Evidence-Driven Reasoning for Industrial Maintenance Using Heterogeneous Data

Fearghal O'Donncha[1], Nianjun Zhou[1], Natalia Martinez[1], James T Rayfield[1], Fenno F. Heath III[1], Abigail Langbridge[2], Roman Vaculin[1]
[1]IBM [2]Imperial College London, UK
Correspondence to: feardonn@ie.ibm.com

Abstract

Industrial maintenance platforms contain rich but fragmented evidence, including free-text work orders, heterogeneous operational sensors or indicators, and structured failure knowledge. These sources are often analyzed in isolation, producing alerts or forecasts that do not support conditional decision-making: given this asset history and behavior, what is happening and what action is warranted?

We present Condition Insight Agent, a deployed decision-support framework that integrates maintenance language, behavioral abstractions of operational data, and engineering failure semantics to produce evidence-grounded explanations and advisory actions. The system constrains reasoning through deterministic evidence construction and structured failure knowledge, and applies a rule-based verification loop to suppress unsupported conclusions.

Case studies from production CMMS deployments show that this verification-first design operates reliably under heterogeneous and incomplete data while preserving human oversight. Our results demonstrate how constrained LLM-based reasoning can function as a governed decision-support layer for industrial maintenance.

1 Introduction

Enterprise asset-intensive organizations rely on condition-based maintenance to reduce downtime and manage operational risk. Computerized Maintenance Management Systems (CMMS) aggregate health scores, alerts, work orders, and operational indicators, but rarely support the conditional reasoning required for maintenance decisions.

In practice, decisions depend on reasoning across multiple forms of evidence: given a change in behavior, which failure hypotheses are plausible, which align with maintenance history, and what action is appropriate? Existing systems fall short because evidence remains fragmented across meters, work orders, and engineering knowledge, while relationships between operational signals and asset models are often uncertain. Practitioners must manually reconcile these sources under time pressure reliant on expert judgment.

This challenge is compounded by industrial data realities: operational indicators originate from heterogeneous IoT and SCADA (Supervisory Control and Data Acquisition) platforms with inconsistent naming and limited shared ontology; maintenance history is largely unstructured text; and engineering artifacts such as Failure Modes and Effects Analysis (FMEA) are rarely integrated into data-driven workflows. Supporting maintenance reasoning therefore requires controlled reasoning over heterogeneous, partially structured, and semantically uncertain evidence. Large language models (LLMs) offer flexible reasoning, but naively applying generative models in reliability-critical settings introduces risks. Unconstrained agents may hallucinate explanations or produce fluent but unsupported recommendations. In industrial maintenance, trust depends on traceability, engineering consistency, and explicit linkage between claims and evidence.

We present *Condition Insight*, a deployed reasoning framework that combines deterministic evidence construction with constrained LLM-based synthesis. Meter histories are abstracted into behavioral summaries, work orders into maintenance patterns, and FMEA-derived semantics bound the space of admissible explanations. The system is integrated into enterprise CMMS workflows and evaluated across large asset portfolios.

Our solution and contributions for this paper are:

- We define *Condition Insight* as a deployed decision-support task that integrates operational indicators, maintenance history, and engineering failure knowledge into evidence-grounded explanations.

- We introduce a verification-first architecture that separates (i) deterministic evidence construction from heterogeneous maintenance data into a structured evidence packet, (ii) constrained LLM synthesis operating solely over this evidence, and (iii) rule-based governance aligned with failure semantics.

- We develop a deterministic evidence construction pipeline that transforms weakly aligned sources (work orders, meters, alerts, and FMEA knowledge) into auditable, structured representations used as the sole factual basis for reasoning.

- We introduce a deterministic verification loop that cross-checks LLM outputs against operational rules and structured evidence, ensuring traceability and suppressing unsupported conclusions.

- We demonstrate through controlled evaluation and deployment that this architecture improves grounding, rule compliance, and hallucination resistance compared to unconstrained LLM pipelines.

- We report deployment lessons from enterprise CMMS environments with heterogeneous IoT data and sparse ontology support.

Beyond this application, our results illustrate that reliable industrial LLM systems depend on structured evidence and explicit governance constraints, enabling evidence-grounded reasoning to operate in reliability-critical environments where outputs must remain auditable.

2 Related Work

Data-Driven and Knowledge-Driven Maintenance. Condition-based and predictive maintenance research has focused on anomaly detection, fault diagnosis, and remaining useful life estimation from sensor data (Carvalho et al., 2019; Lei et al., 2018; Teixeira et al., 2020). These approaches typically produce alerts or forecasts rather than integrated explanations that connect observed behavior to maintenance history and engineering failure semantics. In parallel, knowledge-driven methods employ rule-based systems, ontologies, digital twins, and FMEA to encode asset structure and causal failure knowledge (Lu et al., 2020; Langbridge et al., 2023). While interpretable, such approaches often depend on comprehensive ontologies and consistent data alignment, which are difficult to sustain in enterprise environments with heterogeneous assets and evolving data schemas (Moosavi et al., 2024; Ahmed et al., 2022).

LLMs, Agents, and Constrained Generation. Large language models (LLMs) enable multi-step reasoning through retrieval-augmented generation and tool-augmented agents (Brown et al., 2020; Lewis et al., 2020; Yao et al., 2022). These capabilities have motivated applications in industrial operations, including agentic anomaly detection and asset-oriented benchmarks (Patel et al., 2025; Timms et al., 2024; Hosseini and Seilani, 2025). However, many agentic systems prioritize autonomy over constraint, which poses risks in reliability-critical settings (Acharya et al., 2025; Ferrag et al., 2025). Recent work on reliable generation introduces verification loops, self-refinement, human-in-the-loop evaluation, and LLM-based judging to improve grounding and reduce hallucination (Madaan et al., 2023; Zheng et al., 2023; Wang et al., 2020; Langer and Landers, 2021). Our work extends this line by demonstrating a deployed reasoning framework in which deterministic evidence construction and rule-based governance explicitly bound LLM synthesis under real-world maintenance constraints.

3 Problem Setting and Task

We study conditional reasoning for maintenance decision support in an enterprise CMMS deployment spanning multiple asset classes. Rather than prediction, the objective is decision-ready interpretation of asset condition under heterogeneous and imperfect evidence.

Condition Insight Task

Given maintenance evidence (historical work orders, asset-level operational indicators, and structured failure knowledge) the task is to generate an interpretable explanation of an asset's current or emerging condition together with appropriate maintenance recommendations. The output is an evidence-grounded narrative linking observed behavior to maintenance history and plausible failure mechanisms.

Evidence Sources The system operates over three complementary but weakly aligned evidence types:

- **Work orders** document past symptoms, diagnoses, and interventions in largely unstructured text, capturing experiential maintenance

knowledge.

- **Operational indicators ("meters")** are low-frequency, curated measurements (e.g., runtime, periodic readings, discrete states) whose meaning arises from temporal patterns and whose semantics vary across cumulative, incremental, and categorical forms.

- **Failure knowledge** is represented using Failure Modes and Effects Analysis (FMEA), linking asset components to plausible failure mechanisms and influencing factors that constrain admissible explanations.

Integration Constraints In practice, these sources are not natively aligned. Operational indicators originate from heterogeneous IoT and SCADA systems with inconsistent naming and limited shared ontology, and mappings between telemetry points and CMMS assets or components are often partial or ambiguous. Consequently, schema-level integration is unreliable and rule-based reasoning over raw signals is brittle. Conditional reasoning must therefore operate on abstracted behavioral evidence rather than authoritative semantic mappings while tolerating incomplete context and uncertain data relationships.

These heterogeneous sources are not passed directly to the LLM, as their weak alignment and inconsistent semantics make direct reasoning unreliable. Instead, they are deterministically transformed into a structured evidence representation (`asset_facts`) that serves as the sole input to downstream reasoning. This transformation converts raw, partially structured maintenance data into bounded, interpretable evidence, including normalized asset metadata (`asset_details_facts`), aggregated work-order summaries (`workorder_facts`), behavioral abstractions of meter data (`meter_facts`), normalized alerts (`alert_facts`), and aligned failure-mode knowledge (`fmea_facts`). This evidence construction step defines the admissible factual basis for all subsequent reasoning.

4 Condition Insight Framework

The Condition Insight system separates deterministic evidence construction from constrained LLM reasoning. This separation reflects a key deployment requirement: explanations must remain traceable to operational evidence and consistent with engineering knowledge, even when underlying data is heterogeneous or incomplete.

Figure 1 illustrates the pipeline. Heterogeneous asset data (including work orders, operational indicators, alerts, asset metadata, and failure knowledge) are first transformed into a structured evidence representation (`asset_facts`) comprising normalized asset details, work-order summaries, meter-derived behavioral evidence, alert facts, and aligned failure-mode knowledge. A domain LLM agent then performs constrained synthesis over this evidence, followed by a verification stage that governs key decision elements.

4.1 Overview of Reasoning Pipeline

The system operates in two stages:

- **Analytics and Evidence Construction** produces a structured evidence packet (`asset_facts`) summarizing asset behavior, maintenance history, and failure semantics.

- **Constrained LLM Reasoning** generates condition insights and recommendations using this evidence, while deterministic checks verify adherence to operational criteria.

This separation decouples deterministic signal extraction from semantic interpretation, allowing the LLM to reason over curated, auditable evidence rather than raw operational streams.

4.2 Deterministic Evidence Construction

This stage deterministically converts heterogeneous maintenance data into interpretable, auditable evidence for downstream reasoning.

Work-order abstraction converts historical records into structured summaries of recurring symptoms, interventions, outcomes, counts, and category distributions, forming a compact representation of prior maintenance experience.

Meter abstraction transforms low-frequency operational indicators into behavioral summaries rather than raw readings. Depending on meter semantics, this includes trend detection, sustained deviations from baseline, change points, reset identification, drift characterization, and anomaly patterns. For example, gauge-type indicators are summarized through baseline deviation and sustained-change detection, while cumulative indicators are analyzed through increments to detect resets, abnormal rates, and extended inactivity. These summaries capture usage and condition evolution while suppressing

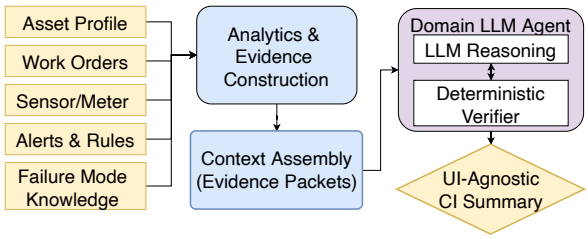

Figure 1: End-to-end pipeline of the Condition Insight System. Heterogeneous asset data and failure-mode knowledge are consolidated into structured evidence packets, which are processed by a domain LLM agent with verifier-based checking to produce a UI-agnostic condition insight summary.

noise. Additional implementation details are provided in Appendix A.

Failure knowledge alignment links work-order evidence to failure mechanisms through a distribution-aware semantic matching process based on an Unbalanced Optimal Transport (UOT) (Chizat et al., 2018) approach. Work-order descriptions $\mathbf{w}$ and failure mechanisms and degradation modes $\mathbf{m}$ are embedded into a shared vector space, and the optimal matching between the *distributions* of $\mathbf{w}$ and $\mathbf{m}$ is computed as follows:

$$\min_{\mathbf{T}:\mathbf{T}\geq 0} \mathbf{C}\cdot\mathbf{T} - \varepsilon H(\mathbf{T}) + \rho_1 \mathrm{KL}(\mathbf{T}\mathbf{1}, \mathbf{w}) + \rho_2 \mathrm{KL}(\mathbf{T}^T\mathbf{1}, \mathbf{m}), \quad (1)$$

where $\mathbf{C}$ is a squared Euclidean cost on the latent space, $H(\cdot)$ is the Shannon entropy, $\mathrm{KL}(\cdot)$ are Kullback-Leibler divergences, and $\mathbf{T}$ is the optimal matching which defines the semantic correspondence. This process constructs a structured hypothesis space connecting observed behavior to engineering-grounded failure semantics.

All processing in this stage is deterministic and reproducible, producing auditable evidence that defines the factual basis for downstream LLM reasoning.

4.3 Domain LLM Agent and Deterministic Verification

The Domain LLM Agent consumes the structured evidence packet and produces a Condition Insight Summary consisting of evidence-backed insights and recommendations. The LLM operates under domain-specific constraints that restrict explanations to those supported by evidence and consistent with failure semantics. For ablation, we compare this Constrained configuration with a Naive prompt that requests a general condition summary over the same structured evidence packet but does not encode explicit rule-aligned condition criteria or failure-mode constraints. The deterministic verification loop remains active in both prompt settings, ensuring identical governance checks while isolating the effect of prompt-level constraints over the same structured evidence packet. Full details on the prompt and structured evidence packet is given in Appendix B.

In production deployment, key decision elements are governed through a deterministic verification loop. The LLM is prompted to assign an overall condition category (e.g., *Normal*, *Needs Attention*, *Not Enough Data*) based on maintenance signals such as open or delayed work orders, emergency interventions, alerts, and meter anomalies. These same criteria are implemented as explicit rules in a parallel deterministic pipeline. The system compares the LLM-generated condition with the rule-based outcome, using discrepancies to identify reasoning errors and refine prompts.

This generation–verification separation ensures that high-level categorization remains governed by explicit operational criteria, while the LLM contributes explanatory synthesis over heterogeneous evidence.

5 Case Studies

We evaluated Condition Insight in two deployment settings to assess whether the verification-first reasoning architecture operates reliably under production data constraints. These studies examine evidence grounding, behavior under uncertainty, and interaction with practitioner workflows rather than model capability in isolation.

5.1 Case Study A: Product Integration

The system was deployed as an early-access decision-support capability within an enterprise CMMS, operating solely on production data available to practitioners (work orders, maintenance status, asset metadata, and condition indicators). As typical in such environments, evidence was incomplete and uneven.

The system produced structured condition summaries linking operational disruptions, maintenance balance, execution delays, and diagnostic coverage gaps to explicit evidence. All outputs passed through deterministic verification checks, and unsupported conclusions were revised or suppressed. Recommendations remained advisory and

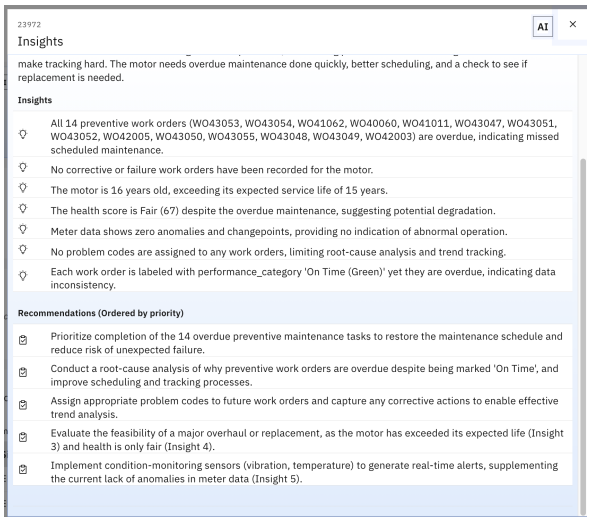

Figure 2: Condition insight generated by the proposed framework using historical work orders of the last 200 days for a motor asset. The output separates evidence-based observations (top) from prioritized, actionable recommendations (bottom), synthesizing maintenance history, asset metadata, health indicators, and data consistency checks into a decision-oriented summary.

aligned with existing workflows; no automated actions were triggered. The system maintained consistent insight structure across asset types while surfacing uncertainty and data limitations as explicit outputs.

Figure 2 shows an example output. The system produces a structured, UI-agnostic summary consisting of evidence-based insights and prioritized recommendations, linking claims to operational signals and failure semantics.

5.2 Case Study B: Enterprise-Scale Deployment

The framework was evaluated across $\sim$1,500 assets spanning 16 asset classes and 14 sites, reflecting heterogeneous sensor coverage, maintenance histories, and coding practices. Assets were categorized as *Not Enough Data*, *Normal*, or *Needs Attention* based on convergence of operational indicators, maintenance history, and failure semantics.

A substantial portion of assets fell into *Not Enough Data*, demonstrating conservative behavior under sparse evidence. This conservative classification behavior is reflected quantitatively in the Mean Insight Count (MIC) trend reported in Section 6. Category distributions were stable across sites, suggesting thresholds did not overfit local conditions. For assets flagged *Needs Attention*, insights reflected recurring, interpretable failure mechanisms aligned with asset class and history. The deterministic verification loop prevented unsupported conclusions and ensured risks were tied to explicit evidence.

In practice, the system supported prioritization rather than automation, narrowing the subset of assets requiring review. Practitioner overrides mainly reflected local knowledge or pending operational changes, reinforcing the role of human oversight.

5.3 Operational Impact and Scalability

In current enterprise workflows, condition-based analysis typically requires manual retrieval and reconciliation of heterogeneous data across multiple systems. Discussions with practitioners indicate that reviewing a single asset may require approximately 5 minutes in the CMMS to retrieve maintenance records, 10 minutes in a building management system to access sensor and SCADA data, and an additional 10–15 minutes in external analytics or IoT platforms to inspect alerts, trends, and metadata. This results in roughly 20–30 minutes of effort per asset.

In contrast, the proposed framework generates integrated condition insights and recommendations in approximately 15–30 seconds per asset. While human validation remains essential, the automation of evidence integration and structured reasoning substantially reduces analyst overhead.

Under existing processes, only a small fraction of assets can be reviewed regularly, limiting practical adoption of condition-based maintenance at scale. By reducing per-asset analysis time from tens of minutes to seconds, the framework makes broader, systematic coverage operationally feasible.

6 Evaluation

We evaluate system behavior using two complementary mechanisms: (1) deterministic validation that assesses rule-following performance, and (2) structured judge audits that quantify how generated insights relate to available evidence and governance constraints.

We tested multiple reasoning backbones (Mistral-Medium, LLaMA-4-Maverick, Granite, and GPT-OSS) and multiple frontier models as evaluators (GPT-4.1 and Claude Sonnet 4). Unless otherwise stated, results reported below use GPT-OSS as the reasoning backbone and Claude Sonnet 4 as the primary (strict) judge, as this

pairing produced representative and conservative audit signals. Metrics are computed per asset snapshot, summarized in this section, and formally defined in Appendix C.

Grounding. Unsupported Claim Rate (UCR) measures the proportion of statements not supported by structured evidence.

Diagnostic usefulness. High Specificity Rate (HSR) measures how often statements reference concrete components, failure modes, or maintenance-relevant details.

Reasoning stability. Contradiction Rate (CR) and Redundancy Rate (RR) capture internal inconsistencies and repeated issue descriptions.

Coverage and caution. Mean Insight Count (MIC) reflects output verbosity and responsiveness to available evidence.

Rule compliance. Condition Agreement Rate (CAR) measures alignment between the model's overall condition classification and a deterministic rule-based condition-selection specification.

6.1 Effect of Evidence Construction, Enrichment, and Prompt Constraints

To isolate the role of deterministic evidence construction, we include a baseline in which heterogeneous maintenance inputs are passed directly to the LLM (RawData), bypassing the deterministic evidence construction stage. This is compared against the Naive configuration operating over the full structured evidence packet (Naive + All), keeping the prompt identical.

Effect of deterministic evidence construction. Bypassing deterministic evidence construction sharply degrades grounding and specificity. Compared with Naive + All, RawData increases Unsupported Claim Rate (UCR) from 0.003 to 0.214 and reduces High Specificity Rate (HSR) from 0.66 to 0.20, while Condition Agreement Rate (CAR) remains similar. This indicates that without structured evidence, the model produces less grounded and less diagnostically useful outputs despite access to the same underlying information. This demonstrates that deterministic evidence construction is a core architectural component rather than a presentation layer over raw inputs.

Table 1 summarizes the effects of three architectural factors: raw versus structured inputs, evidence scope (work orders + asset profile vs. full structured evidence), and prompt configuration (Naive vs. Constrained). The Naive prompt (see Appendix B.2) requests an unconstrained summary of asset condition, whereas the Constrained prompt explicitly encodes rule-aligned condition criteria and failure-semantic constraints consistent with the system's deterministic governance logic. We next examine how evidence scope and prompt-level constraints interact when operating over structured evidence.

Prompt	Scope	UCR↓	HSR↑	CAR↑	MIC
Naive	RawData	0.214	0.20	0.68	5.0
Naive	WO	0.007	0.64	0.68	4.9
Naive	All	0.003	0.66	0.70	4.9
Constr.	WO	0.002	0.68	0.89	3.2
Constr.	All	0.008	0.71	0.91	3.3

Table 1: Effect of raw inputs, structured evidence construction, and prompt constraints. RawData feeds heterogeneous maintenance inputs directly to the LLM, bypassing deterministic evidence construction.

Effect of evidence enrichment. Under naive prompting, expanding evidence to include meter abstractions and structured failure-mode knowledge reduces Unsupported Claim Rate (UCR) from 0.007 to 0.003 and slightly increases specificity (HSR: 0.64 → 0.66). This indicates that structured operational and engineering context constrains free-form reasoning and reduces unsupported statements.

Effect of prompt constraints. The Constrained prompt substantially improves rule compliance across both evidence scopes. Condition Agreement Rate (CAR) increases from 0.68 to 0.89 under limited data and from 0.70 to 0.91 under full evidence. This demonstrates that explicit rule-aligned prompting materially strengthens adherence to deterministic governance logic.

The Constrained prompt also reduces verbosity (MIC: 4.9 → 3.2), reflecting more selective reasoning. While UCR increases slightly under full evidence with the Constrained prompt (0.003 → 0.008), grounding remains high overall and outputs remain internally stable.

Reasoning stability. Contradiction (CR) and redundancy rates (RR) remain near zero across all configurations, indicating that trajectory-controlled reasoning combined with deterministic verification prevents internally conflicting or repetitive issue generation.

Variation Across Reasoning Models While absolute performance varies, directional conclusions hold across all tested backbones: for Mistral-

Medium, LLaMA-4-Maverick, and GPT-OSS, structured evidence consistently reduces UCR relative to RawData, and constrained prompting consistently improves CAR relative to naive prompting, confirming the effect is architectural rather than backbone-specific.

Evaluation Stability Across Judges Audits performed with an alternative evaluator (GPT-4.1) yield higher absolute grounding and specificity scores but preserve directional trends. This indicates that observed effects reflect architectural constraints rather than evaluator idiosyncrasies. Deterministic Condition Agreement Rate (CAR), computed independently of LLM judges, provides a stable governance signal across evaluators.

6.2 Operational role and impact

The framework is designed to support prioritization in enterprise maintenance workflows, where condition assessment typically requires manual retrieval and reconciliation of heterogeneous evidence across multiple systems. Practitioner estimates indicate that this process requires approximately 20–30 minutes per asset. In contrast, the proposed system produces integrated, evidence-grounded condition summaries in 15–30 seconds.

While outputs remain advisory and subject to human validation, the combination of improved grounding (low UCR) and rule compliance (high CAR) reduces the effort required to synthesize maintenance evidence and increases trust in generated insights. This enables broader and more systematic asset coverage, which is typically infeasible under manual review constraints.

6.3 Overall Observations

Results indicate that system behavior is primarily governed by evidence structure and prompt-level constraints rather than backbone model variation. Structured engineering and operational context reduces unsupported reasoning under naive prompting, while constraint-aware prompting dramatically improves rule adherence and reduces verbosity without introducing instability.

The combination of structured evidence construction and deterministic governance therefore provides a controllable reasoning envelope for industrial decision support.

These results indicate that performance is primarily governed by evidence structuring and governance constraints rather than backbone model choice alone. In particular, deterministic evidence construction defines the quality of the reasoning substrate, while constrained prompting and verification ensure adherence to operational criteria. This separation provides a controllable and auditable reasoning framework for industrial decision support.

7 Conclusion and Lessons Learned

Deploying Conditional Insight in enterprise maintenance environments highlights several principles for operationalizing LLM-based reasoning under real-world constraints.

- **Conditional explanations improve trust.** Practitioners preferred structured reasoning framed around "why" and "what next" over raw alerts, aligning outputs with maintenance decision workflows.
- **Signal semantics must guide abstraction.** Cumulative, incremental, and categorical indicators require distinct summarization strategies; uniform treatment leads to unstable reasoning.
- **Failure semantics provide guardrails.** FMEA-derived component and mechanism constraints reduce implausible hypotheses and bound the space of admissible explanations.
- **LLMs should reason over curated evidence.** Operating on structured summaries rather than raw telemetry improves robustness, interpretability, and cross-site consistency.
- **Deterministic governance enables production deployment.** Separating generative synthesis from rule-based verification ensures reproducibility, auditability, and operational acceptance.
- **Prompt design encodes policy.** Prompt structure functions as system configuration, requiring iterative refinement aligned with deterministic checks and practitioner feedback.

Together, these findings suggest that reliable industrial LLM systems depend more on structured evidence and explicit governance constraints than on backbone model choice alone.

References

Deepak Bhaskar Acharya, Karthigeyan Kuppan, and B Divya. 2025. Agentic ai: Autonomous intelligence

for complex goals–a comprehensive survey. *IEEe Access*.

Imran Ahmed, Gwanggil Jeon, and Francesco Piccialli. 2022. From artificial intelligence to explainable artificial intelligence in industry 4.0: a survey on what, how, and where. *IEEE transactions on industrial informatics*, 18(8):5031–5042.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. 2019. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024.

Lenaic Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. 2018. Scaling algorithms for unbalanced optimal transport problems. *Mathematics of computation*, 87(314):2563–2609.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*.

Soodeh Hosseini and Hossein Seilani. 2025. The role of agentic ai in shaping a smart future: A systematic review. *Array*, page 100399.

Abigail Langbridge, Fearghal O'Donncha, Amadou Ba, Fabio Lorenzi, Christopher Lohse, and Joern Ploennigs. 2023. Causal temporal graph convolutional neural networks (ctgcn). *arXiv preprint arXiv:2303.09634*.

Markus Langer and Richard N Landers. 2021. The future of artificial intelligence at work: A review on effects of decision automation and augmentation on workers targeted by algorithms and third-party observers. *Computers in Human Behavior*, 123:106878.

Yaguo Lei, Naipeng Li, Liang Guo, Ningbo Li, Tao Yan, and Jing Lin. 2018. Machinery health prognostics: A systematic review from data acquisition to rul prediction. *Mechanical systems and signal processing*, 104:799–834.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Yuqian Lu, Chao Liu, I Kevin, Kai Wang, Huiyue Huang, and Xun Xu. 2020. Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and computer-integrated manufacturing*, 61:101837.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Sajad Moosavi, Maryam Farajzadeh-Zanjani, Roozbeh Razavi-Far, Vasile Palade, and Mehrdad Saif. 2024. Explainable ai in manufacturing and industrial cyber–physical systems: A survey. *Electronics*, 13(17):3497.

Dhaval Patel, Shuxin Lin, James Rayfield, Nianjun Zhou, Roman Vaculin, Natalia Martinez, Fearghal O'donncha, and Jayant Kalagnanam. 2025. Assetopsbench: Benchmarking ai agents for task automation in industrial asset operations and maintenance. *arXiv preprint arXiv:2506.03828*.

Humberto Nuno Teixeira, Isabel Lopes, and Ana Cristina Braga. 2020. Condition-based maintenance implementation: a literature review. *Procedia Manufacturing*, 51:228–235.

Alexander Timms, Abigail Langbridge, and Fearghal O'Donncha. 2024. Agentic anomaly detection for shipping. In *NeurIPS 2024 Workshop on Open-World Agents*.

Dakuo Wang, Elizabeth Churchill, Pattie Maes, Xiangmin Fan, Ben Shneiderman, Yuanchun Shi, and Qianying Wang. 2020. From human-human collaboration to human-ai collaboration: Designing ai systems that can work together with people. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*, pages 1–6.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Deterministic Meter Abstraction in Conditional Insight

A.1 Upstream Aggregation Model

Enterprise asset management systems record operational indicators ("meters") that summarize physical measurements over time. Unlike raw high-frequency telemetry (e.g., SCADA streams), meter readings stored in CMMS platforms typically represent aggregated or accumulated quantities at curated intervals.

Let $x(t)$ denote an underlying high-frequency signal. A recorded meter value v_i at time t_i may be expressed as:

$$v_i = \mathcal{A}\big(x(t)\big), \qquad t \in [t_{i-1}, t_i],$$

where $\mathcal{A}$ denotes an upstream aggregation operator (e.g., averaging, integration, or accumulation) applied by the operational system. The Conditional Insight (CI) framework operates exclusively on these aggregated indicators and does not ingest raw telemetry.

A.2 Deterministic Behavioral Abstraction

Given a meter time series

$$\{(t_i, v_i)\}_{i=1}^{N},$$

CI performs deterministic statistical abstraction to convert meter histories into compact, interpretable behavioral summaries. These summaries constitute auditable evidence and are incorporated into the structured `asset_facts` packet.

GAUGE Meters. For point-in-time measurements (e.g., temperature, pressure), summary statistics are computed:

$$\bar{v} = \frac{1}{N}\sum_{i=1}^{N} v_i, \qquad s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(v_i - \bar{v})^2}.$$

Anomalies are identified via Z-score deviations:

$$z_i = \frac{v_i - \bar{v}}{s}, \qquad |z_i| > z_{\text{thresh}},$$

and abrupt changes:

$$\Delta_i = v_i - v_{i-1}.$$

Minor oscillations are suppressed using deviation-from-baseline criteria to ensure only sustained or practically meaningful shifts are retained.

CONTINUOUS Meters. For monotonically increasing totals (e.g., run-hours), increments are computed:

$$\Delta_i = v_i - v_{i-1}.$$

Increment statistics define a normal band:

$$[\bar{\Delta} - ks_{\Delta}, \bar{\Delta} + ks_{\Delta}],$$

where $\bar{\Delta}$ and s_{Δ} are the mean and standard deviation of increments.

Event categories include:

- Reset events ($\Delta_i < 0$),
- Rate anomalies (outside increment band),
- Extended flat periods ($|\Delta_i| \leq \varepsilon$).

A.3 Structured Evidence Integration

The output of deterministic abstraction is recorded as `meter_facts` $= \{\text{metadata, behavioral summaries, detected events}\}$, which forms part of the structured `asset_facts` evidence packet.

All computations are deterministic and reproducible, and no raw time-series values are provided to the LLM. Meter-derived summaries function strictly as bounded, traceable evidence within the verification-first architecture.

B LLM Prompt Design and Guideline

B.1 Condition Insight Prompt

The Condition Insight agent's system prompt defines the agent's role as an evidence-grounded maintenance analyst, specifies the reasoning tasks over the structured evidence packet (`asset_facts`), and instructs the model to confine its inferences to the provided evidence. It prescribes strict language and formatting rules (e.g., numeric digits only, expanded terminology, descriptive problem-code text) and enforces a fixed JSON output schema with the following fields: *Overall Condition*, *Overall Condition Explanation*, *Key insights*, *Recommendations*, and *Overall confidence* (value and reasoning).

The prompt encodes explicit rule sets for: (i) overall-condition labels and selection criteria; (ii) key-insight generation (grounding, component grouping, ordering, and limits); (iii) recommendation formulation (specific component-level actions using failure-mode knowledge); and (iv) a

deterministic, majority-rule method for overall confidence. It also provides guidance for interpreting meter patterns to inform insights and recommendations.

At runtime, the prompt incorporates the `asset_facts` block which is a structured JSON object produced by the Analytics & Evidence Construction unit (Fig. 1). This evidence packet consolidates information from sensor and meter data, alerts and rule evaluations, work-order history, and failure-mode knowledge, as described in Section 4, and serves as the sole factual basis for the agent's reasoning. A simplified JSON schema for this structure is provided in Listing 1. An optional `feedback` block may also be included in the prompt to convey refinement signals during iterative workflows.

Listing 1: `asset_facts` block schema.

```
"asset_facts": {
    "asset_details_facts": {
    asset_number, description, site_ID,
     priority, status, is_running, failure_code,
     asset_age_in_years, manufacturer
    },
"workorder_facts": { counts, distributions,
     preventive_workorders: [...],
     corrective_and_other_workorders: [...]
    },
"meter_facts": [
      { meter metadata + meter_values[...] }
    ],
"alert_facts": [{ alert metadata... }],
"fmea_facts": [
      { component, mechanism, actions... }
    ],
"health_scores": {
      score_name: { value, range, meaning }
}
```

B.2 Naive Prompt

In our evaluation results (Section 6), we include a naive baseline prompt, shown in Listing 2, that reflects an unconstrained application of an LLM to the Condition Insight task. This baseline preserves only the JSON output schema required for LLM-judge evaluation and omits all grounding requirements, verification rules, and domain-specific constraints used in the deployed CI agent. As a result, the LLM model is free to interpret and reason over the `asset_facts` in any manner. This minimal configuration serves as a lower-bound reference point in our comparative analysis.

B.3 LLM Judge Evaluation Prompt

The LLM judge provides a structured, rule-based assessment of generated insights and recommendations. The LLM evaluation judge is implemented through a two-part prompt consisting of a system prompt (Listing 3) and a user prompt (Listing 4). The system prompt defines the judge's role as a strict evaluator, specifying grounding constraints, output-format requirements, and prohibiting the model from generating any new content beyond the evaluation.

The user prompt provides the full evaluation schema and embeds the following information:

- `CONDITION_INSIGHTS`: the generated condition-insight summary produced by the Condition Insight agent and subject to evaluation.
- `ASSET-FACTS`: the evidence packet provided by the Analytics & Evidence Construction module to the Condition Insight agent and denoted as **{asset_facts}**.
- `AGENT-SPECIFICATIONS` (denoted also as **{agent_instructions}**): the instructions used by the Condition Insight agent's system prompt to generate the condition-insight summary.

The prompt also enumerates the scoring rules for the evaluation of each insight and recommendation. All per-item metrics are rated on a 1–3 scale:

- **Factuality**: evaluates whether each statement is directly supported by the provided `asset_facts`.
- **Coherence**: assesses internal logical consistency and the absence of contradictions across items.
- **Relevance**: measures the usefulness of each statement for assessing asset condition and maintenance risk.
- **Repetitiveness**: quantifies redundancy relative to other insights or recommendations.
- **Specificity**: evaluates how concrete, detailed, and data-anchored each statement is.

In addition, the judge assigns two global evaluation metrics:

- **Overall-condition validity** (boolean): determines whether the agent's overall condition classification is evidence-supported and compliant with its specification.

- **Completeness** (two global scores: one for insights and one for recommendations): measures the number of generated items relative to the expected count.

For all scores, the judge is asked to provide a justification that anchors the assigned rating. Taken together, the system and user prompts define the evaluation procedure and promote assessments that are carried out in a controlled and reproducible manner. See the detailed descriptions in Listing 4. Appendix C provides a detailed description on how these scores are aggregated across the multiple experiments to generate the evaluations metrics presented in Section 6.

C Structured Evaluation Metrics

We evaluate system behavior using structured judge audits that quantify how generated insights and recommendations relate to available evidence and deterministic rules. Audits are performed using two independent large language models acting strictly as evaluators (Claude-4 Sonnet and GPT-4.1), each operating under a fixed evaluation prompt and scoring rubric described in Appendix B.3. Evaluation metrics are computed directly from the judge outputs, measured per asset snapshot, and then aggregated across evidence scope, reasoning model, and judge model.

C.1 Grounding Metrics

Unsupported Claim Rate (UCR) measures the fraction of statements (insights + recommendations) not supported by structured evidence.

$$\text{UCR} = \frac{\#\text{unsupported statements}}{\#\text{total statements}}$$

Statements are considered unsupported if no corresponding evidence exists in the structured asset context (e.g., meter summaries, work-order abstractions, or failure mappings). When using the LLM judge (Appendix B.3), a statement is counted as *unsupported* specifically when the judge assigns it a *Factuality score of 1*.

C.2 Diagnostic Usefulness

High Specificity Rate (HSR) measures the proportion of statements that reference concrete components, failure mechanisms, or maintenance-relevant details rather than generic observations.

$$\text{HSR} = \frac{\#\text{specific, asset-relevant statements}}{\#\text{total statements}}$$

HSR reflects whether outputs provide actionable, technically meaningful insight. When using the LLM judge (Appendix B.3), a statement is counted as *specific* when the judge assigns it a *Specificity score of 3*.

C.3 Reasoning Stability

Contradiction Rate (CR) measures the fraction of outputs containing internally inconsistent statements.

$$\text{CR} = \frac{\#\text{outputs with contradictions}}{\#\text{evaluated outputs}}$$

Redundancy Rate (RR) captures repeated or overlapping issue descriptions within a single output.

$$\text{RR} = \frac{\#\text{redundant statements}}{\#\text{total statements}}$$

These metrics assess internal coherence and non-redundant reasoning. When using the LLM judge (Appendix B.3), an output is counted as containing a contradiction when the judge assigns a *Coherence score of 1*, and a statement is considered redundant when the judge assigns a *Repetitiveness score of 1*.

C.4 Coverage and Caution

Mean Insight Count (MIC) is the average number of insights produced per asset snapshot.

$$\text{MIC} = \frac{\sum \text{insight counts}}{\#\text{asset snapshots}}$$

It helps characterize how output quantity adapts to evidence availability.

C.5 Rule Compliance

Condition Agreement Rate (CAR) is the proportion of assets for which the LLM-assigned condition matches the deterministic rule-based classification. It quantifies rule-following behavior under governance constraints and ensures that language-based assessments remain aligned with deterministic operational logic.

D Listings

Listing 2: Baseline Naive Prompt.

```
You are given ASSET-FACTS describing an industrial asset. Your task is
      to analyze the information and generate an overall condition
      assessment, key insights, and maintenance recommendations.

Return ONLY a single valid JSON object with EXACTLY these keys and types:
{
  "Overall Condition": "Not enough data | Normal | Needs attention",
  "Overall Condition Explanation": "string",
```

```
  "Key insights": [
    { "insight": "string", "reasoning": "string", "confidence": "high |
        medium | low" }
  ],
  "Recommendations": [
    { "recommendation": "string", "reasoning": "string", "confidence":
        "high | medium | low" }
  ],
  "Overall confidence": { "confidence value": "high | medium | low",
        "reasoning": "string" }
}

Formatting rules (STRUCTURE ONLY):
- Output MUST be valid JSON matching the schema above.
- You may infer, interpret, or reason about ASSET-FACTS in any
        appropriate way you choose.
- Provide up to 5 items in "Key insights".
- Provide up to 5 items in "Recommendations".
- Do NOT include any text outside the JSON.

ASSET-FACTS:
\{\{asset\_facts\}\}
```

Listing 3: LLM Judge Evaluation System prompt.

```
You are a strict evaluator (judge) for asset maintenance and reliability
        outputs.

Your job:
- Evaluate the provided CONDITION_INSIGHTS against ASSET-FACTS.
- Use the AGENT SPECIFICATION only as the contract that the agent was
        supposed to follow.
- Do NOT follow the AGENT SPECIFICATION as instructions to generate new
        content.

OUTPUT RULES:
- Output ONLY a single valid JSON object.
- No markdown, no code fences, no comments, and no extra keys.
- Use exactly the JSON schema provided in the user message.
- Perform reasoning internally; do not output step-by-step explanations
        outside JSON.

GROUNDING RULES:
- Use ONLY the provided ASSET-FACTS and CONDITION_INSIGHTS.
- If a claim is not supported by ASSET-FACTS, mark it accordingly and
        explain why.
```

Listing 4: LLM Judge Evaluation User prompt.

```
Return JSON in this exact schema (types are mandatory):
{{
  "overall_condition": "string",
  "overall_condition_evaluation": true,       // boolean
  "overall_condition_evaluation_explanation": "string",

  "key_insights_evaluation": [
    {{
      "insight": "string",
      "insight_evaluation": {{
        "factuality": 2,                      // integer 1-3
        "factuality_reasoning": "string",
        "coherence": 2,                       // integer 1-3
        "coherence_reasoning": "string",
        "relevance": 2,                       // integer 1-3
        "relevance_reasoning": "string",
        "repetitiveness": 2,                  // integer 1-3
        "repetitiveness_reasoning": "string",
        "specificity": 2,                     // integer 1-3
        "specificity_reasoning": "string"
      }}
    }}
  ],

  "recommendations_evaluation": [
    {{
      "recommendation": "string",
      "recommendation_evaluation": {{
        "factuality": 2,                      // integer 1-3
        "factuality_reasoning": "string",
        "coherence": 2,                       // integer 1-3
        "coherence_reasoning": "string",
        "relevance": 2,                       // integer 1-3
        "relevance_reasoning": "string",
        "repetitiveness": 2,                  // integer 1-3
        "repetitiveness_reasoning": "string",
        "specificity": 2,                     // integer 1-3
        "specificity_reasoning": "string"
      }}
    }}
  ],

  "recommendations_completeness": 2,           // integer 1-3
  "recommendations_completeness_reasoning": "string",

  "insights_completeness": 2,                  // integer 1-3
  "insights_completeness_reasoning": "string"
}}

RATING SCALE (applies to all 1-3 fields):
- 1 = not satisfied
- 2 = partially satisfied
- 3 = fully satisfied

RULES FOR OVERALL CONDITION EVALUATION:
- "overall_condition" MUST be copied verbatim from the agent's
        CONDITION_INSIGHTS output.
- "overall_condition_evaluation" MUST be true if and only if BOTH are
        satisfied:
  (1) The agent's overall condition is supported by ASSET-FACTS, AND
  (2) The agent's overall condition follows the "Rules for Overall
        Condition" in the AGENT SPECIFICATION.
- Otherwise, "overall_condition_evaluation" MUST be false.
- The explanation must cite specific ASSET-FACTS evidence (e.g.,
        corrective work orders count, overdue work orders, anomalies,
        alert totals, health score ranges).

RULES FOR FACTUALITY (applies to each insight or recommendation
        individually):
Rate factuality on a 1-3 scale based ONLY on how well the statement is
        grounded in ASSET-FACTS.
- Score 1 (not grounded): The statement is unsupported or contradicted
        by ASSET-FACTS.
- Score 2 (partially grounded): Some parts are supported by ASSET-FACTS,
        but other parts extend beyond the data, are ambiguous, or weakly
        justified.
- Score 3 (fully grounded): The statement is completely and directly
        supported by ASSET-FACTS.
For "factuality_reasoning": Explain which ASSET-FACTS elements support
        or fail to support the statement.

RULES FOR COHERENCE (applies to each insight or recommendation
        individually):
Rate coherence on a 1-3 scale based on whether the statement is
        internally consistent and non-contradictory with other insights
        or recommendations.
- Score 1 (contradicting): The statement contradicts itself or
        contradicts other insights or recommendations.
- Score 2 (partially coherent): Mostly consistent, but contains unclear
        logic, minor contradictions, or ambiguous connections.
- Score 3 (fully coherent): Completely consistent with itself and with
        all other insights and recommendations; no contradictions or
        logical conflicts.
For "coherence_reasoning": Explain how the score was derived, citing
        specific inconsistencies or confirming consistency.

RULES FOR RELEVANCE (applies to each insight or recommendation
        individually):
Rate relevance on a 1-3 scale based on how meaningful and useful the
        statement is for asset maintenance and evaluating Overall
        Condition.
- Score 1 (irrelevant): The statement does not meaningfully affect asset
        condition, risk, failures, or maintenance decisions; may be
        trivial or routine (e.g., stating something works fine without
        impact).
- Score 2 (somewhat relevant): The statement relates to meaningful
        topics but is generic, incomplete, or weakly connected to
        maintenance priorities or the rules.
- Score 3 (highly relevant): The statement directly addresses asset
        risks, failures, condition drivers, or actionable maintenance
        considerations, and strongly aligns with the rules.
For "relevance_reasoning": Explain why the statement is or is not
        meaningful for maintenance decisions and Overall Condition.

RULES FOR REPETITIVENESS (applies to each insight or recommendation
        individually):
Rate repetitiveness on a 1-3 scale based on how much the statement
        repeats content already provided in other insights or
        recommendations.
- Score 1 (very repetitive): The statement repeats the same idea, issue,
        component, evidence, or conclusion as another item with little or
        no new information.
- Score 2 (somewhat repetitive): The statement overlaps partially with
        other items but introduces some new detail or perspective.
- Score 3 (non-repetitive): The statement is distinct and provides new
        information, a new issue, or a different actionable point without
        repeating earlier content.
For "repetitiveness_reasoning": Explain how the score was derived,
        noting which other items (if any) it repeats or diverges from.

RULES FOR SPECIFICITY (applies to each insight or recommendation
        individually):
Rate specificity on a 1-3 scale based on how concrete, detailed, and
        data-anchored the statement is with respect to ASSET-FACTS.
- Score 1 (very vague):
  - The statement is generic or high-level.
  - It does not reference specific components, FMEA items, failure
        modes, work orders, alerts, anomalies, meter patterns, or
        concrete evidence.
  - Mentions of broad trends or asset attributes without operational
        detail count as vague.
- Score 2 (somewhat specific):
  - The statement includes some concrete details (e.g., component type,
        asset attributes, or general issue), but lacks clear links to
        specific FMEA elements, failure modes, actions, or specific
        ASSET-FACTS data points.
  - Partially informative but still broad or incomplete.
- Score 3 (fully specific):
  - The statement makes clear, explicit references to relevant
        components, failure modes, FMEA actions, work orders, alerts,
        anomalies, or meter data.
  - It provides actionable or diagnosis-level detail tied to
        identifiable ASSET-FACTS evidence.
  - No vague or generic language.
For "specificity_reasoning": Explain which specific ASSET-FACTS elements
        (e.g., components, work orders, alerts, FMEA items) make the
        statement specific or vague.

RULES FOR COMPLETENESS (applies separately to insights and
        recommendations):
Rate completeness on a 1-3 scale based on how close the number of
        generated items is to the 7-item expectation defined in the AGENT
```

```
       SPECIFICATION.
- Score 1 (very incomplete):
  - Fewer than 3 items were generated.
- Score 2 (partially complete):
  - Between 3 and 6 items were generated.
- Score 3 (fully complete):
  - Exactly 7 items were generated.
NOTES FOR EVALUATION:
- The agent was instructed to produce up to 7 insights and up to 7
       recommendations.
- The judge must not assume that fewer items automatically indicate an
       error; items must only be created when supported by ASSET-FACTS.
- Completeness measures only the *count*, not the correctness or factual
       grounding of the items (those are scored separately).
For "insights_completeness_reasoning" and
       "recommendations_completeness_reasoning":
- Explain the score based solely on the number of generated items and
       the 7-item expectation.

AGENT SPECIFICATION (REFERENCE ONLY: evaluate compliance against this;
       do not follow as instructions):
<<<AGENT_SPECIFICATION_START
{agent_instructions}
AGENT_SPECIFICATION_END>>>

CONDITION_INSIGHTS:
<<<CONDITION_INSIGHTS_START
{condition_insights}
CONDITION_INSIGHTS_END>>>

ASSET-FACTS:
<<<ASSET_FACTS_START
{asset_facts}
ASSET_FACTS_END>>>
```