv:2404.17723* (2024).
[17] Xuejiao Zhao, Siyan Liu, Su-Yin Yang, and Chunyan Miao. 2025. MedRAG: Enhancing Retrieval-augmented Generation with Knowledge Graph-Elicited Reasoning for Healthcare Copilot. *arXiv preprint arXiv:2502.04413* (2025).
[18] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*.
[19] Hongli Zhou, Hui Huang, Rui Zhang, Kehai Chen, Bing Xu, Conghui Zhu, Tiejun Zhao, and Muyun Yang. 2026. Toward Robust LLM-Based Judges: Taxonomic Bias Evaluation and Debiasing Optimization. *arXiv preprint arXiv:2603.08091* (2026).
[20] Yue Zhou, Mingxin Zhang, and Meng Zha. 2025. Knowledge-Graph-Driven Fault Diagnosis Methods for Intelligent Production Lines. *Sensors* 25, 13 (2025), 3912.
[21] Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2024. LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities. *World Wide Web* 27, 5 (2024).
[22] Kaiwen Zuo, Yirui Jiang, Fan Mo, and Pietro Lio. 2024. KG4Diagnosis: A Hierarchical Multi-Agent LLM Framework with Knowledge Graph Enhancement for Medical Diagnosis. *arXiv preprint arXiv:2412.16833* (2024).

A DKG NODE ACTIONS

Table 4 lists the 14 cause nodes of the receiving family's DKG (v2 taxonomy) with their SQL verification accuracy and the recommended agent action. SQL accuracy is reported for the 8 nodes that reached the ≥70% verification threshold; "UV" (unverified) marks nodes that ran the SQL grounding loop without converging; "no GT" marks nodes excluded from the SQL loop because the stratified ground-truth sampler could not produce ≥10 positive examples (frequency too low).

B RUNNING EXAMPLE

We trace a single ticket through the full pipeline to make each stage concrete.

Input ticket. A customer contacts support: "My shipment was delivered to the warehouse 8 days ago but still shows 'Receiving' status. When will my items be available?"

Stage 1 (Extraction). The LLM extracts: *Symptom*: "shipment stuck in receiving status"; *Cause*: "warehouse receiving backlog due to elevated inbound shipment volume"; *Deeper cause*: "peak season demand causing processing queue depth to exceed standard window"; *Action*: "advise customer on standard receiving SLA and escalate if beyond window."

Stage 2 (Taxonomy). The symptom clusters with 987 other "stuck in receiving" variants (cosine ≥ 0.85) into the receiving family (988 chains total). The cause "warehouse receiving backlog" becomes node N0; "peak season demand" becomes N1 under N0.

Stage 3 (SQL Grounding). For node N1, the LLM generates a query checking whether the shipment arrived during peak months (Oct–Feb) and has waited more than 4 days. Tested against 30 ground-truth tickets: 24/30 correct (80%), verified after 1 iteration.

Stage 4 (Evaluation). The agent walks the tree: ROOT matches, N0 matches (8 days waiting), N1 matches (December arrival). Derived path matches gold path → 100% coverage.

Stage 5 (Augmentation). A new ticket mentions "pre-scan processing delay." Cosine similarity to existing children is < 0.70, so it enters LLM triage and is added as a new node.

C DKG SYMPTOM FAMILIES

Table 5 summarizes the 12 candidate symptom families produced by the pipeline and the 6 that survived the 2% frequency threshold.

D RAG BASELINE DETAILS

The RAG baseline operates in two LLM calls with database access between them.

Retrieval. Historical diagnostic chains are embedded using a proprietary text embedding model (1024 dimensions, L2-normalized). At evaluation time, the customer's issue text is embedded and the top-20 chains by cosine similarity are retrieved (excluding chains from the same contact).

Step 1: Query generation. The agent receives the customer's issue, 20 retrieved cases, the full operational schema with domain context, and case identifiers. It generates 1–3 SQL queries and an initial hypothesis.

Step 2: Synthesis. Query results are returned to the agent, which produces a diagnostic path as a list of short labels.

Key differences from DKG+SQL. The RAG agent must (1) decide *what* to check, (2) write correct SQL from schema documentation, and (3) interpret raw query results. The same LLM and evaluation metric are used for both conditions.

E PER-TICKET RESULTS

Table 6 shows per-ticket results for the first 10 evaluation tickets on the receiving family, drawn from the *same evaluation run* that produced Table 2. "M1 unord." is the unordered semantic step coverage (best of top-3 candidates) reported as the headline M1 metric. "Action" is M2 (✓/×). The 10-ticket sample is unstratified, so action accuracies for the easier-to-handle baselines (RAG, LLM-only) on this prefix are higher than their full n=100 averages; the DKG+SQL action rate (70% vs. 73%) is consistent.

F TRANSIT FAMILY DISCUSSION

The transit family ("shipment stuck in transit," n=65) is the hardest in our evaluation (Table 2, bottom rows), where all five methods achieve overlapping confidence intervals on action accuracy and the tree-driven gain over LLM-only is markedly smaller than on the receiving and quantity-discrepancy families (DKG+Tree at 37% vs. LLM-only at 32%, only +5pp). This is the regime described in Section 5.3 where neither structural priors alone nor SQL grounding

Table 4: Receiving family ("shipment stuck in receiving") v2 cause nodes with SQL verification results. Frequencies count the number of build-set chains in which the cause appears. † marks leaf nodes (actionable root causes).

Node	Cause label (abbr.)	Freq	SQL	Recommended Action
N0	Warehouse inbound processing delay (post-delivery, pre-availability)	900	80%	Advise standard receiving SLA; flag if past window
N2†	Standard warehouse processing delay	453	77%	Advise standard SLA
N4†	Multi-step inventory placement / redistribution	98	77%	Explain sort-and-place pipeline; set ETA
N5†	Multi-step warehouse placement pipeline	61	77%	Same as N4; site-specific timing
N6†	Inter-warehouse redistribution transfer	37	80%	Monitor inter-site transfer completion
N7†	Warehouse manual inspection / verification hold	50	77%	Advise inspection in progress
N10†	Carrier-to-warehouse tracking sync issues	85	70%	Verify tracking refresh; request POD if stale
N11†	External carrier tracking sync failure	61	83%	Request warehouse-stamped POD
N1	Warehouse receiving queue backlog (parent)	735	UV	Defer to children N2–N7
N3	Warehouse receiving backlog (high inbound volume)	282	UV	Set peak-season expectations
N8	Warehouse inbound dock queue backlog	6	UV	Advise dock congestion if applicable
N9	No expedited receiving channel	11	UV	Inform of standard processing only
N12	External carrier reducing warehouse visibility	24	no GT	Request customer-provided tracking docs
N13	Receiving blocked by incomplete processing	3	no GT	Manual escalation

SQL: ≥70% accuracy on 30 ground-truth tickets; UV = unverified (<70% after 7 iters);
no GT = excluded from SQL loop due to insufficient positive examples (<10).

Table 5: DKG symptom families derived from 1,193 build-set diagnostic chains. Frequencies count chain-to-family symptom matches. 6 of 12 candidates survived the 2% frequency threshold (bold).

#	Symptom Family	Freq	Nodes	Eval
0	**Shipment stuck in receiving**	**988**	**14**	***n*=100**
1	**Quantity discrepancy**	**536**	**21**	***n*=100**
2	**Stuck in transit**	**148**	**20**	***n*=65**
3	**Items marked unavailable**	**62**	**13**	—
4	**Shipment closed unexpect.**	**61**	**24**	—
5	**Available inventory dropped**	**33**	**14**	—
	6 discarded (freq < 2%)	<23	—	—
	Total (6 families)		**106**	

Table 6: Per-ticket results (receiving family, first 10 tickets, same run as Table 2). $M1_u$ = unordered step coverage (best of top-3). Act. = M2.

	DKG+SQL		RAG		LLM-only	
#	$M1_u$	Act.	$M1_u$	Act.	$M1_u$	Act.
1	100%	✓	100%	✓	100%	✓
2	100%	✓	67%	×	67%	×
3	100%	✓	100%	✓	33%	×
4	67%	×	100%	✓	0%	×
5	100%	×	67%	✓	100%	✓
6	100%	✓	67%	×	33%	✓
7	100%	✓	100%	✓	100%	✓
8	67%	×	100%	✓	100%	✓
9	33%	✓	33%	✓	50%	×
10	67%	✓	0%	×	50%	✓
Avg (10)	**83%**	**70%**	**73%**	**70%**	**63%**	**60%**
Avg (100)	**74%**	**73%**	**60%**	**43%**	**63%**	**41%**

helps appreciably—symptom diversity is high, frequency priors are flatter, and the relevant operational signals are spread across more tables.

G AUGMENTATION DECISIONS

Table 7 shows LLM triage decisions on the top 15 gap candidates from a fresh set of 100 chains, illustrating which novel causes were accepted versus filtered as duplicates.

H CLUSTERING AND AUGMENTATION THRESHOLDS

Two cosine similarity thresholds control clustering ($\tau_s = 0.85$ for symptom merging) and augmentation ($\tau_a = 0.70$ for gap detection). We selected τ_s by sweeping $\{0.70, ..., 0.95\}$ on 200 labeled symptom pairs, maximizing F1 (precision 0.92, recall 0.87; Table 8). We set τ_a lower because the LLM triage step serves as a high-precision second filter (Table 9).

Lower thresholds over-merge distinct symptoms (e.g., "stuck in receiving" and "stuck in transit" merge at 0.75). Higher thresholds fragment families into near-duplicates.

Table 7: LLM triage for top 15 gap candidates (receiving family, 100 new chains).

Gap Cause	Decision	Reason
Pre-scan processing delay	New	Distinct pre-scan step
System synchronization lag	New	System sync ≠ physical delay
Transfer blocks receiving	New	Distinct from placement routing
Warehouse queue delay	Duplicate	≈ Processing backlog
Receiving queue backlog	Duplicate	≈ Processing backlog
Delayed due to high volume	Duplicate	≈ Peak season demand
... 9 more duplicates filtered ...		

Table 8: Sensitivity of symptom clustering threshold τ_s.

τ_s	Families	Total Nodes	Merged Errors
0.75	8	98	4/20
0.80	9	121	2/20
0.85	**11**	**145**	**1/20**
0.90	14	178	0/20
0.95	19	231	0/20

Table 9: Sensitivity of augmentation threshold τ_a (receiving family, 100 new chains).

τ_a	Gaps to Triage	New Nodes	False Gaps
0.60	142	3	127/142
0.70	**74**	**3**	**59/74**
0.80	31	2	17/31
0.90	8	0	3/8

I HUMAN EVALUATION OF LLM JUDGE

We validated both the LLM judge and the normalization pipeline against domain-expert judgments.

LLM judge accuracy. A domain expert reviewed 20 diagnostic label pairs sampled from DKG+SQL and LLM-only evaluation results, judging each as "same mechanism" or "different mechanism." The LLM-A judge achieved 80% agreement with the expert (Cohen's $\kappa = 0.60$, moderate agreement). Disagreements occurred on parent-child relationships (e.g., "manual inspection" as a type of "receiving delay") and stage distinctions—domain-specific nuances where the judge lacks operational context.

Normalization accuracy. Separately, the same expert reviewed 50 randomly sampled cause-level normalization mappings (raw extracted string → canonical DKG cluster), judging each as correct, partially correct, or incorrect. The normalization achieves **92% strict accuracy** (46/50 correct), substantially exceeding the LLM judge's 80% agreement. This gap arises because normalization leverages embedding similarity against a fixed cluster vocabulary, whereas the LLM judge must resolve semantic equivalence from scratch on every comparison.

Inter-run variance. We quantified judge instability by running the LLM-A judge three times on the same 20-ticket evaluation set; M2 (action accuracy) varied by ±2–3pp across runs. We report this at temperature 0.3 to make the instability visible, but emphasize that temperature 0 does not remove it: API-served LLMs are not bit-reproducible across runs—provider-side batching and routing introduce nondeterminism—so reproducibility cannot be guaranteed by lowering temperature alone. Evaluation against the normalized benchmark is instead deterministic by construction (σ=0), which is precisely why it, rather than a lower judge temperature, is the appropriate basis for longitudinal monitoring where sub-3pp drifts must be detected.

J ADDITIONAL EVALUATION METRICS

In addition to diagnostic accuracy (M1) and action accuracy (M2) reported in the main text, we define two additional metrics.

M3: Ordered path accuracy. Did the agent identify the correct diagnostic steps *in the correct causal order*?

$$\text{OrderedAcc}(\mathbf{g}, \mathbf{d}) = \frac{1}{|\mathbf{g}|} \sum_{i=1}^{|\mathbf{g}|} \mathbf{1}\big[\text{sem}(g_i, d_i) = 1\big] \tag{4}$$

where d_i is the i-th element of the derived path (or $\perp$ if $|\mathbf{d}| < i$). M1 $\geq$ M3 by construction.

M4: Root cause accuracy. Did the agent identify the correct deepest root cause?

$$\text{RootCauseAcc}(\mathbf{g}, \mathbf{d}) = \mathbf{1}\big[\exists\, d_j \in \mathbf{d},\ \text{sem}(g_{|\mathbf{g}|}, d_j) = 1\big] \tag{5}$$

K TRAVERSAL ALGORITHM

Algorithm 1 DKG-Guided Diagnostic Agent

Require: DKG $\mathcal{G} = (V, E, q, a)$, ticket case identifiers x, reference date t
Ensure: Diagnostic narrative with matched causes and recommended actions

$\mathcal{M} \leftarrow \emptyset$ ▷ Matched nodes
$\mathcal{E} \leftarrow \emptyset$ ▷ Evidence (query results)
procedure TRAVERSE(node v)
  if $q(v) \neq \perp$ **then** ▷ Node has SQL query
    $r_v \leftarrow \text{EXECUTE}(q(v), x, t)$ ▷ Run parameterized SQL
  else
    $r_v \leftarrow \perp$ ▷ Non-verifiable; defer to children
  end if
  if $r_v = \text{TRUE}$ **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{v\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v, r_v)\}$
  end if
  for each child u s.t. $(v, u) \in E$ **do** ▷ Evaluate *all* children
    TRAVERSE(u)
  end for
  if $r_v = \perp$ **and** $\exists\, u \in \text{children}(v) : u \in \mathcal{M}$ **then** ▷ Child inference
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{v\}$
  end if
end procedure
TRAVERSE(root($\mathcal{G}$))
$\mathcal{A} \leftarrow \{a(v) : v \in \mathcal{M} \cap \text{leaves}(V)\}$ ▷ Collect leaf actions
return LLM-SYNTHESIZE($\mathcal{M}, \mathcal{E}, \mathcal{A}$) ▷ Single LLM call

K.1 Augmentation Convergence

Let C_t denote the fraction of tickets covered after t augmentation rounds, and α_t the fraction of remaining gaps captured in round t. Each round satisfies $C_{t+1} = C_t + (1 - C_t) \cdot \alpha_t$, so $1 - C_t \leq \prod_{i=1}^{t}(1 - \alpha_i)$. If $\alpha_t \geq \alpha_{\min} > 0$, coverage converges exponentially: $1 - C_t \leq (1 - \alpha_{\min})^t$. Empirically, $C_0 = 0.97$ and $\alpha_1 \approx 1.0$ (all 3 novel causes captured in one round).

L SQL GROUNDING ALGORITHM

Algorithm 2 details the iterative SQL grounding procedure summarized in Section 4.4.

Algorithm 2 Test-Driven SQL Grounding

Require: Cause taxonomy V, ticket pool $\mathcal{D}$, database schema $\mathcal{S}$, accuracy threshold $\theta = 0.7$, max iterations $K = 7$
Ensure: Grounded DKG with verified SQL queries

Phase A: Ground Truth Generation

for each node $v \in V$ **do**
  Sample $M = 30$ tickets (stratified: ≥10 TRUE examples)
  $y_{i,v} \leftarrow$ LLM-Classify(ticket$_i, v$) for each ticket i ▷ TRUE/FALSE from ticket text
end for

Phase B: Iterative SQL Generation

for each node $v \in V$ **do**
  $\mathcal{R} \leftarrow$ BatchFetch$(\mathcal{S}, \{x_i\}_{i=1}^{M})$ ▷ Single DB call; cache locally
  $\Delta \leftarrow$ ColumnDiff$(\mathcal{R}, \mathbf{y}_v)$ ▷ Columns that discriminate TRUE vs FALSE
  for $k = 1, \ldots, K$ **do**
    $q_k(v) \leftarrow$ LLM-GenSQL$(v, \Delta, \{q_j\}_{j<k}, \text{failures}_{k-1})$
    $\text{Acc}_k \leftarrow \frac{1}{M}\sum_i \mathbf{1}[q_k(v)(\mathcal{R}_i) = y_{i,v}]$ ▷ Evaluate on cached data via a local SQL engine
    if $\text{Acc}_k \geq \max(\theta, 0.8)$ **then**
      break ▷ Verified
    end if
    failures$_k \leftarrow$ per-ticket results + raw data for 3 failing cases
  end for
  Mark v as *verified* if $\text{Acc}_K \geq \theta$, else *non-verifiable*
end for

M SQL GROUNDING RESULTS AND VALIDATION GATES

Table 10 shows v1 taxonomy results for reference. Table 11 shows the v2 taxonomy results used in all experiments.

M.1 Validation Gates

The DKG is validated through three successive gates:

Gate 1: Frequency validation. Every leaf node must have ≥5 supporting tickets. Nodes below threshold are pruned, with tickets rolled up to the parent. For the receiving family, all 9 nodes (v1 taxonomy) passed with 24–222 supporting tickets.

Table 10: SQL grounding results, v1 taxonomy (receiving family, 9 nodes, 30 ground-truth tickets per node). Max iterations = 7.

Node	Cause	Acc	Iters	Status
ROOT	Stuck in receiving	73%	7	indirect
N0	Warehouse receiving backlog	73%	7	indirect
N1	Peak season demand	77%	7	near-threshold
N2	Manual verification	100%	3	✓verified
N3	Dock congestion	—	7	non-verifiable
N4	Multi-step placement	77%	7	near-threshold
N5	Pending transfer	—	—	no positive cases
N6	Compliance issues	70%	7	near-threshold
N7	Missing delivery docs	—	—	no positive cases
Summary		1 verified, 2 indirect, 3 near-thr., 3 non-verif.		

Table 11: SQL grounding summary, v2 taxonomy (used in all experiments). Verified = ≥70% boolean accuracy.

Family	Nodes	Verified	Unverif.	Acc Range	Tickets
0: Stuck in receiving	14	8	4	70–83%	988
1: Qty discrepancy	21	13	4	70–87%	536
2: Stuck in transit	20	14	2	70–100%	148
Total	55	35	10		

Verified + Unverif. counts nodes with ground truth; remaining 10 excluded (insufficient positive examples).

Gate 2: SQL verifiability. Each node must achieve ≥70% boolean accuracy on a test-driven SQL loop.

V1 taxonomy results (reported in Table 10): The receiving family had 9 nodes with 252 ground-truth tickets (30 per node). Results: 1/9 verified at ≥80%, 2 indirect (73%), 3 near-threshold (70–77%), 3 non-verifiable.

V2 taxonomy results (used in all experiments in Section 5): After cause re-clustering, the receiving family has 14 nodes with 8/12 verified (≥70%), the quantity-discrepancy family has 21 nodes with 13/17 verified, and the transit family has 20 nodes with 14/16 verified. The v2 taxonomy achieves higher verification rates due to improved cause clustering that produces more SQL-distinguishable nodes.

Gate 3: End-to-end coverage. The DKG+SQL agent achieves 74% diagnostic accuracy and 73% action accuracy on 100 held-out tickets (receiving family).

M.2 Error Analysis

Action accuracy (73%) substantially exceeds ordered diagnostic accuracy (55%) because correct actions often require only partial depth. The tree's frequency priors guide the agent to the correct branch in most cases, and the correct branch determines the action even without reaching the deepest cause. For example:

> *Gold trace (4 levels):* stuck in receiving → warehouse backlog → high inbound volume → sequential sorting/weighing required
> *Derived path (2 levels):* stuck in receiving → warehouse backlog

Action: "advise customer to wait for warehouse processing"
→ **correct**

The deeper gold levels explain *why* the warehouse is backlogged but do not change the resolution. Conversely, 4 of the 27 failure cases (n=100) have high coverage but wrong action: the agent identifies most steps but misidentifies the sub-cause, leading to a wrong recommendation. **Action accuracy depends on identifying the correct branch, not the correct depth.**