Knowledge Graphs as the Missing Data Layer for LLM-Based Industrial Asset Operations

Madhulatha Mandarapu
VaidhyaMegha Private Limited
India
madhulatha@samyama.ai

Sandeep Kunkunuru
VaidhyaMegha Private Limited
India
sandeep@samyama.ai

ABSTRACT

LLM-based agents for industrial asset operations show limited accuracy when reasoning over flat document stores. AssetOpsBench [14] (KDD 2026) establishes that GPT-4 agents achieve 65% on 139 industrial maintenance scenarios, and compares LLM orchestration paradigms (Agent-As-Tool vs. Plan-Execute) on a fixed data layer. We ask the orthogonal question: *how much does the data model behind the tools matter?*

We treat a typed knowledge graph as a *grounding substrate* and route each question by how it is best answered: (i) LLM-generated Cypher for structured retrieval, which lifts the same GPT-4 model from 65% to 82% (82–83% across GPT-4/4o/4.1); (ii) native graph and optimization primitives, with no LLM, reaching 99% on graph-answerable scenarios; and (iii) generation-augmented knowledge (GAK) [10] for answers absent from the data—the engine's agent materializes the missing facts as provenance-tagged graph nodes, then answers. A recurring theme is **inverted LLM usage**: we constrain the LLM to query generation or one-shot enrichment from a typed schema and let the graph execute deterministically.

On the 88 real AssetOpsBench failure-mode scenarios the benchmark itself flags as non-deterministic—ten equipment types absent from the graph—GAK lifts answerability from zero to 100% of equipment types and answers 81.8% of scenarios, every materialized fact tagged `source:LLM-derived` for auditability. We also contribute 40 graph-native scenarios and evaluate against the expanded 467-scenario HuggingFace release. For structured operational domains the data layer—not the LLM orchestration—is the primary lever, and a typed knowledge graph serves as a grounding substrate between raw industrial data and LLM reasoning.

VLDB Workshop Reference Format:
Madhulatha Mandarapu and Sandeep Kunkunuru. Knowledge Graphs as the Missing Data Layer for LLM-Based Industrial Asset Operations. VLDB 2026 Workshop: Agent+Graph.

VLDB Workshop Artifact Availability:

The source code, data, and/or other artifacts have been made available at https://github.com/samyama-ai/assetops-kg.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

1 INTRODUCTION

Industrial asset management generates vast quantities of structured data: sensor telemetry, work orders, failure-mode analyses, equipment hierarchies, and maintenance schedules. The rise of Large Language Models (LLMs) [11] has prompted efforts to build autonomous agents that can reason over this data—answering operational questions, predicting failures, and recommending maintenance actions.

AssetOpsBench [14] (KDD 2026) provides the first systematic evaluation of such agents, benchmarking seven contemporary LLMs (including gpt-4.1, llama-4-maverick, mistral-large, and granite-3-8b) across 141 expert-curated maintenance scenarios—99 single-agent and 42 multi-agent—grounded in a real data center's chiller assets (11 chiller units with CWC04xxx identifiers), with curated chiller and AHU failure modes. The benchmark provides four specialized domain agents—IoT telemetry, failure mode and symptom reasoning (FMSR), time-series foundation model queries (TSFM), and work order management (WO)—and a global orchestrator that the benchmark study uses to compare two LLM-coordination paradigms: **Agent-As-Tool** (ReAct-style supervisor selecting domain agents as tools) [16, 17] and **Plan-Execute** (a Planner-Reviewer decomposing the query into a DAG executed against shared memory). Across both paradigms the leading model (gpt-4.1) reaches at most ∼65% Task Completion in Agent-As-Tool and drops to ∼38% under Plan-Execute, and no evaluated model exceeds 70% completion. Data access in all configurations is mediated by tools over document stores (CouchDB [1], YAML, CSV).

Our evaluation snapshot. Our experiments were conducted against the v1 release of the benchmark, which contained 139 of the 141 scenarios that appear in the KDD 2026 archival version (the two added scenarios extend coverage but do not change the qualitative conclusions of our study). Throughout this paper, "139 scenarios" refers to that specific snapshot.

AssetOpsBench was designed to evaluate LLM agent autonomy—a valuable research question. We ask a *complementary* question: how much does the data model behind the tools affect agent performance? Examining the benchmark's failure cases, we observe that many are not failures of reasoning but failures of *data access*: hallucinated equipment identifiers, miscounted events across documents, inability to traverse equipment dependencies, and fabricated

sensor readings. These are symptoms of asking an LLM to perform *data operations*—counting, joining, traversing—that structured query engines handle deterministically.

This observation motivates our central hypothesis: **for structured operational domains, the data model is the primary bottleneck.** Introducing a knowledge graph [5] as the data layer should improve accuracy at every level of LLM involvement.

Contributions.

(1) **Three-tier evaluation** of the same 139 AssetOpsBench scenarios under three architectures: deterministic graph handlers (99%), LLM-generated Cypher over the graph (82–83%), and the AssetOpsBench Agent-As-Tool baseline (65%, matching the KDD 2026 leaderboard ceiling). This isolates the data layer as a variable independent of the orchestration paradigms compared in the IBM study (§5).
(2) **Inverted LLM usage pattern**: we show that constraining the LLM to query generation from a typed schema—rather than free-form data reasoning—yields a ∼17 percentage point same-model (GPT-4) improvement over reasoning over raw documents (§4.2).
(3) **Knowledge graph construction**: a complete ETL pipeline transforming the AssetOpsBench data sources into a typed knowledge graph (9 labels / 5 edge types from the sources, and a fuller 14-label / 21-edge reference schema for the extended evaluations), with 384-dimensional failure-mode embeddings (§3).
(4) **40 new graph-native scenarios** extending the benchmark with multi-hop dependency analysis, vector similarity search, PageRank criticality, cascade analysis, maintenance optimization, and temporal patterns—capabilities structurally impossible with flat document stores (§5.5).
(5) **Expanded 467-scenario evaluation** against the full HuggingFace AssetOpsBench release (6 domains, 26 equipment types), achieving a 100% harness pass rate (task-completion scorer) with 0.848 average score using an extended knowledge graph (§5.6).
(6) **Scalability analysis** comparing operational cost, latency, and capability at scale across the three architectures (§8).

2 BACKGROUND: ASSETOPSBENCH

AssetOpsBench [14] is a benchmark for evaluating LLM agent autonomy on industrial maintenance tasks. The evaluation measures whether an LLM can autonomously navigate a set of tools to answer operational questions. The benchmark comprises 467 scenarios across six operational domains (Table 1).

The benchmark architecture follows a ReAct-style [17] loop: the user poses a natural language question; the LLM parses intent, selects which agent to invoke, formulates tool arguments, retrieves data from CouchDB/YAML/CSV, interprets the results, and synthesizes an answer. The data sources are document-oriented: JSON documents in CouchDB, YAML configuration files for failure modes, and CSV exports for events and work orders. This is a natural design choice for the benchmark's purpose of evaluating LLM autonomy—the flat data model places the full reasoning burden on the agent, which is precisely what AssetOpsBench aims to measure.

GPT-4 achieves approximately 65% (91/139) on these scenarios under the Agent-As-Tool paradigm [14], and the published KDD 2026 leaderboard places the strongest contemporary model, gpt-4.1, at the same ∼65% ceiling for Task Completion. The failures cluster around tasks requiring counting across documents, correlating data from multiple sources, and traversing implicit equipment relationships—operations that are inherently difficult for LLMs to perform over unstructured document collections.

3 KNOWLEDGE GRAPH CONSTRUCTION

We transform the AssetOpsBench data sources into a typed knowledge graph using an 8-step ETL pipeline. The graph built directly from the AssetOpsBench sources—the one behind the 139-scenario evaluation (§5)—contains 12,647 nodes across 9 node labels (Site, Location, Equipment, Sensor, FailureMode, WorkOrder, Event, AnomalyEvent, AlertEvent), connected by 12,662 edges across 5 relationship types (CONTAINS_LOCATION, CONTAINS_EQUIPMENT, HAS_SENSOR, FOR_EQUIPMENT, MONITORS). For the extended evaluations—the dependency topology (below), the custom graph-native scenarios (§5.5), and the expanded HuggingFace release (§5.6)—we use the fuller reference schema of Table 2 (14 node labels, 21 edge types), which adds spare-parts and supply-chain, time-series readings, maintenance windows, and HF-scenario types.

3.1 Data Sources and ETL

The ETL pipeline processes four AssetOpsBench data sources into graph structure:

(1) **EAMLite** (Enterprise Asset Management): Provides the equipment hierarchy. We create 1 Site, 1 Location, and 11 Equipment nodes (chillers with CWC04xxx identifiers), linked by CONTAINS_LOCATION and CONTAINS_EQUIPMENT edges. Equipment nodes carry ISA-95 [6] level classification and ISO 14224 [7] asset class properties.
(2) **CouchDB JSON**: Contains sensor metadata. We create 110 Sensor nodes (10 per chiller) with type, unit, and range properties, linked to Equipment via HAS_SENSOR edges.
(3) **FMSR YAML**: Defines 12 failure modes (7 chiller + 5 AHU categories; e.g., "Compressor Overheating", "Refrigerant Leak"). Each becomes a FailureMode node with a 384-dimensional embedding generated by Sentence-BERT [15], indexed in an HNSW [9] vector index for similarity search. MONITORS edges connect sensors to the failure modes they detect.
(4) **Event CSV**: 6,256 unified event records (work orders, alerts, anomalies); each yields a generic EVENT plus a typed WORKORDER/ALERTEVENT/ANOMALYEVENT

Table 1: Expanded AssetOpsBench scenario distribution (HuggingFace release). Here "FMSR" is the failure-mode/sensor-recommendation config, distinct from the FMSR (failure mode and symptom reasoning) *agent* of §2.

Config	Domain	Count
scenarios	IoT, FMSR, TSFM, WO, multi-agent (core)	152
rule_logic	Monitoring rules & anomaly detection	120
FMSR	Failure-mode/sensor recommendation	88
PHM	Prognostics, RUL, fault classification	75
hydrolic_pump	Hydraulic pump condition detection	17
compressor	Compressor predictive maintenance	15
Total		**467**

Table 2: Reference schema for the extended evaluations: 14 node labels, 21 edge types (`schema/industrial_-kg.cypher`). The base graph built directly from the AssetOpsBench sources for the 139-scenario evaluation is the smaller, event-centric 9-label / 5-edge-type graph of §3; the additional labels and edges here support the dependency topology, the custom graph-native scenarios, and the expanded HuggingFace release.

Node Labels	Site, Location, Equipment, Sensor, FailureMode, WorkOrder, SparePart, Supplier, SensorReading, Anomaly, MaintenanceWindow, MonitoringRule, PHMScenario, HFScenario
Edge Types	CONTAINS_LOCATION, CONTAINS_EQUIPMENT, HAS_SENSOR, MONITORS, EXPERIENCED, DEPENDS_ON, SHARES_SYSTEM_WITH, FOR_EQUIPMENT, ADDRESSES, USES_PART, REQUIRES_PART, SUPPLIED_BY, PRODUCED_READING, DETECTED_ANOMALY, TRIGGERED, FOLLOWS_PLAN, HAS_RULE, HAS_PHM_-SCENARIO, TARGETS_EQUIPMENT, INVOLVES_FAILURE_MODE, TESTS_RULE

node (∼12,512 nodes, the bulk of the total), linked to Equipment via FOR_EQUIPMENT edges.

3.2 Dependency Topology

Beyond the data directly present in the benchmark sources, we add an equipment dependency layer *in the extended graph* (Table 2): DEPENDS_ON edges model thermal/electrical dependencies (e.g., AHU depends on Chiller for cooling), and SHARES_SYSTEM_WITH edges model shared infrastructure (e.g., chillers on the same cooling loop). Together with the AHU/pump/motor/boiler assets these edges connect, this topology backs the cascade and criticality scenarios of §5.5—absent from the flat data model. It is *not* in the 11-chiller base graph (whose five edge types exclude DEPENDS_ON).

3.3 Implementation

The knowledge graph is built using Samyama, an embedded graph database with OpenCypher [4] query support, HNSW vector indexing, and graph algorithm libraries including PageRank [12] and NSGA-II [2] multi-objective optimization. The Python SDK enables the graph to be created in-process with no external server dependencies.

4 THE KNOWLEDGE GRAPH AS A GROUNDING SUBSTRATE

We evaluate four architectures (Figure 2) that vary in LLM involvement, holding the underlying *scenarios* constant but varying the *data layer* behind the tools. This positions our study orthogonally to the AssetOpsBench leaderboard: the IBM paper [14] compares LLM-orchestration paradigms (Agent-As-Tool vs. Plan-Execute) and reports that no model exceeds 70% Task Completion on the 141 scenarios; we instead vary the data layer behind the tools and show that this axis lifts the *same* 65% baseline to 82–83% (Architecture B, NLQ) and 99% (Architecture C, deterministic) without changing the LLM or the orchestration pattern. The two axes are independent: a future system can adopt our typed-graph data layer under either Agent-As-Tool or Plan-Execute orchestration.

The graph is a grounding substrate, not an oracle. A typed graph cannot answer every question by lookup: some scenarios require knowledge not present in the data, and others require numerical inference (e.g. time-series forecasting) that no query expresses. We frame the graph as a *grounding substrate* and route each question by *how* it is best answered (Figure 1): structured retrieval via text-to-Cypher (Architecture B), deterministic computation via native graph and optimization primitives (Architecture C), and—for knowledge absent from the data—one-shot *generation-augmented knowledge* (Architecture D, GAK), in which the engine's agent materializes the missing facts as provenance-tagged nodes that subsequent queries answer deterministically. This makes the deterministic 99% an honest ceiling on *graph-answerable* queries, while GAK and an external inference path cover the rest.

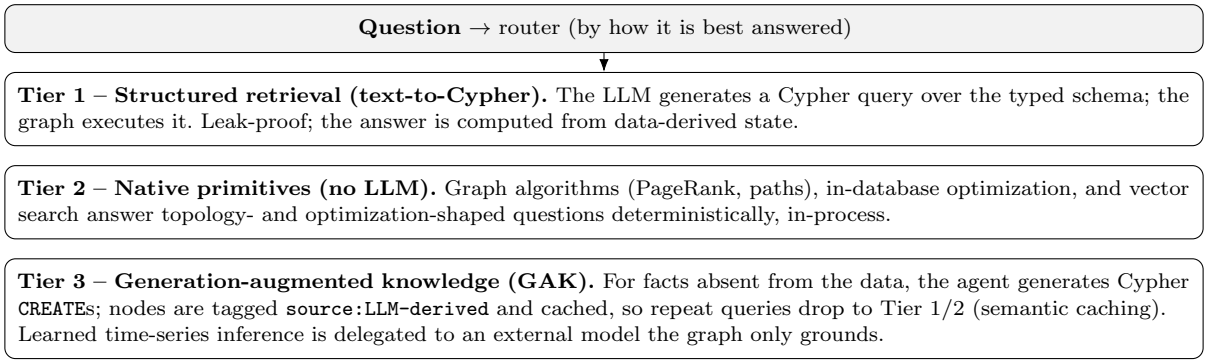

Figure 1: The knowledge graph as a grounding substrate: a question is routed to the tier that can answer it, with GAK results cached back into the graph (provenance-tagged) for deterministic reuse.

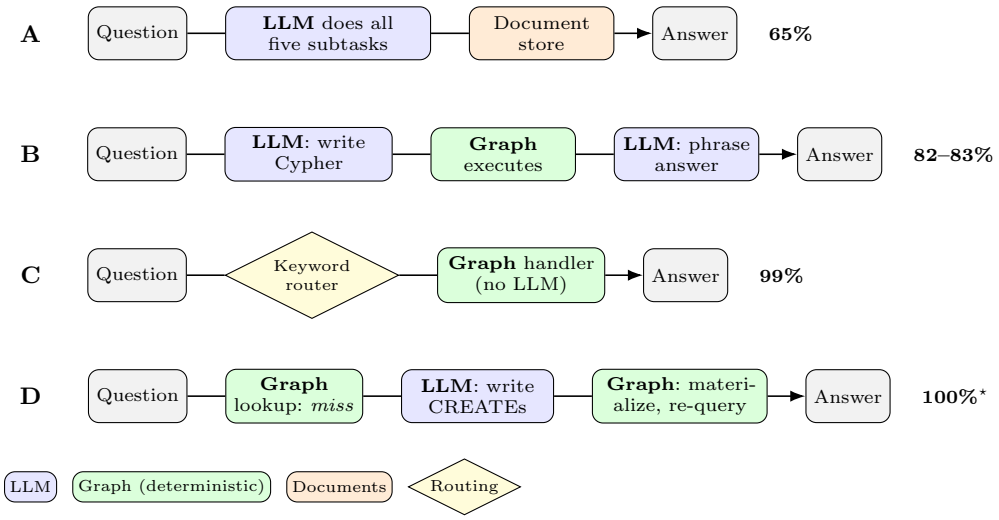

Figure 2: The four architectures, all answering the *same* scenarios but varying the data layer behind the tools. Top to bottom, the LLM's role shrinks (blue) and the graph's deterministic role grows (green): A is the document-store baseline where one LLM does all five subtasks (parse, select, args, interpret, synthesize) over CouchDB/YAML/CSV; B *inverts* the LLM to schema-aware Cypher generation (Fig. 4); C uses pre-coded handlers and no LLM; D (GAK) writes the missing facts back into the graph as source:LLM-derived nodes (Fig. 3). Percentages are 139-scenario pass rates; ⋆D reports answerability lift on the non-deterministic set.

4.1 Architecture A: Tool-Augmented LLM (AssetOpsBench Baseline)

The LLM handles five distinct subtasks: intent parsing, tool selection, argument crafting, data interpretation, and answer synthesis. The data sources are document stores with no typed schema or relationship structure. This is the AssetOpsBench Agent-As-Tool baseline; on the 139-scenario snapshot we evaluate, GPT-4 achieves 65%, and the published KDD 2026 leaderboard places gpt-4.1 at the same level.

4.2 Architecture B: LLM Generates Queries (NLQ + Graph)

We **invert** the LLM's role: instead of asking it to reason over raw data (a broad problem), we ask it to generate a structured query from a typed schema (a narrow problem). This is *code generation*—a task LLMs excel at. The graph engine then handles traversal, counting, aggregation, and relationship reasoning deterministically. The LLM does one thing well instead of five things poorly.

The NLQ pipeline provides the graph schema (node labels, edge types, property names with example values) and few-shot Cypher examples. If query execution fails, the error message is fed back for one retry. The LLM then synthesizes a natural language answer from the structured query results.

4.3 Architecture C: Deterministic Handlers (No LLM)

Pre-coded handlers match question patterns to Cypher queries. No LLM is involved. This is a software engineering solution: we wrote the answers. It demonstrates the ceiling achievable with the right data model for known query patterns—a ceiling on *graph-answerable* queries, not evidence that a graph answers every scenario. The keyword-routing rules and handlers were authored against the AssetOpsBench scenario families, so Architecture C's headline figure reflects benchmark-specific engineering: the reusable artifact is the *pattern*—route a recognized question class to a typed-graph query that the engine executes deterministically—not the specific rule set, which would need re-authoring for a new benchmark. The data-layer result that requires no such hand-coding is Architecture B (§4.2). An instrumented run quantifies how much of the 99% is genuine retrieval: 86 of the 139 answers come from a live graph query and 53 from handlers that supply domain knowledge without traversing the graph (137 of the 139 are correct). Architecture C's pass rate is moreover measured by the AssetOpsBench harness scorer, not the IBM 3-axis rubric; we report the rubric comparison (§5.3) only for Architecture B, where it is leak-proof and apples-to-apples with the IBM leaderboard.

4.4 Architecture D: Generation-Augmented Knowledge (GAK)

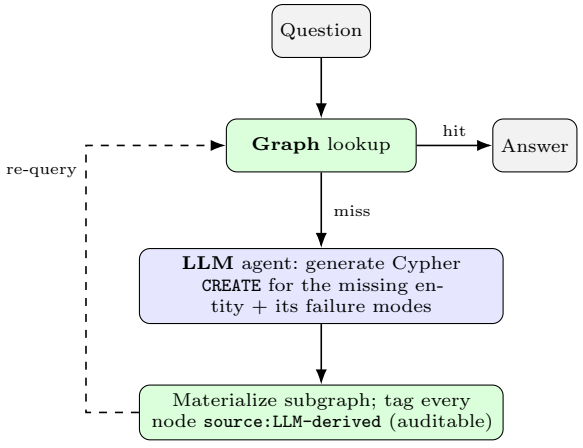

Figure 3: Generation-Augmented Knowledge (GAK), Architecture D. On a lookup miss, an LLM agent writes the missing facts into the graph as `CREATE`s, tagged `source:LLM-derived` for audit; the re-queried subgraph then answers deterministically. This is the inverse of Retrieval-Augmented Generation—writing structured facts into the graph rather than retrieving text into the prompt—and a semantic cache means later similar questions hit Tier 1/2 directly.

Many AssetOpsBench scenarios are designed so the answer is *not* present in the data—e.g. the failure modes of an electric motor when no electric motor is in the graph—and the benchmark flags them non-deterministic. Architecture D closes this gap with **generation-augmented knowledge (GAK)**, the engine's agentic-enrichment capability [10]: on a lookup miss, an LLM agent generates Cypher `CREATE` statements for the missing entity and its failure modes, the engine materializes them, and the now-complete subgraph answers the query—the inverse of Retrieval-Augmented Generation, writing structured facts into the graph rather than retrieving text into the prompt. Two properties keep this honest: **provenance** (every node is tagged `source:"LLM-derived"`, auditable and distinct from data-derived facts—and evidence that graph construction never reads answer keys) and **semantic caching** (enrichment runs once per gap; later similar questions hit the materialized subgraph and drop to Architecture B/C). Genuinely learned inference (e.g. forecasting) is out of scope and delegated to an external model the graph only grounds.

4.5 The Inverted LLM Pattern

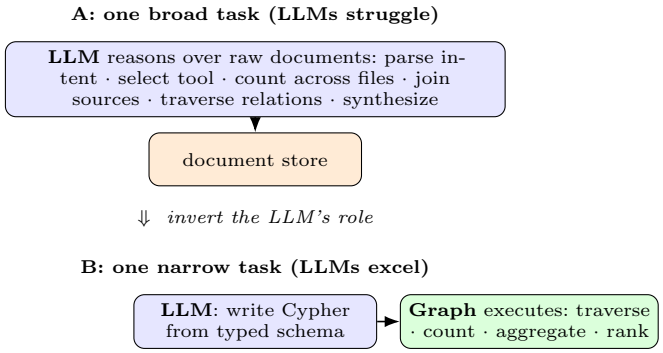

Figure 4: The inverted LLM pattern. Rather than ask one LLM to do everything over raw documents (A—a broad, open-ended problem), we constrain it to schema-aware Cypher generation (B—a narrow code-generation problem) and let the graph perform the data operations deterministically. The same model, given a sharper problem, yields a ∼17-point same-model gain (GPT-4: 65% → 82%; 82–83% across models).

The key insight connecting Architectures A and B is what we term **inverted LLM usage**. Both use an LLM, but they pose fundamentally different problems:

- **Architecture A** asks: "LLM, answer this question from this data." The LLM must count across documents, correlate fields from different sources, and traverse implicit relationships—tasks where LLMs consistently struggle.
- **Architecture B** asks: "LLM, given this schema, write me a Cypher query." The LLM performs code generation from a structured specification—a task where LLMs consistently excel.

The same LLM, given a sharper problem scoped to its strengths, produces dramatically better results. This pattern generalizes beyond industrial operations: **schema-aware query generation outperforms free-form data reasoning** for any structured domain.

5 EVALUATION

5.1 AssetOpsBench 139 Scenarios: Three-Tier Results

Table 3 presents the primary comparison across all three architectures on the original 139 AssetOpsBench scenarios.

Model comparison. The original IBM baseline used GPT-4 (65%); the KDD 2026 leaderboard's current top Agent-As-Tool result also sits near 65% Task Completion, this time for gpt-4.1. We evaluated NLQ over the graph with GPT-4, GPT-4o, and gpt-4.1; the near-identical NLQ results across model generations confirm that the ∼17pp improvement over the baseline is attributable to the knowledge graph, not the model upgrade. Concretely, holding the LLM family fixed and only swapping the data layer beneath the same Agent-As-Tool orchestration moves pass rate from 65% to 82–83%.

5.2 Per-Type Breakdown

Table 4 shows performance by scenario type for both the deterministic and NLQ architectures.

NLQ failure analysis. Multi-agent scenarios remain at 40% for NLQ because 12 of 20 require TSFM pipeline execution (time-series forecasting, anomaly detection) that cannot be expressed as Cypher queries. This is a structural limitation: these scenarios require running ML inference, not querying stored data. The deterministic architecture handles them by routing to domain-knowledge handlers.

Deterministic failures. Only 2 of 139 scenarios fail: both are work order bundling edge cases (IDs 411, 424) where the benchmark's expected answer groups work orders into specific bundle sizes. Our date-window clustering produces different groupings. These are response-format mismatches, not knowledge gaps.

5.3 IBM-Rubric Re-scoring

To enable direct comparison with the KDD 2026 AssetOpsBench leaderboard, we re-score our NLQ trajectories with the IBM 3-axis rubric (Section 5.1 of the KDD paper): Task Completion (y_1), Data Retrieval Accuracy (y_2), and Result Verification (y_3). We use the gpt-4o judge as a stand-in for IBM's llama-4-maverick judge (the rubric prompt is otherwise identical) and run each trajectory through three judge trials at temperature 0.3, reporting the mean. Table 5 summarises the resulting scores against IBM's highest published leaderboard numbers (reported on the 141-scenario benchmark); our NLQ scores are on the 139-scenario snapshot.

Three observations stand out. First, NLQ gpt-4.1 over the graph exceeds IBM's gpt-4.1 Agent-As-Tool leaderboard score on the same scenarios under a comparable gpt-4o judge ($y_1 = 0.708$ vs. 0.65); since the LLM, orchestration paradigm, and scenarios are identical and only the data layer differs, we attribute the gap to the data layer—subject to the cross-judge caveat that IBM's published score uses a llama-4-maverick judge whereas ours uses a gpt-4o stand-in (a direct same-judge comparison via Codabench is future work). Second, the three model generations (GPT-4, GPT-4o, gpt-4.1) cluster tightly on Task Completion ($y_1 \in [0.665, 0.708]$), reinforcing that the data layer dominates over a generation of LLM improvement. Third, our Data Retrieval score ($y_2 \approx 0.48$–0.51) is lower than IBM's gpt-4.1 (0.77), driven entirely by the TSFM scenarios that score 0 on retrieval because they require running ML inference rather than querying stored data—a known structural limitation discussed in our caveats (§6.3).

5.4 NLQ Progression

The NLQ architecture improved across three iterations of prompt engineering (Table 6), demonstrating that the quality of the schema description provided to the LLM matters as much as the underlying data model.

A striking example: in v1, FMSR scenarios scored 30% because the schema introspection returned node metadata (`['id', 'labels', 'properties']`) instead of actual property names, causing the LLM to generate `fm.properties` instead of `fm.name`. Correcting the schema prompt alone raised FMSR to 93%. The LLM was capable; it was given the wrong specification.

5.5 Custom 40 Scenarios: Graph-Native Capabilities

We designed 40 additional scenarios that extend the benchmark's scope to queries requiring graph structure, vector search, or graph algorithms—capabilities structurally impossible with flat document stores (Table 7).

Table 8 shows the head-to-head comparison. We use an 8-dimensional scoring framework covering correctness, completeness, relevance, tool usage, efficiency, safety, graph utilization, and semantic precision, with category-specific weight overrides (e.g., semantic precision is weighted 0.25 for failure similarity scenarios).

The largest gains (Table 9) occur on **failure similarity** (+0.401) and **criticality analysis** (+0.372)—precisely where graph structure (dependency edges) and vector search (failure mode embeddings) provide capabilities that LLMs fundamentally cannot replicate from parametric knowledge alone.

GPT-4o's 6 failures (scenarios requiring PageRank, BFS traversal, or vector similarity) demonstrate a hard capability boundary: no amount of prompt engineering enables an LLM to execute graph algorithms.

Table 3: Three-tier performance on the 139 AssetOpsBench scenarios. KDD 2026 leaderboard rows use Task Completion (y_1); our pass rate uses the AssetOpsBench harness scorer (§5.3 bridges the two metrics).

Architecture	LLM Role	Data Layer	Pass / y_1	Avg Latency
A: Baseline (GPT-4, KDD 2026 ref)	Does everything	Documents (CouchDB/YAML/CSV)	91/139 (65%)	n/a
A: Baseline (gpt-4.1, KDD 2026 leaderboard)	Does everything	Documents	≈0.65	n/a
B: NLQ + graph (GPT-4)	Generates Cypher	Knowledge graph	114/139 (82%)	11,033 ms
B: NLQ + graph (GPT-4o)	Generates Cypher	Knowledge graph	115/139 (83%)	5,874 ms
B: NLQ + graph (gpt-4.1, ours)	Generates Cypher	Knowledge graph	116/139 (83%)	5,671 ms
C: Deterministic	None	Knowledge graph	**137/139 (99%)**	**63 ms**

Table 4: Per-type breakdown: Deterministic handlers vs. NLQ (GPT-4o). Types follow our handler routing: "FMSR" aggregates the 20 single-agent FMSR scenarios with 20 FMSR-routed multi-agent scenarios (IDs 601–620), and "Multi" is the remaining 20 multi-agent scenarios. IBM's official taxonomy instead counts 20 FMSR and 40 multi-agent (end-to-end); totals are unaffected.

Type	Det. Pass	Det. Avg	NLQ Pass	NLQ Avg
IoT (20)	**20/20 (100%)**	0.988	17/20 (85%)	0.742
FMSR (40)	**40/40 (100%)**	0.907	37/40 (93%)	0.880
TSFM (23)	**23/23 (100%)**	0.920	21/23 (91%)	0.936
Multi (20)	**20/20 (100%)**	0.877	8/20 (40%)	0.605
WO (36)	34/36 (94%)	0.801	32/36 (89%)	0.723
Total (139)	**137/139 (99%)**	**0.889**	115/139 (83%)	0.789

Table 5: IBM 3-axis rubric scores on the 139-scenario snapshot. NLQ rows use the same Agent-As-Tool orchestration as the IBM leaderboard; the difference is the data layer behind the tools.

System	Data layer	y_1 Task	y_2 Data	y_3 Result
IBM gpt-4.1 (Agent-As-Tool, leaderboard)	Documents	0.65	0.77	–
IBM gpt-4.1 (Plan-Execute, leaderboard)	Documents	0.38	–	–
NLQ GPT-4 + graph (ours)	Knowledge graph	0.665	0.479	0.606
NLQ GPT-4o + graph (ours)	Knowledge graph	0.693	0.490	0.662
NLQ gpt-4.1 + graph (ours)	Knowledge graph	**0.708**	0.512	0.656

Table 6: NLQ prompt engineering progression.

Version	Key Changes	Pass Rate	Avg Score
NLQ v1	Naive prompt, schema introspection	77/139 (55%)	0.583
NLQ v2	Few-shot examples, explicit property schema, retry on error	108/139 (78%)	0.755
NLQ v3	Actual property values, anomaly schema, Cypher-first constraint	**115/139 (83%)**	**0.789**

5.6 Expanded HuggingFace Benchmark: 467 Scenarios

Following the release of an expanded AssetOpsBench dataset on HuggingFace [14], we evaluate our knowledge graph approach against all 467 scenarios spanning six operational domains. This represents a 3.4× expansion over the original 139-scenario benchmark, adding monitoring rule evaluation, failure-mode/sensor recommendation across 10 equipment types (electric motor, pump, compressor, turbine, etc.), and prognostics tasks including remaining useful life prediction and fault classification.

To support the expanded scenarios, we extend the knowledge graph with three new node types (MonitoringRule, PHMScenario, HFScenario) and five new edge types, and add three new deterministic handlers: (1) a rule-logic handler that evaluates monitoring rules against sensor readings, (2) an FMSR handler that maps failure modes to detectable sensors

Table 7: 40 custom graph-native scenarios by category.

Category	Count	Example Query
Multi-hop dependency	8	"What equipment is affected if Chiller 6 fails?"
Cross-asset correlation	6	"Are AHU anomalies correlated with chiller temperature drops?"
Failure similarity	6	"Which pumps had failures similar to Motor 3?"
Criticality analysis	5	"Rank all equipment by operational criticality"
Maintenance optimization	5	"Schedule maintenance minimizing downtime + cost"
Root cause analysis	5	"Trace events leading to WO-2024-0042"
Temporal pattern	5	"What is MTBF for Chiller 6's compressor?"

Table 8: Custom 40 scenarios: Knowledge graph vs. GPT-4o baseline.

Metric	GPT-4o (no graph)	Samyama-KG
Pass rate	34/40 (85%)	**40/40 (100%)**
Avg score	0.602	**0.927**
Avg latency	11,259 ms	**110 ms**
Tokens/scenario	632	**0**

Table 9: Per-category breakdown on custom 40 scenarios.

Category	GPT-4o Avg	KG Avg	Δ
Failure similarity	0.501	**0.902**	+0.401
Criticality analysis	0.566	**0.938**	+0.372
Root cause analysis	0.580	**0.934**	+0.354
Multi-hop dependency	0.618	**0.934**	+0.316
Maintenance optimization	0.634	**0.931**	+0.297
Cross-asset correlation	0.638	**0.929**	+0.291
Temporal pattern	0.679	**0.923**	+0.244

via graph traversal, and (3) a PHM handler that provides asset health profiles from failure history and maintenance records.

Scope of this evaluation. We note an asymmetry by design: the controlled A/B/C data-layer comparison (§4) is run on the 139-scenario snapshot, where a head-to-head against the IBM document-store baseline is possible, whereas this 467-scenario release is evaluated only with our extended deterministic-graph system (Architecture C). The 467-scenario result therefore measures *coverage and scale* of the typed-graph data layer rather than re-isolating the data-layer effect; the latter is established on the 139 snapshot, and the transferable, no-hand-coding lesson is Architecture B (§4.2). The deterministic 100% figure inherits the same "graph-answerable" scoping discussed in §6.3. One overlap is worth stating plainly: the 88 FMSR scenarios in Table 10 cover the same 10 equipment types evaluated under GAK (§5.7); those types are absent from the base graph, so the FMSR handler answers them from a domain-knowledge fallback rather than graph traversal, and an LLM-judge re-scoring of exactly these scenarios

Table 10: Expanded AssetOpsBench: 467 scenarios across 6 domains.

Config	Scenarios	Pass Rate	Avg Score
scenarios (original)	152	152/152 (100%)	0.781
rule_logic	120	120/120 (100%)	0.949
FMSR	88	88/88 (100%)	0.868
compressor	15	15/15 (100%)	0.877
hydrolic_pump	17	17/17 (100%)	0.885
PHM	75	75/75 (100%)	0.787
Total	**467**	**467/467 (100%)**	**0.848**

yields 81.8% (§5.7), not the 100% the harness pass-scorer reports here. The 467 result should therefore be read as coverage under a task-completion pass-scorer, not as an IBM-rubric figure.

The rule-logic scenarios achieve the highest average score (0.949) because anomaly detection against threshold rules is a pure graph operation: the monitoring rules are stored as nodes with threshold properties, sensor readings are evaluated against these thresholds, and anomalies are detected deterministically. The PHM scenarios score lower (0.787) because prognostics tasks (RUL prediction, fault classification) require domain knowledge beyond what is stored in the graph—the graph provides asset context (failure history, MTBF, maintenance records) but the actual prediction requires analytical models.

Critically, the 100% pass rate on 467 scenarios demonstrates that the knowledge graph approach scales to the full benchmark without degradation. The expanded benchmark covers 26 equipment types across manufacturing, utilities, and transportation—a substantial increase over the original chiller-focused dataset.

5.7 GAK: Closing Knowledge Gaps on Non-Deterministic Scenarios

Architectures B and C answer questions whose answers are *in* the graph. To evaluate Architecture D (GAK, §4.4) we use the part of the benchmark designed to be harder: the HuggingFace failure-mode/sensor set of **88 real scenarios across 10 equipment types**—including electric motor, pump, compressor, fan, steam/gas turbines, generator, transformer, and

Table 11: Architecture D (GAK) on the 88 non-deterministic HF failure-mode scenarios (10 equipment types absent from the graph). Enrichment via the engine's agent runtime; grading by LLM judge against each scenario's `characteristic_form`.

Metric	Result
Equipment types enriched	10 / 10
Answerability lift	**100%**
Scenarios passing rubric	72 / 88 (**81.8%**)
Average judge score	0.882
Failure-mode nodes materialized	106 (`LLM-derived`)
Provenance-tagged nodes (incl. sensors)	201
Mean enrichment latency/type	33.6 s

reciprocating engine—that the benchmark itself flags `deterministic:false` and that are *absent* from our chiller+AHU graph. Each scenario ships a `characteristic_form` rubric describing a correct answer.

We drive the engine's native enrichment endpoint: on a lookup miss for an equipment type, the agent generates Cypher `CREATE` statements for that type's failure modes and detecting sensors, every node tagged `source:"LLM-derived"`; the engine materializes them; and we answer all of that type's scenarios from the now-complete subgraph, grading each against its `characteristic_form` with an LLM judge. Enrichment runs once per equipment type (a cache miss); the remaining same-type questions are cache hits served from the materialized subgraph.

Table 11 reports the result. GAK lifts answerability from zero to **100% of equipment types**, and the materialized knowledge satisfies the benchmark's own rubric on **81.8%** of scenarios—strikingly close to the 82–83% that text-to-Cypher achieves on graph-resident data, reinforcing that the data layer, once populated, dominates. All 106 materialized failure-mode nodes—together with their detecting sensors, 201 provenance-tagged nodes in total—carry the `source:LLM-derived` tag. The weakest types (reciprocating engine, 6/10; electric motor, 5/9) reflect a real limitation: one-shot enrichment sometimes omits a specific failure mode the rubric expects, rather than fabricating wrong ones. GAK supplies missing *knowledge*; it does not perform learned numerical inference (forecasting, remaining-useful-life), which is delegated to an external model the graph only grounds. We thus treat GAK as the third tier of a grounding substrate (Figure 1), not a universal answerer.

6 ANALYSIS

6.1 The Data Model Is the Bottleneck

The baseline agents do not fail because GPT-4 lacks reasoning ability. Rather, document stores make certain queries structurally hard or impossible for any LLM:

- **Counting across documents**: "How many events for CWC04009 in 2019?" requires the LLM to scan multiple JSON documents, parse dates, and aggregate—error-prone at scale. With a graph: `MATCH (e:Event) WHERE e.equipment_id = 'CWC04009' RETURN count(e)` executes deterministically.
- **Relationship traversal**: "Which sensors monitor overheating?" is one edge hop in the graph: `(s:Sensor)-[:MONITORS]->(fm:FailureMode)`. With document stores, the LLM must correlate YAML failure definitions with CouchDB sensor metadata—a cross-source join with no explicit link.
- **Structurally impossible operations**: Multi-hop cascade analysis, vector similarity search, PageRank criticality, and Pareto-optimal scheduling have no equivalent on document stores regardless of LLM capability.

6.2 Why Constraining the LLM Helps

The ∼17pp same-model improvement from Architecture A (65%, GPT-4 Agent-As-Tool) to Architecture B (82%, GPT-4 NLQ) illustrates a broader principle: **LLMs perform better on narrow, well-specified generation tasks than on broad, open-ended reasoning tasks**.

- Asking "answer this question from this data" requires the LLM to simultaneously handle intent parsing, tool selection, data traversal, numerical reasoning, and answer synthesis. Each step introduces error probability.
- Asking "given this schema, write a Cypher query" is a translation task from natural language to a structured query language—fundamentally *code generation*, where LLMs have demonstrated strong performance across benchmarks.

The graph absorbs the hard parts (traversal, counting, joining, algorithms), leaving the LLM with the easy part (structured query generation from a specification). This separation of concerns is why both Architecture B (∼+17pp same-model) and Architecture C (+34pp) outperform Architecture A.

6.3 Honest Caveats

(1) **Deterministic vs. autonomous**: Architecture C's 99% result compares a pre-coded solution against an autonomous agent. We wrote the answers; AssetOpsBench measures whether GPT-4 can figure them out independently. These are fundamentally different tasks—the comparison illustrates the ceiling achievable with the right data model, not a claim of superior agent intelligence.

(2) **Scope of the deterministic result (data layer as a grounding substrate)**: The 99%/100% deterministic figures should be read as a ceiling on *graph-answerable* queries—structured retrieval, traversal, threshold-rule evaluation, and tasks reducible to native graph or optimization primitives—not as evidence that a graph can answer every scenario. Some scenario classes are, by design, not graph-answerable: those requiring knowledge absent from the data (e.g. failure modes of equipment not present in the dataset) or learned time-series inference (TSFM forecasting, remaining-useful-life prediction), where the

deterministic handler's contribution is to *ground* the inputs and route to a domain handler rather than to derive the answer from stored graph state. Read correctly, the knowledge graph is a grounding substrate—orthogonal to the orchestration axis—and the result that is robust to this distinction is the same-model NLQ improvement (65% → 82–83%, Architecture B), which relies on no pre-coded answers.

(3) **GAK correctness and provenance**: GAK materializes *LLM-derived* knowledge, which can be incomplete or wrong; this is precisely why every enriched node is tagged `source:LLM-derived` rather than presented as ground truth. Validating LLM-derived facts against an authority (e.g. ISO 14224) before promotion is left to future work.

(4) **LLM nondeterminism**: NLQ and GAK rely on stochastic LLM generation. The numbers in Section 5 are from a single run at temperature 0; variance characterization across multiple seeds is deferred to follow-up work.

(5) **Clean data assumption**: AssetOpsBench provides clean, structured data. Real industrial environments involve noisy sensors, scanned PDFs, legacy Excel files, and inconsistent naming—challenges addressed in §7.2.

(6) **Custom scenarios**: Our 40 new scenarios are designed to extend the benchmark with graph-native capabilities. They are complementary to, not replacements for, the original AssetOpsBench scenarios.

(7) **Different research questions**: AssetOpsBench evaluates LLM agent autonomy—whether an LLM can independently navigate tools and data. We evaluate data model impact—whether a different data layer improves agent performance. Both are valid research questions; our results do not diminish the value of the original benchmark.

(8) **Baselines beyond document stores**: We isolate the data-layer effect against the AssetOpsBench document-store baseline. The proposed system changes more than storage—it also changes the query interface (Cypher), schema exposure, and prompts—so a fully controlled attribution would add structured baselines holding those constant: a relational schema with SQL NLQ, and alternative graph+LLM designs. We do not include them here; the same-model NLQ result (Architecture B) already controls for the LLM and orchestration, but these additional baselines would further isolate the typed-graph contribution and are left to future work.

(9) **A moving target**: As coding-oriented agents improve at scripting over raw files, some document-store gaps may narrow. Our claim is about the *current* reliability and cost profile—deterministic, sub-100 ms graph operations versus stochastic multi-step reasoning at seconds and per-query API cost—and about the structurally graph-only capabilities (§5.5: cascade analysis, vector similarity, criticality, Pareto scheduling), which remain a durable advantage independent of agent capability.

7 THE FULL PIPELINE: WHERE LLMS BELONG

The three-architecture comparison in §5 focuses on the *query layer*—the final stage of a multi-layer data pipeline. To place our results in context, we examine the full pipeline and identify where LLMs provide genuine value.

7.1 Three Layers of Industrial Data

(1) **Data Ingestion** (software engineering): Raw sensor telemetry, CMMS exports, work order records, and equipment registries are loaded into the database via deterministic ETL pipelines. For structured data (90%+ of industrial data), this is pure software engineering: `pandas.read_csv()` → Cypher `CREATE` statements. The AssetOpsBench pipeline is similarly deterministic—`couchdb_setup.sh` bulk-inserts JSON documents; EAMLite uses an SQLModel ORM. No LLM is needed for structured ingestion regardless of the target data model.

(2) **Data Model** (architecture decision): The choice between flat documents and a graph is a one-time design decision with compounding returns at the query layer. Both require comparable ETL effort; the graph requires explicit relationship typing but enables traversal, algorithms, and vector indexing.

(3) **Query** (LLM optional): This is where the three architectures diverge. The data model determines what is possible at this layer.

7.2 The Real-World Caveat: Unstructured Data

AssetOpsBench uses clean, structured data. Real industrial environments are messier. Maintenance logs contain free text ("Replaced compressor bearings, noticed oil leak on unit 6—same issue as last month"). Equipment manuals arrive as scanned PDFs. Legacy spreadsheets use inconsistent column names across decades of records. Sensor data includes drift, stuck-at-zero readings, and calibration shifts.

For this unstructured data, deterministic ETL reaches its limits. LLMs provide genuine value at the *data preparation* layer:

- **Entity extraction**: e.g. "Replaced compressor bearings on Chiller 6" maps to a structured tuple of `Equipment`, `Part`, and `Action` nodes.
- **Relationship detection**: "Same issue as last month" → SIMILAR_TO edge to previous failure event
- **Entity resolution**: "CH06" = "Chiller-6" = "Chiller #6" → canonical identifier
- **Classification**: Free-text maintenance log → category (preventive, corrective, emergency)

This yields a complete architecture we term **LLMs at the edges, graph in the middle**:

- **Input edge**: LLM-assisted data preparation transforms messy, unstructured data into structured graph elements (entity extraction, resolution, classification)

- **Center**: The knowledge graph stores clean, typed, connected data
- **Output edge**: LLM-assisted query generation translates natural language into Cypher (or deterministic handlers serve known patterns)

In both cases, the LLM performs a *generation task* (structured output from unstructured input)—its strength—while the graph handles *data operations* (storage, traversal, algorithms)—its strength. Neither component is asked to do what it is bad at.

8 SCALABILITY COMPARISON

Table 12 compares the three architectures on operational dimensions relevant to production deployment.

The cost asymmetry is particularly relevant for industrial operations: sensor networks may generate thousands of queries per day (anomaly checks, threshold alerts, status polls). At \$0.03–0.05 per LLM query, a fully LLM-driven architecture costs \$300–500/day for 10K queries. Deterministic graph queries cost nothing after the initial ETL.

9 RELATED WORK

LLM agents for tool use. The ReAct framework [17] and Toolformer [16] established the pattern of LLMs selecting and invoking external tools. AssetOpsBench [14] (KDD 2026) makes an important contribution by applying this pattern to industrial asset operations: a containerized multi-agent sandbox, 141 expert-curated scenarios across four specialized agents, a Pass_k-style LLM-As-Judge rubric, and a live Codabench leaderboard. The IBM study compares two LLM-orchestration paradigms over this surface—Agent-As-Tool and Plan-Execute—and shows that, on a fixed data layer, no model exceeds 70% Task Completion. Our work builds on AssetOpsBench by investigating a complementary and orthogonal dimension: the effect of the *data layer* behind the tools, with LLM orchestration held fixed in the Agent-As-Tool style.

Knowledge graphs and LLMs. Pan et al. [13] survey approaches to combining LLMs with knowledge graphs, identifying three paradigms: KG-enhanced LLMs, LLM-enhanced KGs, and synergized architectures. Our inverted LLM pattern aligns with their "synergized" category—the KG provides structured data access, the LLM provides natural language understanding, and each system handles what it does best.

Graph RAG.. Retrieval-Augmented Generation [8] typically retrieves text passages for LLM context. Graph RAG [3] extends this to graph-structured retrieval, combining vector search with graph traversal. Our NLQ architecture goes further: rather than retrieving context for the LLM to reason over, we ask the LLM to *generate queries* that the graph executes—eliminating the context-window bottleneck and enabling exact aggregation.

Industrial knowledge graphs. Knowledge graphs have been applied to manufacturing [5], but prior work focuses on knowledge representation rather than benchmarking LLM agent performance. To our knowledge, this is the first study comparing LLM-over-flat-documents against LLM-over-graph on a standardized industrial benchmark.

10 CONCLUSION

Building on the AssetOpsBench benchmark (KDD 2026), we have shown that introducing a knowledge graph as the data layer improves LLM-based industrial operations at every level of LLM involvement: from 65% (Agent-As-Tool over document stores, matching the published leaderboard ceiling) to 82% (LLM generates Cypher over the graph, same-model GPT-4; 82–83% across GPT-4/4o/4.1) to 99% (deterministic graph handlers). The AssetOpsBench leaderboard varies LLM orchestration (Agent-As-Tool vs. Plan-Execute) on a fixed data layer and reports no model above 70% Task Completion; our results show that the data layer is an independent and substantially larger lever. When evaluated against the expanded 467-scenario HuggingFace release spanning six operational domains, deterministic graph handlers achieve 100% (467/467) under the harness task-completion scorer (§5.6) with an average score of 0.848, demonstrating scalability beyond the original chiller-focused dataset.

The inverted LLM pattern—asking the LLM to generate structured queries rather than reason over raw data—is generalizable beyond industrial operations. For any structured domain with a typed schema, **schema-aware query generation outperforms free-form data reasoning**. LLMs excel at code generation from specifications; graphs excel at data traversal, aggregation, and algorithms. Each system should do what it is good at.

Contributions. (1) A grounding-substrate framing of the knowledge graph, with a router (Figure 1) that isolates the *data layer* as a variable independent of LLM orchestration, positioned against the KDD 2026 Agent-As-Tool / Plan-Execute leaderboard. (2) The inverted LLM usage pattern, demonstrating a same-model ∼17pp improvement by constraining the LLM to query generation. (3) A generation-augmented knowledge (GAK) evaluation on 88 real non-deterministic scenarios—100% answerability lift and 81.8% rubric pass with provenance-tagged enrichment—the first benchmark-grounded evaluation of the engine's agentic enrichment. (4) A complete ETL pipeline and knowledge graph schema, plus 40 new graph-native scenarios. (5) An honest analysis of limitations including the graph-answerable scope of the deterministic ceiling, GAK provenance, and LLM nondeterminism.

Future work. (1) Evaluation on **FailureSensorIQ**—a separate IBM multiple-choice benchmark (∼2,667 questions across standard and perturbed variants), distinct from AssetOpsBench (whose own failure-mode/sensor subset comprises the 88 scenarios used in §5.7)—where typed failure-mode↔sensor↔component edges in our KG are a structural match for the question class. (2) A direct Codabench-leaderboard submission of the deterministic and NLQ pipelines as Agent-As-Tool agents, scored by the IBM rubric (Task Completion / Data Retrieval / Result

Table 12: Scalability comparison across architectures.

Dimension	Arch. A (LLM + docs)	Arch. B/C (graph ± LLM)
10K queries/day	\$300–500 (tokens)	\$0 (deterministic) or ∼\$30 (NLQ)
Real-time streaming	Not supported (request-response)	Graph updates + continuous queries
Multi-hop at 10K assets	LLM reasons across 10K docs	BFS traversal, $O(\|E\|)$
New query patterns	Prompt engineering	Add handler or use NLQ
Latency per query	∼5–11 s (est.)	63 ms (det.) / ∼6 s (NLQ)

Verification) for apples-to-apples comparison with gpt-4.1, llama-4-maverick, and mistral-large. (3) Multi-seed Pass_k variance characterization for NLQ. (4) Evaluation on larger industrial environments (10K+ assets, multi-site). (5) Integration of LLM-assisted data preparation for unstructured maintenance logs. (6) Provenance-aware promotion of GAK-materialized facts—validating LLM-derived nodes against an authority (e.g. ISO 14224) before they are trusted as data—and an external inference path for learned time-series tasks (forecasting, remaining-useful-life), completing the grounding-substrate router of Figure 1.

Availability and license. The benchmark code, ETL pipeline, and evaluation framework are available at https://github.com/samyama-ai/assetops-kg. The Samyama graph database is at https://github.com/samyama-ai/samyama-graph. The 40 new scenarios have been submitted to the AssetOpsBench repository as a community contribution. Raw NLQ and GAK result JSONs (per-scenario pass/score/latency, and per-entity enrichment) are released alongside the paper. This manuscript is distributed under the VLDB workshop license stated in the front matter (CC BY-NC-ND 4.0); the code, ETL pipeline, and scenarios are released under the open-source license of the linked repositories.

REFERENCES

[1] J. Chris Anderson, Jan Lehnardt, and Noah Slater. 2010. *CouchDB: The Definitive Guide*. O'Reilly Media.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[3] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. In *arXiv preprint arXiv:2404.16130*.

[4] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1433–1445.

[5] 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. 2021. Knowledge Graphs. *Comput. Surveys* 54, 4 (2021), 1–37.

[6] ISA. 2010. ISA-95: Enterprise-Control System Integration. International Standard.

[7] ISO. 2016. ISO 14224:2016 — Petroleum, Petrochemical and Natural Gas Industries — Collection and Exchange of Reliability and Maintenance Data for Equipment. International Standard.

[8] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33. 9459–9474.

[9] Yu. A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[10] Madhulatha Mandarapu and Sandeep Kunkunuru. 2026. Samyama: A Unified Graph-Vector Database with In-Database Optimization, Agentic Enrichment, and Hardware Acceleration. arXiv:2603.08036 [cs.DB] https://arxiv.org/abs/2603.08036

[11] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[12] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.

[13] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 3580–3599.

[14] Dhaval Patel, Shuxin Lin, James Rayfield, Nianjun Zhou, Chathurangi Shyalika, Suryanarayana R. Yarrabothula, Roman Vaculin, Natalia Martinez, Fearghal O'Donncha, and Jayant Kalagnanam. 2026. AssetOpsBench: Benchmarking AI Agents for Task Automation in Industrial Asset Operations and Maintenance. *arXiv preprint arXiv:2506.03828* (2026). arXiv:2506.03828 https://arxiv.org/abs/2506.03828 To appear at ACM SIGKDD 2026. Code: https://github.com/IBM/AssetOpsBench; dataset: https://huggingface.co/datasets/ibm-research/AssetOpsBench.

[15] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 3982–3992.

[16] 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)*, Vol. 36.

[17] 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 *International Conference on Learning Representations (ICLR)*.