MFG-AgentGraph: SHACL-Validated and PROV-O-Traceable LLM Agents for Manufacturing Anomaly Triage

Faizan Ali[1,*], Ameya Punekar[1] and Juergen Bock[1]

[1]Technische Hochschule Ingolstadt, Esplanade 10, 85049 Ingolstadt, Germany

Abstract

Manufacturing anomaly triage needs more than plausible recommendations. An operator must understand why an action applies to a specific asset, check it against encoded domain rules, trace it back to its evidence, and run the entire pipeline within the factory network. Existing RAG, GraphRAG, and KG-guided LLM systems each cover part of this problem, but they rarely combine enforceable validation, standards-based provenance, and local deployment in a single architecture. We propose MFG-AgentGraph, a locally deployable KG-LLM agent that couples an SSN/SOSA-aligned RDF knowledge graph with hybrid SPARQL and vector retrieval, SHACL validation serving as an active repair gate on the agent's structured output, and PROV-O/PROV-AGENT-style provenance traces. The core claim is architectural: a candidate recommendation must pass a machine-checkable validity gate and carry a structured evidence package before the operator sees it. We specify a four-strategy evaluation protocol (naive RAG, GraphRAG-style retrieval, ontology-grounded RAG, and the full KG-LLM agent) that isolates the incremental value of graph structure, expert ontology grounding, and agentic validation with provenance. This paper is deliberately the framework component of a two-part research program: it specifies the architecture and a falsifiable evaluation protocol, reports a small pilot that demonstrates the SHACL repair-gate in operation, and leaves the full comparative study to a companion implementation study.

Keywords

knowledge graphs, RAG, SHACL, PROV-O, manufacturing anomaly triage, LLM agents

CEUR-WS.org/Vol-4312/paper19.pdf

1. Introduction

At 14:03, a pump station emits an anomalous acoustic signature. An operator asks: "What should I inspect first, and why?" A large language model can produce a fluent checklist, but fluency is not the same as accountability. The operator needs to know which asset the recommendation targets, which observation triggered it, which standard operating procedure supports it, whether the action is structurally valid for that asset, and how the system arrived at the answer.

This gap between raw answer generation and auditable decision support is what we address. Retrieval-augmented generation (RAG) retrieves relevant text but does not guarantee grounding in the correct asset, sensor, or procedure [1]. GraphRAG adds graph structure, yet its graph is typically built by an LLM from source documents, not anchored in an expert manufacturing ontology, and its outputs are not validated against domain constraints or linked to provenance traces [2, 3]. KG-guided LLM methods such as Think-on-Graph and Reasoning on Graphs show that an LLM can explore graph paths, but they do not by themselves enforce SHACL-validated outputs or emit PROV-O audit traces [4, 5].

RAGE-KG 2026: Retrieval-Augmented Generation Enabled by Knowledge Graphs, co-located with ISWC 2026, October 25–26, 2026, Bari, Italy

*Corresponding author.

faizan.ali@thi.de (F. Ali); punekarameya.work@gmail.com (A. Punekar); juergen.bock@thi.de (J. Bock)

0009-0007-2945-2149 (F. Ali)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-04

Manufacturing imposes additional constraints. LLM-based assistants are already being explored for factory knowledge sharing and operator support [6, 7, 8], but sensor data, maintenance records, and SOPs are often sensitive; factory networks may be isolated; and operator-facing systems should not depend on an external cloud service for every recommendation. A practical architecture must therefore be locally deployable and measurable in terms of latency, memory, and per-stage runtime.

MFG-AgentGraph closes this gap with a single guarantee: before an operator sees a recommendation, the system must prove that the recommendation is structurally valid and must attach the evidence it rests on. To do this it grounds every recommendation in an expert manufacturing knowledge graph and then subjects the agent's own output to two machine-checkable gates. The first is SHACL, a W3C language for validating the shape of RDF data [9], used here not as a one-off check but as an active repair gate inside the agent loop. The second is a provenance contract in PROV-O, the W3C vocabulary for recording which activities, inputs, and agents produced a result [10], extended with PROV-AGENT-style agent-workflow traces [11]. The knowledge graph itself is aligned to SSN/SOSA, the standard vocabulary for sensors, observations, and the features they observe [12], so assets, sensors, and anomaly events are typed objects rather than strings. The novelty is not a new foundation model, graph database, or provenance standard, but a coupling none of these lines provides on its own: a deterministic SHACL repair-gate that sits *inside* the agent loop and is bound to a PROV-O evidence contract, applied to manufacturing anomaly triage. We are not aware of an existing architecture that makes a machine-checkable validity gate and a standards-based provenance package *joint preconditions* for showing a recommendation to a factory operator.

The paper makes four contributions, each tied to a verifiable artifact rather than to a completed empirical comparison: (1) an architectural specification for a local KG-LLM agent integrating SSN/SOSA grounding, hybrid SPARQL/vector retrieval, SHACL validation, and PROV-O provenance; (2) a SHACL-gated validation and repair loop with an explicit bounded repair policy; (3) a structured evidence-package output contract linking KG support resources, document citations, SHACL status, and a provenance trace; and (4) a four-strategy evaluation protocol for examining the incremental value of graph structure, expert ontology grounding, and agentic validation/provenance. The scope is deliberately architectural: the shapes, output contract, trace contract, repair policy, and protocol can be inspected independently of future performance results. The empirical study will execute the comparison specified here.

2. Related work

MFG-AgentGraph draws on four adjacent research streams.

RAG and GraphRAG. RAG retrieves relevant passages and then generates an answer [1], but vector-only retrieval does not represent assets, sensors, or action validity as structured objects. GraphRAG builds a graph from LLM-extracted entities and relations, then answers over community summaries [2]; yet that graph is not an expert manufacturing ontology, and its outputs are neither constraint-validated nor provenance-traced. Variants such as LightRAG, RAPTOR, and KAG improve speed or hierarchical retrieval, but they keep the same separation from explicit validation and provenance [13, 14, 15].

KG-guided and ontology-grounded reasoning. A useful taxonomy places plain RAG at **S1**, LLM-built KG retrieval at **S2**, ontology-guided retrieval at **S3**, and a synergized LLM+KG agent at **S4** [16]. We adopt this **S1**–**S4** ladder as the four strategies we compare throughout the paper (**S1** naive RAG, **S2** GraphRAG, **S3** OG-RAG, and **S4** the proposed KG-LLM agent) and specify each configuration in Section 3 (Table 3). Think-on-Graph and Reasoning on Graphs show that an LLM can explore KG paths [4, 5], and Think-on-Graph 2.0 extends the pattern to hybrid graph-and-text retrieval [17], while OG-RAG demonstrates the value of grounding retrieval in a domain ontology rather than in an LLM-built structure [18]. These methods motivate the retrieval and reasoning core of MFG-AgentGraph, but

they do not close the manufacturing auditability gap: they do not enforce SHACL-validated outputs or emit PROV-O audit traces.

Agent foundations and self-correcting generation. ReAct interleaves reasoning traces and tool actions, so a KG can appear as one of the agent's tools [19]. Self-RAG adds a learned neural critic to decide when to retrieve and whether a generation is supported [20], and Reflexion shows that verbal feedback from an evaluator can drive iterative agent self-correction [21]. MFG-AgentGraph instantiates the same generate–critique–refine pattern, but replaces the learned or model-generated critic with a deterministic SHACL critic, trading some flexibility for reproducibility and auditability. On the symbolic side, Ahmetaj et al. study repairing SHACL violations at the data level using answer-set programming [22]; MFG-AgentGraph instead repairs the candidate recommendation, treating the violation report as feedback to the generating agent. xpSHACL and AML-SHACL show that LLMs can explain or generate SHACL constraints [23, 24]; here SHACL is used as an active repair gate inside the agent loop.

Standards for grounding, validation, and provenance. SSN/SOSA gives the ontology backbone for sensors, observations, and features of interest [12, 25]; SHACL supplies the constraint language [9]; PROV-O records activities, entities, and agents [10]; and PROV-AGENT extends this model to agentic workflows [11]. Manufacturing reference architectures such as RAMI 4.0 and the Asset Administration Shell motivate interoperable asset models [26, 27], and SAREF complements SSN/SOSA in industrial settings [28].

Gap summary. Existing RAG and GraphRAG systems improve retrieval but do not make manufacturing recommendations constraint-valid and auditable. KG-guided systems reason over graph paths but omit SHACL and PROV-O. Provenance frameworks trace actions but do not validate decisions. Table 1 summarizes the difference; the final row states the design targets of the proposed architecture, not measured capabilities.

Table 1
Related-work gap motivating MFG-AgentGraph. Column keys: *Str. KG* = structured knowledge graph; *Exp. ont.* = expert domain ontology; *Agent loop* = iterative agent loop; *SHACL gate* = SHACL validation gate; *PROV-O* = PROV-O provenance trace; *Local mfg.* = local factory deployment. In the last row, *Pilot* marks a capability demonstrated at small scale by the pilot of Section 4, and *By design* an intended target not yet demonstrated (the pilot used a small synthetic ontology and a cloud-served model, so expert-ontology depth and local deployment remain design targets).

System family	Str. KG	Exp. ont.	Agent loop	SHACL gate	PROV-O	Local mfg.	Added cost
Naive RAG	No	No	No	No	No	Partial	Low
GraphRAG	LLM-built	No	No	No	No	No	LLM index cost
OG-RAG	Partial	Yes	No	No	No	Limited	Ontology authoring
ToG / RoG	Yes	Domain-generic	Yes	No	No	No	KG curation
AGENTiGraph [29]	Yes	Domain-generic	Yes	No	No	No	Multi-agent orchestration
PROV-AGENT [11]	Prov. graph	No mfg. ontology	No (traces workflows)	No	Yes	No	Trace storage
MFG-AgentGraph (proposed)	Pilot	By design	Pilot	Pilot	Pilot	By design	Ontology + SHACL authoring

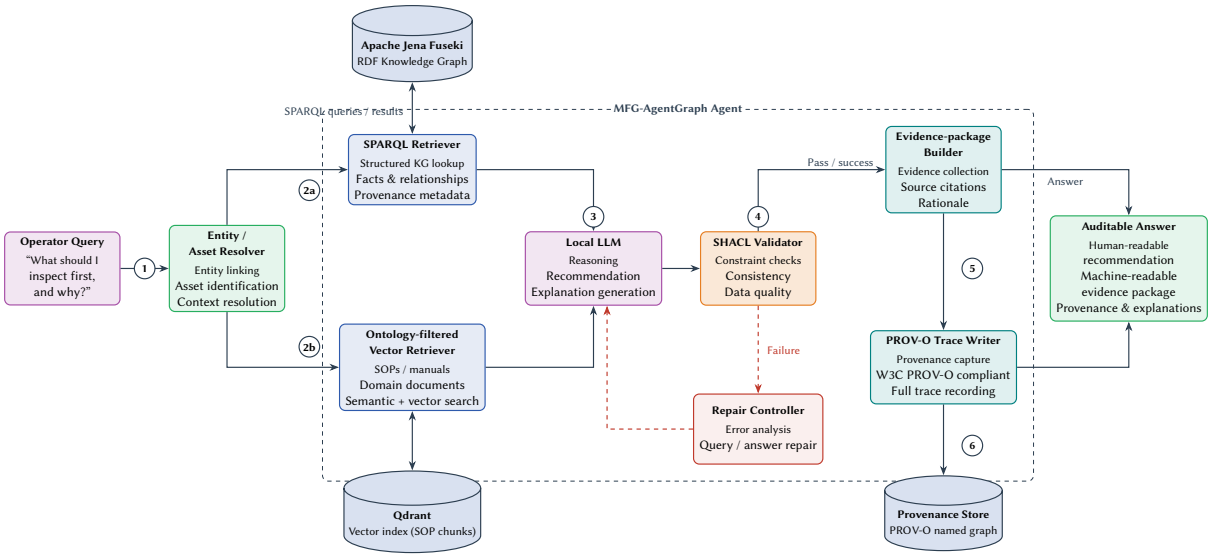


Figure 1: End-to-end MFG-AgentGraph architecture and data flow. An operator query (1) is resolved to KG identifiers, driving hybrid retrieval over the Apache Jena Fuseki RDF store (2a) and the Qdrant vector index (2b). The local LLM generates candidate actions (3) that the SHACL validator checks (4); on failure the repair controller returns violated shape IDs to the agent (dashed path), and only a passing candidate reaches the evidence-package builder (5) and PROV-O trace writer (6), whose trace is stored in a provenance named graph. The operator receives a recommendation that is asset-bound, evidence-backed, constraint-valid, and traceable.

3. MFG-AgentGraph architecture

MFG-AgentGraph is designed as a locally deployable KG-LLM agent architecture for manufacturing anomaly triage. Its purpose is not only to retrieve relevant information, but to return recommendations whose evidence can be inspected, whose structural validity can be checked, and whose generation process can be audited.

3.1. Design requirements and system overview

The architecture is driven by six requirements: (R1) local deployment on factory-controlled hardware; (R2) structured domain grounding via SSN/SOSA; (R3) hybrid evidence retrieval through SPARQL and vector search; (R4) constraint-valid action plans via SHACL; (R5) auditable evidence packages with provenance identifiers; and (R6) a comparable strategy harness exposing S1–S4.

Factory data are organized into two complementary stores. The structured store is an SSN/SOSA-aligned RDF knowledge graph containing assets, sensors, observations, anomaly events, procedures, and provenance records, hosted in Apache Jena Fuseki with factory data, SHACL shapes, and provenance traces kept in separate named graphs. The unstructured store is a Qdrant vector index over SOP and manual passages. At query time, a local KG-LLM agent retrieves from both stores, generates candidate actions, validates them with SHACL, repairs invalid plans using validation feedback, and emits an evidence package. Figure 1 shows the end-to-end architecture and data flow.

3.2. From query to auditable answer

The S4 pipeline has five phases. Entity and asset resolution maps names, timestamps, and sensor references to KG identifiers. Hybrid retrieval then runs SPARQL over the RDF store for structured context and ontology-filtered vector search over SOP chunks for procedural context. The local LLM

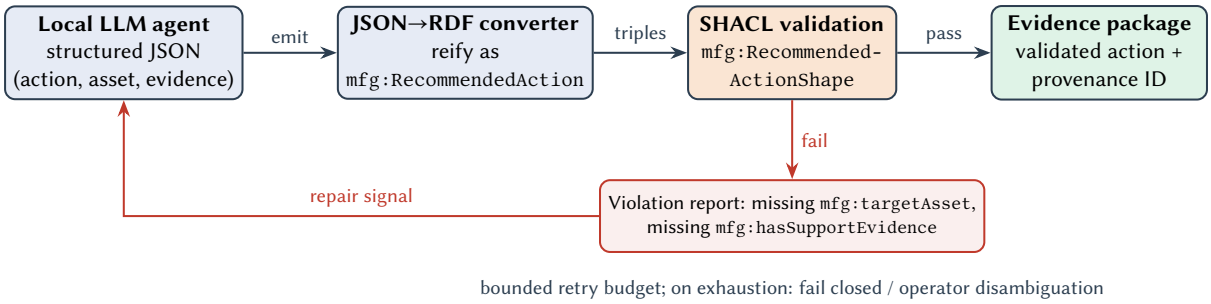


Figure 2: Binding LLM output to SHACL-validated RDF. The LLM emits structured JSON, a converter reifies it as `mfg:RecommendedAction` triples, and SHACL validates the candidate against the shapes graph. Violations are returned to the agent as repair signals under a bounded retry budget; only validated actions leave the gate as part of the evidence package.

generates ranked candidate actions as structured JSON. SHACL validation checks the candidate plan and its support subgraph; if the check fails, violated shape IDs and messages are returned to the agent, which retrieves missing evidence or regenerates the plan until validation passes or a retry limit is reached. Reaching the retry limit is not a crash: the request exits through one of the explicit fallback states of Table 2 (for example, returning "insufficient evidence" or deferring to the operator), so the operator is never shown an unvalidated recommendation. Finally, an evidence package is assembled and a PROV-O trace is written before the answer is returned.

Two boundary decisions determine how these phases should be interpreted. First, the **system boundary**: a single LLM performs candidate generation and repair regeneration, while candidate retrieval, JSON-to-RDF reification, SHACL validation, and provenance writing are deterministic, non-LLM components. The pilot of Section 4 uses one model endpoint for both S1 and S4, and its per-query records score asset grounding separately from action correctness, so selection errors and generation errors remain attributable rather than entangled. Second, **entity resolution is a core design concern, not a peripheral step**: it is the pipeline's single point of failure, because a mis-resolved IRI yields a SHACL-valid, fully traced, but wrongly targeted recommendation that no downstream component is positioned to catch. The architecture therefore does not let the model invent identifiers: retrieval proposes a bounded candidate set of KG identifiers, the LLM only *selects* among them; if no unique asset match is found or multiple candidates remain, it requests operator disambiguation (Table 2) rather than guessing. The pilot evaluates this constrained-selection design over the retrieved candidate set; stress-testing open-world resolution with ambiguous and incomplete queries is part of the planned evaluation protocol (Section 4).

SHACL failures are treated as repair signals, not terminal errors. This yields measurable outcomes: initial validation failure rate, self-correction success rate, number of repair iterations, and final pass rate. We do not overstate SHACL: it checks only encoded constraints and cannot detect missing domain knowledge unless that absence is represented in the shapes or the KG.

3.3. Binding LLM output to SHACL-validated RDF

The central binding is the transformation of an LLM-generated candidate action into RDF that SHACL can validate. The LLM is prompted to emit structured JSON whose fields map directly to RDF properties. A converter reifies the JSON as triples typed as `mfg:RecommendedAction`. Figure 2 illustrates this binding and repair cycle for the pump-station example.

For the pump-station scenario, an initial LLM output might propose "Inspect the pump inlet for cavitation" without naming the asset. After reification, SHACL reports a missing `mfg:targetAsset` and miss-

Listing 1: Repaired structured LLM output for the pump-station query.

```
{
  "ranked_actions": [{
    "rank": 1,
    "action": "Inspect the pump inlet of Pump_12 for cavitation",
    "target_asset": "ex:Pump_12",
    "support_evidence": [
      {"type": "kg_edge", "id": "ex:EvidenceEdge_001"},
      {"type": "doc_chunk", "id": "ex:SOP_Chunk_7"}
    ]
  }]
}
```

ing `mfg:hasSupportEvidence`. The repair controller feeds these messages back to the agent, which regenerates structured output such as Listing 1.

The converter resolves these identifiers to RDF resources rather than validating free-text triple strings. `ex:EvidenceEdge_001` denotes a reified or RDF-star support statement linking an observation to `ex:Pump_12`; the observation uses `sosa:hasFeatureOfInterest`, `sosa:observedProperty`, and a typed `sosa:Result`. `ex:SOP_Chunk_7` is a modeled `mfg:DocumentChunk`. After reification, the action passes `mfg:RecommendedActionShape`. This binding is what makes SHACL an active critic rather than a post-hoc check of free text. A runnable reference implementation of this binding and the `mfg:RecommendedActionShape` gate, validating the pump-station worked example, is openly available in the paper's artifact repository.[1]

3.4. Bounded repair policy

Table 2 maps common shape violations to repair actions and a fallback state.

Table 2
Bounded repair policy for the SHACL validation loop.

Violation class	Example	Repair action	Fallback
Missing target asset	action lacks `mfg:targetAsset`	Re-run entity resolution; regenerate with resolved IRI	Ask operator to disambiguate
Missing observation support	cited observation not in KG	Retrieve observation subgraph for asset/time window	Return insufficient evidence
Missing procedural citation	action has no SOP chunk	Run ontology-filtered vector search for fault/action terms	Mark unsupported; lower rank
Invalid evidence package	no SHACL report or provenance ID	Rebuild package and write trace before returning	Fail closed

The action-plan shape checks that each recommendation targets exactly one known asset IRI in the validation data graph, cites at least one modeled evidence resource, and that the cited evidence actually concerns the target asset (Listing 2).

Here `mfg:Asset` is a subclass of `sosa:FeatureOfInterest`, and `mfg:SupportEvidenceShape` accepts only modeled KG-support statements or document-chunk resources. The final constraint is the rele-

[1]All artifacts accompanying this paper (the SHACL shapes, the runnable proof-of-concept and adversarial gate checks, the pilot benchmark generator, the S1/S4 pilot harness, and the per-query pilot records behind Table 5) are openly available at https://github.com/faizan8201/mfg-agentgraph and archived under the permanent identifier https://doi.org/10.5281/zenodo.22083418.

Listing 2: SHACL shape for a recommended action, including the evidence-relevance constraint.

```
mfg:RecommendedActionShape
  a sh:NodeShape ;
  sh:targetClass mfg:RecommendedAction ;
  sh:property [
    sh:path mfg:targetAsset ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:nodeKind sh:IRI ;
    sh:class mfg:Asset ;
    sh:message "Every recommendation must target exactly one known asset IRI." ;
  ] ;
  sh:property [
    sh:path mfg:hasSupportEvidence ;
    sh:minCount 1 ;
    sh:node mfg:SupportEvidenceShape ;
    sh:message "Every recommendation must cite modeled support evidence." ;
  ] ;
  sh:sparql [
    a sh:SPARQLConstraint ;
    sh:message "Cited evidence must concern the target asset." ;
    sh:select """
      SELECT $this ?ev WHERE {
        $this mfg:hasSupportEvidence ?ev ;
               mfg:targetAsset ?asset .
        FILTER NOT EXISTS {
          { ?ev mfg:supportsObservation/
                sosa:hasFeatureOfInterest/sosa:hosts? ?asset }
          UNION
          { ?ev a mfg:DocumentChunk ;
                mfg:concernsAssetType ?t .
            ?asset mfg:hasAssetType ?t . }
        } } """ ;
  ] .
```

vance check that binds evidence to the target: a KG-support statement must trace through its cited observation to the target asset (or to a station hosting it), and a document chunk must apply to the target's asset type. Without it, a recommendation citing an unrelated asset's evidence would still pass; with it, that case is a validation failure. The artifact repository (footnote 1) runs this as a 64-case matrix over the pilot benchmark in which all 32 own-asset citations conform and all 32 cross-asset citations are rejected. The class constraints also close a second attack: because the repair loop tells the agent which shapes failed, the agent could invent a plausible-looking evidence IRI, but an invented IRI carries no KG typing and fails `mfg:SupportEvidenceShape`, as the repository's adversarial checks demonstrate. Operational deployments add window checks (the cited observation must fall within the anomaly window) on top of the same pattern. Passing SHACL therefore means that the candidate satisfies the encoded structural and evidence constraints; it does not prove that the action is optimal or safe.

3.5. PROV-O evidence trace

The provenance layer records how each recommendation was produced. The S4 agent is modeled as a `prov:SoftwareAgent`; retrieval, LLM, validation, and repair steps are `prov:Activity` instances; and prompts, models, retrieved resources, validation reports, and recommendations are `prov:Entity`

Listing 3: Compact PROV-O trace for the pump-station recommendation.

```
ex:Agent_S4 a prov:SoftwareAgent .
ex:Run_2026_07_02_1403 a prov:Activity ;
  prov:wasAssociatedWith ex:Agent_S4 ;
  prov:used ex:Prompt_42, ex:Model_LocalLLM,
            ex:Obs_2026_07_02_1403, ex:SOP_Chunk_7 .
ex:SHACL_Validation_1 a prov:Activity ;
  prov:used ex:CandidateAction_1, mfg:RecommendedActionShape ;
  prov:generated ex:SHACL_Report_1 .
ex:Rec_InspectPumpInlet a prov:Entity ;
  prov:wasGeneratedBy ex:Run_2026_07_02_1403 ;
  prov:wasDerivedFrom ex:Obs_2026_07_02_1403,
                      ex:SOP_Chunk_7, ex:SHACL_Report_1 ;
  mfg:shaclStatus "passed" ;
  mfg:provenanceId "run-2026-07-02-1403" .
```

instances. PROV-AGENT-style specializations may type tool and model activities, while the trace contract remains expressible in standard PROV-O [10, 30]. Listing 3 shows a compact trace.

When the implementation records model, prompt, retrieval, validation-report, and repair activities, this contract supports audit questions that ordinary explanations cannot answer. The final answer includes a `provenance_id` linking the human-readable recommendation to the machine-readable activity trace. Provenance-write failure is a blocking state: the user-facing response returns "no recommendation: provenance trace unavailable" rather than an untraced recommendation. Because traces can store sensitive prompts and responses, deployments must define retention and access policies for provenance graphs.

3.6. Strategy configurations

The architecture is built to be evaluated through a controlled comparison of four strategies, implemented as configurations of one shared harness (Table 3).

Table 3
Strategy configurations for evaluation.

Strategy	Role	Mechanisms present
S1 Naive RAG	Minimal baseline	Vector retrieval + LLM generation
S2 GraphRAG	Graph baseline	LLM-built KG + community summaries
S3 OG-RAG	Ontology baseline	Expert ontology + hybrid retrieval
S4 KG-LLM Agent	Proposed system	Full architecture: KG grounding, SHACL gate, PROV-O trace

3.7. Deployment and measurable convergence

The implementation stack is deliberately off-the-shelf: Apache Jena Fuseki for the RDF graph, Qdrant for vector storage, sentence-transformer embeddings, and a local LLM runtime such as Ollama, llama.cpp, or vLLM. Before running the experiment we hold the deployment fixed (one workstation, one quantized 7B–13B model family, and fixed context/output lengths and retrieval top-k) so that the strategy is the only variable; other hardware and model sizes are then explored separately as sensitivity analyses.

The repair loop terminates in one of three states: validation passes, the retry budget (three repair attempts in the reference harness) is exhausted, or entity resolution requests operator disambiguation. Because each repair iteration is bounded to one additional LLM call plus SHACL validation, convergence is measurable rather than open-ended. Exact latency targets are left to the companion implementation study; the architecture is designed so that every stage, including entity resolution, SPARQL retrieval, vector retrieval, generation, validation, repair, and provenance writing, is separately instrumented.

4. Proposed evaluation protocol

This section specifies the protocol that the implementation study will execute; unless stated otherwise, all quantities below are planned scale targets, not completed measurements. The one exception is the preliminary pilot at the end of this section (Table 5), which already runs the two extreme strategies, **S1** and **S4**, end-to-end and demonstrates the SHACL repair-gate in operation. The protocol asks a single question: does adding expert ontology grounding, SHACL validation, and provenance tracing to a local KG-LLM agent improve support for manufacturing anomaly triage compared with simpler retrieval strategies? All four strategies (**S1** naive RAG, **S2** GraphRAG, **S3** OG-RAG, **S4** the proposed KG-LLM agent) share the same anomaly scenarios, query set, local deployment envelope, and scoring criteria.

Dataset and scenario construction. The anomaly source is the MIMII industrial machine-sound dataset, which contains normal and anomalous sounds for pumps, fans, sliders, and valves [31]. Newer variants such as MIMII DG and MIMII DUE target domain generalization and domain shifts under changing operational and environmental conditions [32, 33]; we start from the original MIMII for its balanced machine-type coverage and treat detector choice as orthogonal to the architecture. MIMII supplies acoustic events but not an asset hierarchy, maintenance history, SOP corpus, or validated KG. The benchmark therefore combines MIMII-derived events with curated manufacturing context. That combination is treated as a validity boundary: the benchmark construction must publish the synthetic-context schema and generation rules so reviewers can separate acoustic-event grounding from real plant decision-support performance. To prevent evaluation leakage, the synthetic KG and SOP corpus are authored before, and independently of, any strategy's output, and are frozen before S1–S4 are run; no strategy is tuned against the benchmark on which it is scored. As planned scale targets, the benchmark will comprise a KG on the order of 10^4–10^5 triples covering the four MIMII machine types with multiple asset instances, a procedural corpus of approximately 50 SOP/manual documents, and at least 100 queries spanning five difficulty tiers (single-hop facts, multi-hop paths, action plans, contrastive questions, and provenance questions). Gold-standard first-inspection actions will be authored by two domain annotators working from the SOP corpus independently of any retrieval index, with inter-annotator agreement reported and disagreements adjudicated before evaluation.

Strategy comparison. S1 is vector-only RAG; S2 is GraphRAG-style graph summarization; S3 is ontology-guided retrieval without agentic validation or provenance; and S4 is the full KG-LLM agent architecture. The ladder approximates an incremental comparison, but each strategy family still introduces implementation-level differences. Improvements are therefore interpreted as evidence consistent with (not definitive proof of) the added mechanism. Retrieval statistics, graph statistics, prompts, output contracts, seeds or deterministic settings, and model/runtime versions will be disclosed. Within S4, secondary ablations compare post-hoc SHACL without repair, SHACL-gated repair without a user-facing provenance package, and the full system. GraphRAG's indexing cost and local-model graph quality are reported as explicit confounds rather than treating S2 as a reproduction of a cloud-scale implementation.

Research questions and metrics. Table 4 maps each research question to its metrics.

Table 4
Research questions and metrics.

RQ	Focus	Metrics
RQ1	Answer quality	RAGAS faithfulness, answer relevancy, context precision/recall [34, 35], evidence-support coverage, gold-label accuracy
RQ2	Validation	SHACL pass rate, initial failure rate, self-correction success rate, repair iterations
RQ3	Expert-assessed auditability	Structured expert review (5–10 domain engineers), provenance-query tasks, trace completeness
RQ4	Local feasibility	p50/p95 end-to-end and per-stage latency, peak RAM/VRAM, model size, throughput

Two measurement caveats apply. RAGAS-style metrics rely on an LLM judge [34, 35]; the judge model and whether it runs inside or outside the local envelope will be reported explicitly, since evaluation infrastructure need not share deployment constraints; to avoid circular evaluation, RAGAS scores are reported alongside, and cross-checked against, human gold-label accuracy, so no headline claim depends on the LLM judge alone. For RQ3, the exploratory expert review assesses auditability and traceability (not operational trust or field effectiveness) by comparing strategy outputs and answering provenance questions. Results will be analyzed as paired comparisons across strategies on the shared query set, reported per difficulty tier with bootstrap confidence intervals.

The protocol also mandates reporting negative cases: SHACL rejections, repair failures, incomplete traces, latency envelope violations, and abstention behaviour on unanswerable or under-specified queries, evaluated in the spirit of unanswerable-query RAG benchmarks [36]. These failures are first-class results because the architecture's value depends on making invalid or unsupported outputs visible.

Demonstrated: preliminary pilot of the repair-gate in operation. As an early feasibility check (not the full study specified above), we ran **S1** and **S4** on a small synthetic benchmark of 32 anomaly-triage queries spanning eight assets and eight fault modes, using a local-deployment-class 8B instruction-tuned model (`meta-llama-3.1-8b-instruct`) at temperature 0. The model was served through an OpenAI-compatible API endpoint rather than on factory-controlled hardware, so wall-clock time is not a local-latency measurement; we report LLM calls per query as the deployment-independent cost proxy and defer RQ4 latency to the implementation study. Both configurations see the same query and SOP snippet; **S4** additionally sees the retrieval layer's candidate IRIs, so the SHACL gate remains the only independent variable. Gold-standard first actions are read from the KG fault-mode edges, and every metric is judge-free.

Table 5 summarises the outcome. **S1** and **S4** reach comparable first-action accuracy. The two-query difference is descriptive only (no significance test is meaningful at $n = 32$), so we claim comparability, not parity. Only **S4**, however, produces machine-checkable output: every recommendation is bound to a KG asset IRI and carries an evidence package and PROV-O trace, versus none for **S1**. The two queries that account for the entire accuracy gap both concern `Slider_2`; in one of them, **S4**'s answer follows the SOP wording ("inspect the mechanism") but misses the gold keyword ("jam"), an artifact of substring scoring rather than a quality difference, and in the other **S4** names the asset but not the gold part. At this scale we therefore read the gap as noise, while noting that constraining output to a fixed structure may not be accuracy-neutral in general. The trade-off we claim is auditability, not accuracy. The 0% initial SHACL pass is a modelled instantiation of the missing-citation repair path of Table 2, not a spontaneous failure rate: attempt 1 withholds the retrieved evidence IRIs by construction, so the pilot shows the mechanism recovering from a known structured failure in a mean of 1.0 repair iterations, at a cost of two LLM calls instead of one. A control run that supplies the evidence IRIs from the first attempt answers the complementary question: the natural initial failure

rate is then 0% on the same 32 queries, with one LLM call per query, no repairs triggered, and accuracy again 90.6% matching **S1** exactly. The repair loop is therefore a contingency for harder settings (larger benchmarks, noisier retrieval), not a cost every query pays; the withheld-evidence mode exists to exercise it. Per-query records for both modes ship in the artifact repository. Had the retry budget been exhausted, the harness would have abstained through a fallback state of Table 2 rather than returning an unvalidated recommendation; this path was never triggered in the pilot. Asset resolution, i.e., constrained selection over the retrieved candidate list, was correct in every case. The doubled call count is also the only measured cost figure at this stage: the repair cycle doubles generation cost per query, and a per-stage latency budget, published as evaluation targets rather than measurements, places nominal end-to-end latency near a 30-second envelope for a quantised 7B–13B model with a pessimistic case exceeding it; absolute wall-clock numbers are deferred to the implementation study. These numbers illustrate the mechanism at small scale; isolating where the value comes from (in-loop versus post-hoc validation, KG-only planning) requires the protocol's ablations, not a 32-query pilot. The full S1–S4 comparison, a larger benchmark, and latency measurement remain future work. The benchmark generator, harness, SHACL shapes, and per-query records behind Table 5 are openly available in the paper's artifact repository (footnote 1).

Table 5
Preliminary **S1**-vs-**S4** pilot on a 32-query synthetic benchmark (`meta-llama-3.1-8b-instruct`, temperature 0, same query and SOP snippet for both configurations, all metrics judge-free). Accuracy differences at $n = 32$ are descriptive, not significance-tested; the informative result is the capability gap in the lower rows. "—" marks a metric that does not apply to **S1**, which emits free text and no validation loop. All rows are measured except **S1**'s three 0% rows, which are structural: free text cannot bind KG IRIs or emit packages and traces by construction.

Metric (judge-free)	**S1**	**S4**
Gold-standard first action accuracy	90.6%	84.4%
(raw count)	29/32	27/32
Recommendation bound to a KG asset IRI	0%	100%
Machine-checkable evidence package	0%	100%
PROV-O trace emitted	0%	100%
Self-correction success (initial fail → pass)	—	100%
Mean repair iterations	—	1.0
Mean LLM calls per query	1.0	2.0

5. Limitations and threats to validity

Several limitations bound the claims in this paper. First, the architecture is specified but only partially evaluated: the quantities in Section 4 are planned targets and the full-scale S1–S4 comparison has not been run, though the pilot reported in Section 4 demonstrates the SHACL repair-gate of Section 3 in operation on a small synthetic benchmark. In that pilot, the gold-standard first actions are derived from the same KG fault-mode edges that S4 retrieves from, which structurally favours the KG-grounded configuration and is a further reason to read the pilot as a mechanism demonstration rather than a comparative result; the planned benchmark mitigates this through independently authored, adjudicated gold labels (Section 4). Second, SHACL validation is only as complete as the encoded shapes: passing the gate certifies structural and evidence constraints, not the optimality or safety of the recommended action. Third, entity resolution upper-bounds every downstream grounding guarantee; a mis-resolved asset yields a well-formed but wrongly-targeted recommendation, which is why the architecture constrains the model to selecting among retrieved candidates and defers cases without a unique asset match to the operator (Section 3). Fourth, the pilot benchmark is fully synthetic (MIMII-inspired machine types with invented symptom strings), and MIMII acoustic events enter only the planned full

benchmark, so results characterize modeled decision support, not validated real-plant performance. Fifth, RQ1 depends in part on an LLM judge; its scores are cross-checked against human gold-label accuracy. Finally, the planned evaluation fixes one workstation and one quantized 7B–13B model family; generalization to other hardware and model scales is a sensitivity analysis, not a demonstrated property.

6. Conclusion

MFG-AgentGraph is a locally deployable KG-LLM agent architecture for auditable manufacturing anomaly triage. Its central claim is that manufacturing decision support requires recommendations that are grounded in structured asset context, supported by procedural evidence, checked against domain constraints, and traceable to the data, tools, and model invocations that produced them. The architecture combines SSN/SOSA-aligned RDF modelling, hybrid SPARQL and vector retrieval, local LLM reasoning, SHACL validation and repair, and PROV-O/PROV-AGENT-style provenance traces.

This paper contributes the architecture, output and trace contracts, repair policy, and falsifiable S1/S2/S3/S4 protocol. The implementation study will test whether the full KG-LLM agent strategy improves evidence-supported answer quality, validation success, expert-assessed auditability, and local deployment feasibility compared with simpler retrieval baselines, building on the openly released benchmark, shapes, and harness (footnote 1). The broader goal is to make every manufacturing recommendation inspectable, constraint-checkable, and traceable.

Declaration on Generative AI

During the preparation of this work, the authors used **Claude** (Anthropic) and **Kimi** (Moonshot AI) in order to: Drafting content; Improve writing style; Grammar and spelling check. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Advances in Neural Information Processing Systems, volume 33, 2020, pp. 9459–9474. URL: https://arxiv.org/abs/2005.11401. arXiv:2005.11401.

[2] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, J. Larson, From local to global: A graph RAG approach to query-focused summarization, 2024. URL: https://arxiv.org/abs/2404.16130. arXiv:2404.16130.

[3] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, S. Tang, Graph retrieval-augmented generation: A survey, 2024. URL: https://arxiv.org/abs/2408.08921. arXiv:2408.08921.

[4] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, J. Guo, Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph, in: International Conference on Learning Representations, 2024. URL: https://arxiv.org/abs/2307.07697. arXiv:2307.07697.

[5] L. Luo, Y.-F. Li, G. Haffari, S. Pan, Reasoning on graphs: Faithful and interpretable large language model reasoning, in: International Conference on Learning Representations, 2024. URL: https://arxiv.org/abs/2310.01061. arXiv:2310.01061.

[6] S. K. Freire, C. Wang, M. Foosherian, S. Wellsandt, S. Ruiz-Arenas, E. Niforatos, Knowledge shar-

ing in manufacturing using large language models: User evaluation and model benchmarking, 2024. URL: https://arxiv.org/abs/2401.05200. arXiv:2401.05200.

[7] J. Lim, B. Vogel-Heuser, I. Kovalenko, Large language model-enabled multi-agent manufacturing systems, 2024. URL: https://arxiv.org/abs/2406.01893. arXiv:2406.01893.

[8] Y. Li, H. Zhao, H. Jiang, Y. Pan, Z. Liu, Z. Wu, P. Shu, J. Tian, T. Yang, S. Xu, Y. Lyu, P. Blenk, J. Pence, J. Rupram, E. Banu, N. Liu, L. Wang, W. Song, X. Zhai, K. Song, D. Zhu, B. Li, X. Wang, T. Liu, Large language models for manufacturing, 2024. URL: https://arxiv.org/abs/2410.21418. arXiv:2410.21418.

[9] W3C RDF Data Shapes Working Group, Shapes constraint language (SHACL), W3C Recommendation, 2017. URL: https://www.w3.org/TR/shacl/.

[10] W3C Provenance Working Group, PROV-O: The PROV ontology, W3C Recommendation, 2013. URL: https://www.w3.org/TR/prov-o/.

[11] R. Souza, A. Gueroudji, S. DeWitt, D. Rosendo, T. Ghosal, R. Ross, P. Balaprakash, R. Ferreira da Silva, PROV-AGENT: Unified provenance for tracking AI agent interactions in agentic workflows, 2025. URL: https://arxiv.org/abs/2508.02866. arXiv:2508.02866.

[12] K. Janowicz, A. Haller, S. J. D. Cox, D. Le Phuoc, M. Lefrançois, SOSA: A lightweight ontology for sensors, observations, samples, and actuators, Journal of Web Semantics 56 (2019) 1–10. URL: https://arxiv.org/abs/1805.09979. doi:10.1016/j.websem.2018.06.003. arXiv:1805.09979.

[13] Z. Guo, L. Xia, Y. Yu, T. Ao, C. Huang, LightRAG: Simple and fast retrieval-augmented generation, 2024. URL: https://arxiv.org/abs/2410.05779. arXiv:2410.05779.

[14] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, C. D. Manning, RAPTOR: Recursive abstractive processing for tree-organized retrieval, 2024. URL: https://arxiv.org/abs/2401.18059. arXiv:2401.18059.

[15] L. Liang, M. Sun, Z. Gui, Z. Zhu, Z. Jiang, L. Zhong, Y. Qu, P. Zhao, Z. Bo, J. Yang, H. Xiong, L. Yuan, J. Xu, Z. Wang, Z. Zhang, W. Zhang, H. Chen, W. Chen, J. Zhou, KAG: Boosting LLMs in professional domains via knowledge augmented generation, 2024. URL: https://arxiv.org/abs/2409.13731. arXiv:2409.13731.

[16] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: A roadmap, IEEE Transactions on Knowledge and Data Engineering 36 (2024) 3580–3599. URL: https://arxiv.org/abs/2306.08302. doi:10.1109/TKDE.2024.3352100. arXiv:2306.08302.

[17] S. Ma, C. Xu, X. Jiang, M. Li, H. Qu, C. Yang, J. Mao, J. Guo, Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation, 2024. URL: https://arxiv.org/abs/2407.10805. arXiv:2407.10805.

[18] K. Sharma, P. Kumar, Y. Li, OG-RAG: Ontology-grounded retrieval-augmented generation for large language models, 2024. URL: https://arxiv.org/abs/2412.15235. arXiv:2412.15235.

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing reasoning and acting in language models, in: International Conference on Learning Representations, 2023. URL: https://arxiv.org/abs/2210.03629. arXiv:2210.03629.

[20] A. Asai, Z. Wu, Y. Wang, A. Sil, H. Hajishirzi, Self-RAG: Learning to retrieve, generate, and critique through self-reflection, 2023. URL: https://arxiv.org/abs/2310.11511. arXiv:2310.11511.

[21] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, S. Yao, Reflexion: Language agents with verbal reinforcement learning, in: Advances in Neural Information Processing Systems, volume 36, 2023. URL: https://arxiv.org/abs/2303.11366. arXiv:2303.11366.

[22] S. Ahmetaj, R. David, A. Polleres, M. Šimkus, Repairing SHACL constraint violations using answer set programming, in: The Semantic Web – ISWC 2022, volume 13489 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 375–391. URL: https://doi.org/10.1007/978-3-031-19433-7_22. doi:10.1007/978-3-031-19433-7_22.

[23] G. C. Publio, J. E. Labra Gayo, xpSHACL: Explainable SHACL validation using retrieval-augmented generation and large language models, 2025. URL: https://arxiv.org/abs/2507.08432. arXiv:2507.08432, Accepted at the 2nd LLM+Graph Workshop, co-located with VLDB 2025.

[24] T. Westermann, A. Köcher, F. Gehlhoff, Automated validation of textual constraints

against AutomationML via LLMs and SHACL, 2025. URL: https://arxiv.org/abs/2506.10678. arXiv:2506.10678.

[25] W3C and OGC Spatial Data on the Web Working Group, Semantic sensor network ontology, W3C Recommendation, 2017. URL: https://www.w3.org/TR/vocab-ssn/.

[26] German Institute for Standardization (DIN), DIN SPEC 91345:2016-04: Reference Architecture Model Industrie 4.0 (RAMI 4.0), Technical Report, German Institute for Standardization, 2016. URL: https://doi.org/10.31030/2436156.

[27] S. R. Bader, I. Grangel-González, P. Nanjappa, M.-E. Vidal, M. Maleshkova, Modelling with AAS and RDF in industry 4.0, Computers in Industry 154 (2023) 103910. URL: https://doi.org/10.1016/j.compind.2023.103910. doi:10.1016/j.compind.2023.103910.

[28] ETSI, ETSI TS 103 264 V3.1.1: Smart Applications REFerence Ontology (SAREF), Technical Report, European Telecommunications Standards Institute, 2020. URL: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/03.01.01_60/ts_103264v030101p.pdf.

[29] X. Zhao, M. Blum, F. Gao, Y. Chen, B. Yang, L. Marquez-Carpintero, M. Pina-Navarro, Y. Fu, S. Morikawa, Y. Iwasawa, Y. Matsuo, C. Park, I. Li, AGENTiGraph: A multi-agent knowledge graph framework for interactive, domain-specific LLM chatbots, in: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM 2025), Demo Track, 2025. URL: https://arxiv.org/abs/2508.02999. arXiv:2508.02999.

[30] L. Moreau, P. Missier, PROV-DM: The PROV data model, W3C Recommendation, 2013. URL: https://www.w3.org/TR/prov-dm/.

[31] H. Purohit, R. Tanabe, K. Ichige, T. Endo, Y. Nikaido, K. Suefusa, Y. Kawaguchi, MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection, 2019. URL: https://arxiv.org/abs/1909.09347. arXiv:1909.09347, Dataset record: Zenodo 3384388.

[32] K. Dohi, T. Nishida, H. Purohit, R. Tanabe, T. Endo, M. Yamamoto, Y. Nikaido, Y. Kawaguchi, MIMII DG: Sound dataset for malfunctioning industrial machine investigation and inspection for domain generalization task, 2022. URL: https://arxiv.org/abs/2205.13879. arXiv:2205.13879.

[33] R. Tanabe, H. Purohit, K. Dohi, T. Endo, Y. Nikaido, T. Nakamura, Y. Kawaguchi, MIMII DUE: Sound dataset for malfunctioning industrial machine investigation and inspection with domain shifts due to changes in operational and environmental conditions, in: 2021 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), 2021, pp. 21–25. doi:10.1109/WASPAA52581.2021.9632802.

[34] S. Es, J. James, L. Espinosa-Anke, S. Schockaert, RAGAs: Automated evaluation of retrieval augmented generation, in: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, 2024. URL: https://aclanthology.org/2024.eacl-demo.16/. arXiv:2309.15217.

[35] J. Saad-Falcon, O. Khattab, C. Potts, M. Zaharia, ARES: An automated evaluation framework for retrieval-augmented generation systems, 2024. URL: https://arxiv.org/abs/2311.09476. arXiv:2311.09476, Proceedings of NAACL 2024.

[36] X. Peng, P. K. Choubey, C. Xiong, Unanswerability evaluation for retrieval augmented generation, 2025. URL: https://arxiv.org/abs/2412.12300. arXiv:2412.12300.