

Plant Memory: Technical Language Processing and a Verified Maintenance Ontology for Agent-Accessible Failure Knowledge from SAP PM Work Orders

Akash Raj

M.Tech, School of Artificial Intelligence and Data Science

Indian Institute of Technology Jodhpur

m25ai1116@iitj.ac.in

ORCID: 0009-0005-5666-3967

October 2026

Abstract

Maintenance work orders are the largest written record of how industrial equipment fails, yet in most plants they are close to unusable for analysis. Short texts are abbreviated and misspelt, asset identity is buried in free text, work-type labels contradict the work described, and the fields that matter most for reliability (symptom, root cause, action taken) are usually empty. As a result, diagnostic knowledge stays with senior technicians and leaves when they do. We present *Plant Memory*, a system that applies technical language processing (TLP) to SAP Plant Maintenance (PM) exports and builds an ISO 14224-aligned maintenance ontology in which every extracted statement is a reified fact carrying a confidence score, the evidence it was read from, and a verification state. Extraction is rule-first, with an optional large language model (LLM) acting as a second reader whose disagreements raise a fact’s review priority rather than overwrite it. Technicians verify facts through a review queue, answer targeted knowledge-gap questions where failures repeat without a recorded cause, and approve or reject ontology drift proposals when new terms appear. The resulting graph is served to people through a web application and to AI agents through a Model Context Protocol (MCP) server, so both query the same verified knowledge and every answer cites the work orders behind it. On a synthetic 12-month export of 400 work orders from a parboiled-rice plant, rule-based extraction reaches 96.2% component accuracy, 88.9% failure-mode accuracy and 100% root-cause precision at 62% coverage; on 28 diagnostic questions, ranked root causes from the ontology are correct at top-1 in 82.1% of cases against 53.6% for the raw SAP root-cause column. We also report that an LLM used as an independent extractor was less accurate than the rules on every field, and describe the merge policy this led to. We discuss why these numbers are optimistic and what a real-plant evaluation requires.

1 Introduction

A typical computerised maintenance management system (CMMS) export records which parts were issued to which order and what they cost. It rarely records why the equipment failed or what actually fixed it. Where that information exists, it is spread across a 40-character short text (“Parb.- Dryer elevator brg chng”), an optional remarks field written in a mix of English and Hindi (“grease nahi tha, brg jam”), and occasionally a root-cause column filled in for a fraction of breakdowns. Reliability engineers who want to know which failure modes drive downtime, or which spares to stock, must read these records by hand.

Two developments make this worth revisiting. First, technical language processing has emerged as a discipline concerned with exactly this kind of text: short, domain-specific, abbreviation-heavy, and written for

colleagues rather than for machines [3, 4]. Second, AI assistants are starting to be placed in front of planners and engineers. An assistant can only answer from data it can read, and a raw Excel export is a poor substrate: it invites confident answers built on misread abbreviations, with no way for the reader to check them.

Plant Memory addresses both. Its contributions are:

1. **A provenance-first maintenance ontology.** Classes and relations are aligned with ISO 14224 [7] and extended for rice-processing equipment. Every extracted statement is reified as a fact with confidence, extraction method, evidence snippet, source field and a verification state (proposed, verified, rejected).
2. **A TLP pipeline tuned for SAP PM exports.** It merges material lines into events, expands plant abbreviations, resolves assets from functional locations or free-text aliases, and extracts component, symptom, failure mode, root cause and action under an explicit precedence order with calibrated confidences.
3. **An LLM-as-second-reader merge policy,** derived from a measured comparison in which the LLM underperformed the rules as an independent extractor but added value by filling gaps and flagging disagreements.
4. **A human verification loop** comprising a lowest-confidence-first review queue, automatically generated knowledge-gap questions for technicians, and ontology drift proposals that change the vocabulary only on approval. Verification decisions survive rebuilds.
5. **One knowledge core behind three interfaces:** a web application, a command-line interface and an MCP server [1]. The MCP server lets general-purpose AI agents query the verified graph and cite evidence, while its only write tool creates proposals.
6. **An evaluation harness** with a synthetic SAP PM generator that plants known patterns and hidden ground truth, scoring both extraction and end-to-end diagnosis.

2 Background and related work

Technical language processing. Brundage et al. [3] argue that general-purpose NLP pipelines fail on maintenance text because of its vocabulary, brevity and implicit context, and they propose TLP as a human-in-the-loop discipline that adapts NLP tools to technical domains. Dima et al. [4] discuss the adaptations this requires. Earlier work on cleaning work-order data for reliability analysis [6] and on hybrid human-machine tagging of maintenance logs [12] established that useful structure can be recovered with modest, carefully designed tooling and that expert review is central. MaintIE [2] provides a fine-grained annotation schema and benchmark for information extraction from maintenance short texts. Plant Memory follows the TLP view: rules encode plant-specific knowledge, and people remain responsible for what counts as true.

Maintenance ontologies. ISO 14224 [7] standardises equipment taxonomies, failure modes and failure mechanisms for reliability data. Reference ontologies for industrial maintenance, such as ROMAIN [8] and the maintenance-activity ontology of Woods et al. [14], formalise these concepts with richer axiomatisation. Our schema is deliberately lighter (an OWL 2 [13] vocabulary with reified facts) because its purpose is operational: to carry provenance and verification state for every statement extracted from a specific plant’s records.

LLMs, knowledge graphs and tool protocols. Retrieval-augmented generation [9] grounds model output in retrieved text; graph-based variants [5] and broader proposals for unifying LLMs with knowledge graphs [10] retrieve from structured knowledge instead. The Model Context Protocol [1] standardises how assistants

call external tools and read resources. Plant Memory uses MCP to expose reasoning tools over a verified graph rather than raw documents, so that an assistant’s answer can be traced to specific work orders and verification states.

3 Problem setting: what an SAP PM export looks like

The input is an SAP PM export in Excel or CSV with one row per material issue. A month of a real export from a rice processing unit motivated the design and exhibits the following issues, which the synthetic data reproduces:

- **Repeat failures are not linked.** One elevator accumulated three separate orders within a month (a bearing change, a sprocket change and “general maintenance”), and nothing connects them as one recurring problem.
- **Free text is not machine-readable.** Descriptions mix abbreviations and typos (“Parb.”, “brg”, “Plmbr Bkck”, “modificatn”, “ELlevator”), and asset names (“Hodi Elevator”, “Elev1&2”) are embedded in text with no equipment identifier.
- **Work-type labels are unreliable.** Bearing and belt replacements are tagged *Preventive* while comparable work is tagged *Corrective*, which skews any analysis built on the label.
- **Cause and fix are missing.** Cost, quantity and material are present; symptom, root cause, downtime and action taken mostly are not.
- **Records are fragmented.** One order may be posted on several dates, materials are split across lines, and reservation rows (SAP transaction MB21) carry a cost centre but no order number.

4 System design

4.1 Architecture

Figure 1 shows the data flow. Uploaded files are immutable evidence: people never edit raw rows, only the knowledge derived from them. The graph is rebuilt from all registered source files; when an order appears in several files, the newest file wins. Rebuilds preserve verification decisions for any fact whose extracted value is unchanged, and technician-stated facts are carried over untouched.

4.2 Ontology

Table 1 lists the core classes. Relations include *locatedIn* (asset to process area), *onAsset*, *involvesComponent*, *observedSymptom*, *hasFailureMode*, *hasRootCause* and *resolvedBy* (event to term), *consumed* (event to spare part), *realizes* (spare part to component type), *typicallyAffects* (failure mode to component) and *inCategory* (root cause to cause category).

Reification is the central design choice. A statement such as “order 30000230188 has failure mode *bearing wear*” is stored as a fact with: a confidence in $[0,1]$; the extraction method (structured, rules, inferred, llm, llm+rules, technician, with an +edited suffix after human correction); the source field and the exact words it was read from; and a status that moves from *proposed* to *verified* or *rejected*, with the verifier’s name and time. In the RDF export, verified facts are additionally asserted as direct triples, so plain SPARQL queries see only trusted knowledge while proposed facts remain available in reified form. In Neo4j, facts are relationship properties, and every node carries a :PM label so the system never touches other data in a shared database.

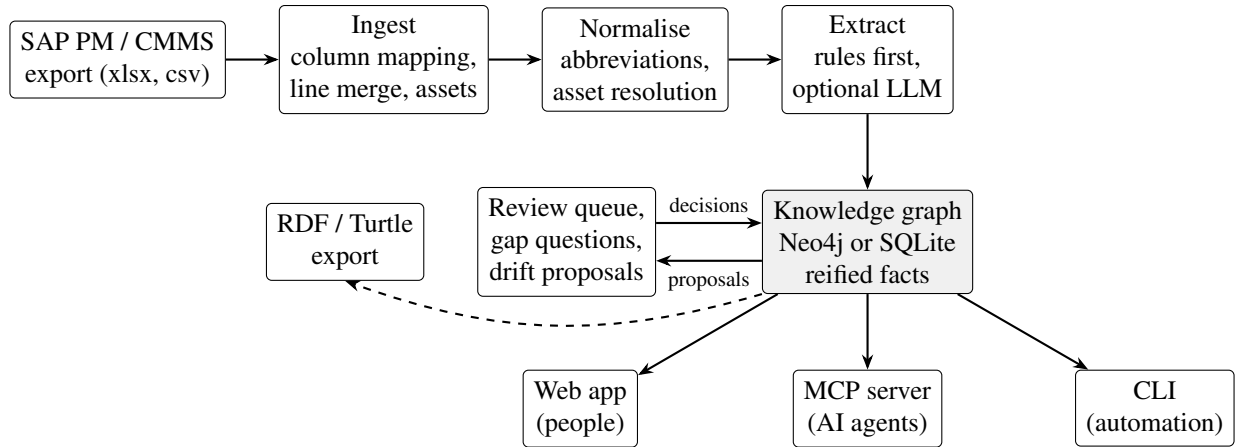


Figure 1: Plant Memory architecture. All three interfaces read and write the same graph through one core library, so a fact verified in the browser is reported as verified the next time an agent asks.

4.3 Technical language processing pipeline

Ingestion. Column names are matched to canonical fields by exact lookup, then fuzzy matching, and users can correct the mapping on screen; only order number, date and description are required. Date order (day-first or month-first) is inferred from the values. Rows are grouped by order: the modal description and work type are taken, header fields use the first non-empty value, and the earliest and latest posting dates are both kept. Rows without an order become consumption-only records. A data-quality flag is raised when a breakdown (from the breakdown column or from words such as “broken” in the text) carries the work type *Preventive*.

Normalisation and asset resolution. Text is lower-cased, letter–digit runs are split (“elev1” → “elev 1”), and a plant abbreviation table is applied (“parb” → parboiling, “brg” → bearing, “plmbr blk” → plumber block, “b/d” → breakdown). Assets are resolved in order of reliability: a functional location that matches the register (confidence 0.98); otherwise the longest alias match in the normalised text, disambiguated by process-area cues (0.85 with an area cue, 0.80 without, 0.50 if still ambiguous); otherwise a fuzzy alias match (0.45). An asset-discovery step proposes new register entries from unresolved functional locations and from residual noun phrases after component, action and symptom words are stripped, accounting for SAP’s 40-character truncation.

Rule-based extraction. Sources are consulted in a fixed precedence: structured SAP columns, then technician remarks, then the short text, then the materials issued. Specific rules encode domain knowledge:

- Values from structured columns that map to the vocabulary receive confidence 0.95; values that do not map become drift candidates.
- The asset’s own name and aliases are removed before component matching, so “Hodi belt conv” is not read as a conveyor-belt component.
- A component named in the text and confirmed by an issued material receives 0.92; named but unconfirmed, 0.75; read only from materials, 0.75, preferring the highest-cost line and ignoring consumables such as adapter sleeves when a primary part is present.
- Root causes are read only from the root-cause column or technician remarks, never from the short text, where words such as “stone” usually name a component.

Class	Role and examples
ProcessArea	Top of the asset hierarchy: parboiling, paddy-to-rice milling, utilities, packing, warehouse
Asset	Maintainable item with a functional location, class and shop-floor aliases (e.g. <i>Dryer elevator</i> , alias “LSU elev”)
Component	Part of an asset: bearing, V-belt, sprocket, plummer block, rubber roll, mechanical seal
SparePart	SAP material that realises a component type, with material code and unit cost
Symptom	Observation: abnormal noise, vibration, belt slip, tripping, reduced output
FailureMode	ISO 14224-aligned, with mechanism code: bearing wear (2.4 Wear), belt slippage (1.5 Looseness), seal leakage (1.1 Leakage)
RootCause	Misalignment, inadequate lubrication, dust ingress, overload, foreign material, cavitation; grouped into cause categories
Action	Replace, repair, weld repair, realign, lubricate, adjust, refurbish, modify
MaintenanceEvent	One work order: dates, work type, cost, downtime, breakdown flag
Fact	Reified statement (subject, predicate, object) with confidence, method, status, verifier and evidence
Evidence	A source row and field, or a technician statement

Table 1: Core ontology classes. The vocabulary contains 26 components, 20 failure modes, 13 root causes, 10 actions and 9 symptoms, each with synonyms used for matching.

- A failure-mode phrase is accepted only if it is compatible with the extracted component (for example, *belt slippage* requires a belt). An incompatible phrase is evidence of a failure the vocabulary cannot express on that component and becomes a drift candidate. With no phrase, the component’s default mode is inferred at confidence 0.55 and labelled as inferred.
- Lubrication rounds, inspections and modifications without a symptom are classified as planned work and receive no failure mode or root cause.

LLM as second reader. When enabled, events are sent in batches to any OpenAI-compatible endpoint (a hosted provider or a local model). The prompt constrains output to vocabulary identifiers, or other : <phrase> when the text clearly states something outside the vocabulary, and requires a per-field confidence and the exact evidence words. Responses are cached by a hash of the event payload, prompt version and model, so rebuilds do not repeat calls; malformed batches are split recursively, and any failure falls back to the rules for the affected events. The merge policy (Section 6.4) was set from measurement:

1. Structured columns always win; agreement raises confidence to 0.97.
2. On disagreement the rule value is kept, its confidence drops by 0.15 (floor 0.3), and a note records the LLM’s suggestion so the fact rises in the review queue.
3. On agreement, confidence becomes $\min(0.93, \max(c_{llm}, c_{rule}) + 0.07)$.
4. The LLM fills empty fields for failure mode, root cause, action and symptom at confidence $\min(0.7, c_{llm})$, but never components.
5. An other : answer counts as drift only where the rules found nothing.
6. The LLM cannot reclassify a failure as planned work when the rules found a component or symptom.

Drift detection. Phrases that fail to map, from structured columns, incompatible failure-mode evidence or LLM other : answers, are grouped into drift proposals with the orders they occur in. An engineer can approve a proposal as a new term (with label, synonyms, applicable components or cause category), map it as

a synonym of an existing term, or reject it. Approved changes are written to an extension file and applied on the next build; the built-in vocabulary is never modified.

4.4 Human verification and knowledge capture

The review queue presents proposed facts lowest-confidence first, each with its evidence snippet, source field and extraction method, and allows a reviewer to verify, correct (which verifies the corrected value) or reject. Bulk verification above a confidence threshold is available and logged. Every decision is written to an audit log with the reviewer’s name.

Knowledge-gap questions are generated where an asset has at least two events with the same failure mode and fewer than half of them have a confident or verified root cause. Each question names the asset, the failure mode, the number of unexplained events and the date range, asks what usually causes the failure and what check would confirm it, and suggests the technician who most often worked those orders. An answer that maps to a known cause becomes a proposed asset-level technician fact; an answer that names an unknown cause becomes a drift proposal.

4.5 Reasoning

All reasoning functions return the orders behind each result, a confidence, and a sparse-evidence flag when fewer than $\theta = 3$ events support the answer.

Root-cause ranking. For an asset and an optional symptom or failure-mode filter, each matching failure event e with a root-cause fact f_e contributes weight $\omega(f_e)$ to that cause, where $\omega = 1$ for verified facts, the extraction confidence for proposed facts, and 0 for rejected facts. Technician statements for the asset add 2.0 if verified and 1.2 if proposed. If the asset has fewer than θ explained events, events from assets of the same class are added at half weight. Scores are normalised, $s_c = w_c / \sum_{c'} w_{c'}$, and banded as high ($s_c \geq 0.5$), medium (≥ 0.25) or low, or as low (sparse evidence) when support is below θ . The response also lists typical actions and parts and the number of unexplained events, pointing to the open knowledge-gap questions.

Similar failures. Each failure event is indexed as a bag of tokens from its description, symptom, remarks, recorded root cause and extracted labels, after normalisation, and ranked against a free-text query with Okapi BM25 [11] ($k_1 = 1.2$, $b = 0.75$).

Bad actors, recurrence and spares. Bad actors are a Pareto ranking of failure events by cost, downtime or frequency, grouped by asset, failure mode or both, reporting how many groups account for 80% of the total. Recurrence uses a sliding window to find the largest set of events with the same failure mode on the same asset within W days (default 60, at least three events). Minimum spare stock per material is $S = \lceil \mu L + z \sigma \sqrt{L} \rceil$, with μ and σ the mean and standard deviation of monthly consumption, L the lead time in months and $z = 1.65$; parts are ranked by the breakdown downtime of the orders that consumed them.

4.6 Interfaces

The MCP server exposes thirteen tools over standard input/output (Table 2) and the OWL schema as a resource. Its instructions tell the client to cite evidence orders, to describe proposed facts as unverified, and to recommend asking a technician when evidence is sparse. Asset arguments accept an identifier, a name or a shop-floor alias. The web application provides an overview (Figure 2), an upload wizard with column mapping and asset discovery, the review queue, an interactive graph (Figure 3), a searchable work-order table showing raw SAP lines beside extracted facts, an editor for the asset register and vocabulary, and

Tool	Returns
<code>list_assets</code>	Assets by process area, with order and failure counts
<code>get_asset_history</code>	Orders for an asset with extracted facts, statuses, cost and downtime
<code>suggest_root_causes</code>	Ranked causes with scores, confidence bands, evidence orders and technician statements
<code>find_similar_failures</code>	BM25 matches for a free-text description, with causes, actions and parts
<code>rank_bad_actors</code>	Pareto by cost, downtime or frequency
<code>detect_recurrence</code>	Repeat failures within a window
<code>recommend_spares</code>	Minimum stock and criticality per material
<code>get_evidence</code>	Raw SAP lines and facts for an order or fact
<code>data_quality_report</code>	Mislabelled work types, unresolved assets, split postings
<code>list_knowledge_gaps</code>	Open technician questions
<code>submit_technician_answer</code>	Records an answer as a <i>proposed</i> fact (the only write tool)
<code>list_drift_proposals</code>	Terms awaiting an engineer’s decision
<code>ontology_terms</code>	The controlled vocabulary

Table 2: MCP tools. Read tools are open to any connected client; the single write tool cannot change verified knowledge.

Layer	Technology
Core and pipeline	Python 3.10+, pandas, openpyxl, rule-based TLP, BM25
LLM extraction	Any OpenAI-compatible API via httpx; evaluated with openai/gpt-oss-120b on Groq; local models via Ollama
Knowledge graph	Neo4j 5 (property graph, Cypher) or SQLite; RDF/Turtle export with rdflib; OWL 2 schema
Agent interface	Model Context Protocol server (Python MCP SDK, stdio transport)
Web API	FastAPI, Uvicorn, background build jobs
Web application	React 18, TypeScript, Vite, Cytoscape.js
Testing	pytest over SQLite, embedded openCypher and optional live Neo4j; API tests in an isolated workspace
Deployment	pip, Docker, Docker Compose

Table 3: Technology stack.

privacy settings that state, before upload, whether work-order text will leave the machine. Figure 4 shows a general-purpose assistant answering a planner’s question through the MCP server.

5 Implementation

Table 3 summarises the technology stack. The implementation is about 5,000 lines of Python; the web API, MCP server and CLI are thin layers over one core library. Storage is abstracted behind a repository interface with two implementations returning identical records: Neo4j, used when credentials are configured, and SQLite as an offline fallback and test reference. Graph queries use plain openCypher without plug-ins, so the same code is also exercised in tests on an embedded openCypher engine. Long builds run as background jobs with progress reporting. The system is packaged for pip and for Docker Compose with Neo4j.

Privacy. Maintenance records contain supplier names, costs and staff identifiers. The plant chooses one of three modes: local only (rules, no data leaves the machine), LLM API (text is sent to a hosted provider), or local LLM (an OpenAI-compatible model on the plant’s own network). The active mode is displayed in the interface at all times.

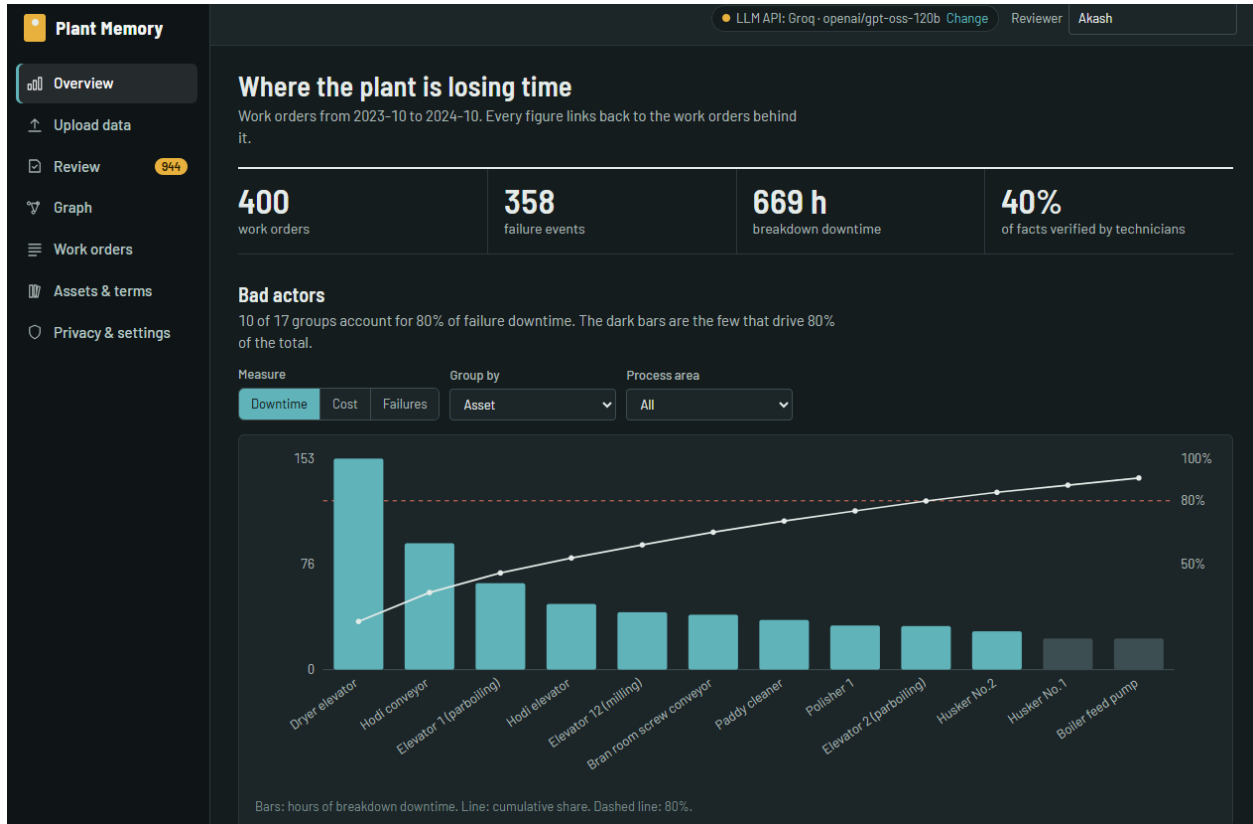


Figure 2: Overview screen on the demo dataset: summary figures and a Pareto of breakdown downtime by asset, with the cumulative-share line and the 80% threshold. The planted bad actor (the dryer elevator) ranks first.

6 Evaluation

6.1 Data

Real exports cannot be published, so evaluation uses a synthetic generator that reproduces the structure and messiness of the motivating export. It simulates 18 assets across five process areas for 12 months (October 2023 to October 2024) with Poisson failure arrivals per asset class and weighted scenarios per class (component, failure mode, symptoms, cause distribution, actions, quantities, downtime). Text is generated from deliberately messy templates, including abbreviations, upper-case variants, 40-character truncation and Hinglish remarks. Structured fields are sparsely populated: the root-cause column is filled for about half of breakdowns and a fifth of other failures, and work types are mislabelled on purpose. The generator plants four patterns: the dryer elevator fails at 2.2 times the class rate with misalignment over-represented; one V-belt is consumed across several elevators; husker rubber rolls dominate milling spend; and from the tenth month the Hodi conveyor shows gearbox oil leaks caused by a choked breather, a failure mode and cause absent from the vocabulary (drift). With seed 7 the export has 590 lines and 400 work orders (357 failures, 34 lubrication rounds and 9 modifications), plus 37 reservation lines. Ground truth is written to a separate file used only for evaluation.

6.2 Extraction (rules only)

Table 4 reports extraction against ground truth. Coverage is the share of events for which a value was extracted, and precision is accuracy on those. Root-cause coverage is limited by design: many work orders

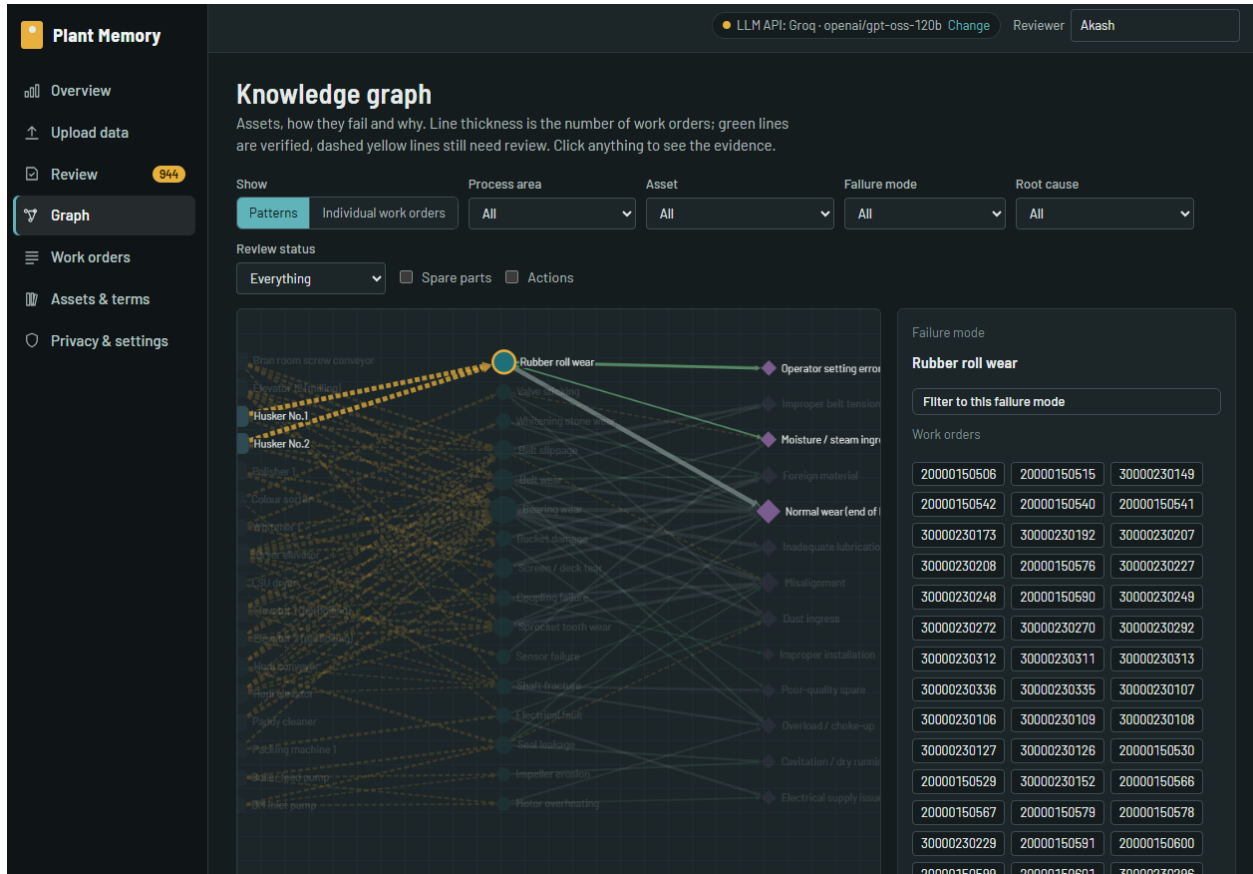


Figure 3: Graph screen (patterns view): assets, failure modes and root causes, with edge width proportional to the number of work orders. Solid green edges are verified and dashed yellow edges await review. Selecting a node (here, *rubber roll wear*) highlights its neighbourhood and lists the work orders behind it.

carry no cause evidence at all, and the system does not guess. Classification of failures versus planned work reaches precision 0.989 and recall 0.972. All 7 events whose true failure mode lies outside the vocabulary were flagged as drift.

6.3 Diagnosis

We pose one question per (asset, failure mode) pair that has at least four true events and a unique dominant true cause: “asset X keeps showing failure mode Y; what is the most likely root cause?” This yields 28 questions. The baseline is what a planner obtains from the raw export: the most frequent value typed in the SAP root-cause column for those orders. Plant Memory answers with `suggest_root_causes` after a simulated review that verifies only facts read from structured columns (confidence ≥ 0.95). Table 5 shows the results. In three of the five Plant Memory errors the system predicts *normal wear* where the true dominant cause is inadequate lubrication or moisture ingress. In two of these, the most common value typed in the root-cause column was “life over” as well, so the recorded causes themselves pointed the wrong way; these are the situations the knowledge-gap questions are meant to resolve.

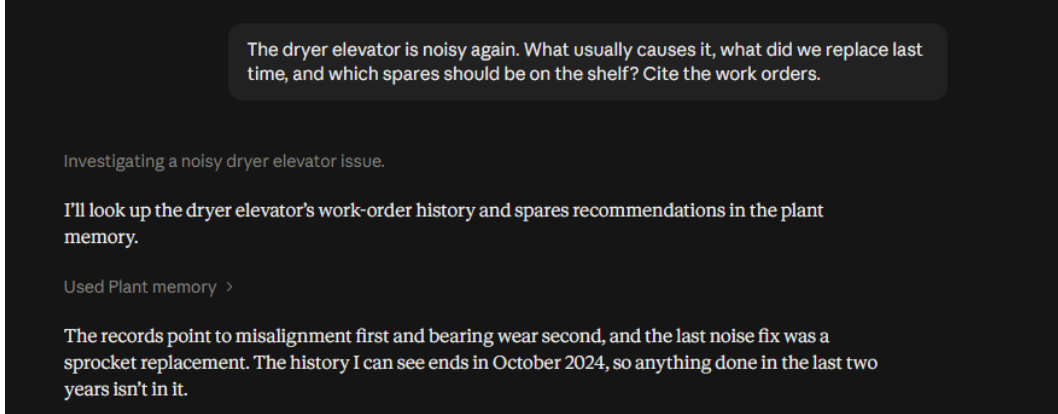


Figure 4: A general-purpose assistant answering a planner’s question through the Plant Memory MCP server. The answer is assembled from tool results over the verified graph and notes the limits of the available history.

Field	Events	Coverage	Precision	Accuracy	Macro F1
Asset	400	1.000	1.000	1.000	1.000
Component	391	0.987	0.974	0.962	0.953
Failure mode	350	0.991	0.896	0.889	0.880
Root cause	350	0.620	1.000	0.620	0.805
Action	400	0.948	0.982	0.930	0.899
Symptom	357	0.608	0.982	0.597	0.767

Table 4: Rule-based extraction on the synthetic export (400 work orders, seed 7).

6.4 Rules versus LLM

We ran `openai/gpt-oss-120b` with low reasoning effort on 399 of the 400 orders and compared it with the rules (Table 6). As an independent extractor the LLM was less accurate on every field, most markedly on failure mode (55.4% versus 88.9%). A recurring error was reading the *Preventive* work type literally and classifying genuine wear failures as planned work; in an initial merge that allowed the LLM to override the rules, this removed 99 failure events and reduced diagnosis top-1 accuracy from 82.1% to 75.0%. The LLM nevertheless added value where the rules were silent, correctly filling 8 actions and 5 symptoms (with 2 incorrect fills each), which motivated the second-reader policy described above. Under that policy, action accuracy rose from 93.0% to 95.0% and symptom accuracy from 59.7% to 61.1%, with component, failure-mode and diagnosis accuracy unchanged.

7 Discussion and limitations

The evaluation is favourable to the system for a structural reason: the ground truth comes from the same generator as the data, and the abbreviation table and synonyms were written alongside the generator’s vocabulary. Real exports will contain unseen abbreviations, other languages and plant-specific idioms, and accuracy will be lower. This is the case the review queue, drift proposals and vocabulary editor are designed for, but their effectiveness has not yet been measured. A credible evaluation requires 50 to 100 hand-labelled orders from a real plant, which would also show whether the rules-versus-LLM ranking holds once the rules face unfamiliar text. The comparison with an LLM is likewise specific to one model, one prompt and synthetic text.

Other limitations follow from scope. The system has been built for a single plant with one taxonomy;

Answerer	Top-1	Top-3	Answers citing evidence
Raw SAP root-cause column	0.536	n/a	n/a
Plant Memory	0.821	0.964	1.000

Table 5: Diagnostic accuracy on 28 questions.

Field	Rules	LLM alone	Rules + LLM (adopted policy)
Component	96.2	87.2	96.2
Failure mode	88.9	55.4	88.9
Root cause	62.3	46.9	n/r
Action	93.0	92.5	95.0
Symptom	59.7	53.5	61.1
Diagnosis top-1	82.1	n/a	82.1

Table 6: Accuracy (%) of rules, the LLM as an independent extractor, and the merged output. n/r: not reported separately; the LLM filled one root cause correctly where the rules were silent.

whether the ontology transfers to other equipment and industries is untested. There is no user study of technician engagement with verification, which is the main operational risk. The application is single-user, without authentication, and table and graph views slow down beyond roughly 50,000 lines. Ontology changes are audited but not yet versioned or reversible, and work-order drafting and preventive-maintenance interval suggestions remain future work.

8 Conclusion

Plant Memory treats maintenance text as evidence and maintenance knowledge as a set of claims that people verify. Rule-first technical language processing, an ISO 14224-aligned ontology with reified provenance, and a verification loop with knowledge-gap questions and drift proposals together produce a graph that both planners and AI agents can query, with every answer traceable to the work orders behind it. On synthetic data the approach substantially outperforms the raw root-cause column for diagnosis, and a measured comparison suggests that, for this kind of text, an LLM is more useful as a second reader than as the primary extractor. The next step is evaluation on real plant data with real technicians.

Code availability

The implementation, synthetic data generator, evaluation harness and reports are available at <https://github.com/AkashRajSingh/Plant-Memory>. The repository was made public in October 2026.

Use of AI tools

AI assistants were used during software development and in drafting and editing this manuscript. The author reviewed all content and is responsible for it. The LLM evaluated in Section 6.4 is part of the system under study.

References

- [1] Anthropic. Introducing the Model Context Protocol. <https://www.anthropic.com/news/model-context-protocol>, November 2024.
- [2] T. K. Bikaun, T. French, M. Stewart, W. Liu, and M. Hodkiewicz. MaintIE: A fine-grained annotation schema and benchmark for information extraction from maintenance short texts. In *Proceedings of LREC-COLING 2024*, pages 10939–10951, Torino, Italy, 2024.
- [3] M. P. Brundage, T. Sexton, M. Hodkiewicz, A. Dima, and S. Lukens. Technical language processing: Unlocking maintenance knowledge. *Manufacturing Letters*, 27:42–46, 2021. doi:10.1016/j.mfglet.2020.11.001.
- [4] A. Dima, S. Lukens, M. Hodkiewicz, T. Sexton, and M. P. Brundage. Adapting natural language processing for technical text. *Applied AI Letters*, 2(3):e33, 2021.
- [5] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph RAG approach to query-focused summarization. arXiv:2404.16130, 2024.
- [6] M. Hodkiewicz and M. T.-W. Ho. Cleaning historical maintenance work order data for reliability analysis. *Journal of Quality in Maintenance Engineering*, 22(2):146–163, 2016.
- [7] International Organization for Standardization. ISO 14224:2016 Petroleum, petrochemical and natural gas industries – Collection and exchange of reliability and maintenance data for equipment, 2016.
- [8] M. H. Karray, F. Ameri, M. Hodkiewicz, and T. Louge. ROMAIN: Towards a BFO compliant reference ontology for industrial maintenance. *Applied Ontology*, 14(2):155–177, 2019.
- [9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33*, 2020.
- [10] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, 2024.
- [11] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.
- [12] T. Sexton, M. P. Brundage, M. Hoffman, and K. C. Morris. Hybrid datafication of maintenance logs from AI-assisted human tags. In *2017 IEEE International Conference on Big Data*, pages 1769–1777, 2017.
- [13] W3C OWL Working Group. OWL 2 Web Ontology Language document overview (second edition). W3C Recommendation, 2012.
- [14] C. Woods, M. Selway, T. Bikaun, M. Stumptner, and M. Hodkiewicz. An ontology for maintenance activities and its application to data quality. *Semantic Web*, 15(2), 2024. doi:10.3233/SW-233299.