

Enhancing Asset Management and Maintenance with Generative AI: A Modular Architecture

Gustavo Herold Bender¹[0009-0001-4033-3462], Rafael Araujo Kluska¹[0000-0002-5296-910X],
Rafael Calixto Maluf¹[0009-0000-5287-332X], Lucas Camilotti¹[0000-0002-2253-587X] and Eduardo
de Freitas Rocha Loures^{1,2}[0000-0002-1963-6186]

¹ Pontifical Catholic University of Paraná (PUCPR), Curitiba, Brazil

² Federal University of Technology – Paraná (UTFPR), Curitiba, Brazil
gustavo.bender@pucpr.edu.br

Abstract. Asset management and industrial maintenance rely on heterogeneous sources such as work orders, asset histories, manuals, inspections, and technician notes. Organizations typically use CMMS alongside broader AMS/EAM platforms to support maintenance execution, asset lifecycle governance, performance monitoring, and decision support. Even when records are centralized, decisions may still be hindered by slow information retrieval, fragmented narratives, and limited context during troubleshooting and planning. This paper proposes a modular architecture that integrates CMMS/AMS/EAM data with Retrieval-Augmented Generation (RAG) to deliver evidence-grounded question answering and maintenance-oriented recommendations. Using Manusis4/Hive4 as a reference platform, the approach implements an interoperability layer that extracts and normalizes maintenance artifacts, generates embeddings, performs semantic retrieval over a unified database, and composes grounded prompts for large language model inference. The solution is exposed through a provider-agnostic web API and uses LangChain to orchestrate retrieval and prompt construction. Instead of computationally intensive fine-tuning, an iterative prompt formulation protocol is adopted to improve response consistency and encourage explicit use of retrieved evidence. The contribution is a deployment-oriented modular architecture, developed following a design-oriented research perspective, that supports model substitution, reduces coupling between operational systems and LLM providers, and prioritizes traceability through evidence grounding. The discussion frames retrieval-grounded generative AI as a preliminary, feasibility-oriented contribution to maintenance workflows, strengthening information access and decision support in asset management environments.

Keywords: Enterprise Asset Management - EAM; Asset Management Systems - AMS; computerized maintenance management system - CMMS; Industrial maintenance; Retrieval-augmented generation - RAG; Large language models - LLM; Generative Artificial Intelligence - GAI

1 Introduction

Industrial asset maintenance is increasingly complex as organizations pursue high availability, safety, and cost efficiency while adopting Industry 4.0 practices. Maintenance decisions depend on heterogeneous sources, including work orders, asset histories, inspection notes, manuals, and technician narratives, often fragmented across tools. Computerized Maintenance Management Systems (CMMS) centralize these records and support planning and traceability, but the resulting information volume and heterogeneity still hinder timely retrieval and synthesis for decision making [16]. From an asset management and operations perspective, the ability to organize and retrieve maintenance knowledge is directly related to reliability, continuity, and lifecycle control [19] [20].

A large share of maintenance knowledge is recorded as free text, which complicates searching for similar cases and consolidating past interventions. This motivates natural language interfaces capable of retrieving and summarizing prior occurrences, explaining procedures, and supporting structured recommendations. Recent work has explored LLM-based maintenance assistants to support planning and operations, indicating the relevance of conversational access to maintenance knowledge [17].

Large Language Models (LLMs) based on the Transformer architecture can summarize, extract, and generate technical explanations from natural language inputs [11]. However, industrial use cases require contextual consistency and traceability: answers must align with the organization’s own manuals, work orders, and maintenance history. Without grounding in organizational evidence, LLMs may generate fluent but unsupported statements, reducing trust and increasing operational risk.

Retrieval Augmented Generation (RAG) offers a practical way to mitigate this limitation by combining dense retrieval and generation: a retriever selects relevant passages from an external corpus and injects them into the prompt context used during generation [7]. Dense passage retrieval with embedding-based similarity search is a common backbone for such pipelines [6]. However, despite these advances, there is still a lack of deployment-oriented and interoperable architectures that effectively integrate RAG-based approaches with CMMS/EAM environments while ensuring traceability and contextual consistency at the asset level.

This paper presents a deployment-oriented interoperability Application Programming Interface (API) that bridges a reference CMMS (Manusis4) and LLM services, enabling contextual queries and maintenance-oriented recommendations grounded in retrieved evidence. The stack is intentionally accessible and reproducible: FastAPI exposes standardized REST endpoints [3], Docker supports portable execution environments [12], LangChain orchestrates retrieval and prompt pipelines [14], and PostgreSQL with pgvector provides a single persistence layer for structured metadata and vector similarity search [13]. LLaMA family models are accessed via GroqCloud and Hugging Face [9] [15].

The contributions of this work are: a modular API architecture that decouples CMMS data access, retrieval, and generation to support provider changes and maintainability; an end-to-end RAG pipeline for maintenance records using PostgreSQL pgvector with explicit metadata scoping for asset-level traceability; and a prompt formulation protocol

tailored to maintenance scenarios, emphasizing evidence use, response consistency, and latency constraints. The approach follows a design-oriented research perspective, in which the proposed artifact is developed and evaluated to address the identified interoperability gap.

This work should be understood as complementary to maintenance decision-support approaches that combine multi-objective optimization, multicriteria ranking, and Generative AI for maintenance scheduling. While such approaches focus on prioritization, selection, and schedule generation through methods such as NSGA-II and TOPSIS, the present study focuses on the interoperability and deployment layer needed to operationalize contextual, retrieval-grounded support through a provider-agnostic API. The remainder of the paper is organized as follows: Section 2 presents theoretical foundations and related work; Section 3 describes the methodology and implementation; Section 4 reports implementation results and preliminary validation activities; Section 5 concludes.

2 Theoretical Concepts

Industrial maintenance seeks to sustain equipment availability, reliability, and safety through preventive, predictive, and corrective actions. In asset-intensive settings, these activities are part of broader asset management practices concerned with lifecycle performance, evidence-based control, and value preservation [19] [20]. Within broader AMS/EAM environments, CMMS platforms support planning by centralizing work orders, asset hierarchies, schedules, and historical interventions, but real-world records remain heterogeneous and often incomplete or noisy. Data quality and structuring in CMMS databases can directly affect decision support and downstream analytics [16].

Large Language Models (LLMs) are neural models trained to predict token sequences, enabling them to generate and transform text for tasks such as question answering, summarization, and information extraction. Most contemporary LLMs are based on the Transformer architecture, which relies on self-attention to model relationships between tokens at scale [11]. In retrieval pipelines, dense encoders frequently build on Transformer-based language representations (e.g., BERT-style encoders) to produce embeddings for similarity search [2] [6].

Despite their general capabilities, LLMs primarily reflect patterns learned during training and may generate plausible but unsupported statements when domain knowledge is organization-specific, incomplete, or outdated. In maintenance contexts, this is a critical limitation because recommendations must be consistent with equipment documentation and with past maintenance records to preserve traceability.

Retrieval Augmented Generation (RAG) addresses this issue by combining retrieval and generation: a retriever searches an external corpus for relevant passages and injects them into the prompt/context used during generation [7]. Dense retrieval methods (e.g., Dense Passage Retrieval) represent queries and documents as embeddings and retrieve by similarity, supporting semantic search over manuals and historical work orders [6]. Related work increasingly investigates conversational and LLM-based tools for maintenance, including chatbots that assist technicians during planning and operations [17]. More broadly, research has emphasized evidence-based asset management, smart

maintenance, and maintenance digital transformation as complementary capabilities grounded in data-driven decision-making and cross-functional integration [19] [20] [21] [22]. However, many industrial deployments still face practical barriers: interoperability with CMMS data sources, repeatable integration patterns, and mechanisms to scope retrieval to the correct asset and documentation set. In particular, the challenge of integrating retrieval mechanisms with structured maintenance systems while preserving asset-level contextualization remains insufficiently addressed in the literature. These challenges motivate the architecture described in this paper, which prioritizes modularity and reproducibility over bespoke, system-specific integrations.

To operationalize LLMs with RAG in production environments, an integration layer is required to orchestrate user requests, retrieval, prompt construction, and model invocation. Implementing this layer as a web API standardizes inputs and outputs, decouples the maintenance system from specific model providers, and supports replication and testing. Frameworks such as FastAPI are well suited for this purpose due to their lightweight design and automatic documentation for REST endpoints [3]. In maintenance management terms, the relevance of this architecture lies in its support for information quality, decision traceability, and operational continuity. Maintenance planning and scheduling depend not only on optimization logic, but also on timely access to reliable contextual information about assets, work orders, prior interventions, and operational constraints. Therefore, an interoperability layer that improves retrieval, evidence organization, and response standardization can be understood as a complementary decision-support capability in asset management environments and as an enabling layer for future performance-oriented studies in operations and maintenance management [22]. This conceptual framing supports the development of the deployment-oriented interoperability architecture proposed in this study.

3 Methodology: Research steps and approach

This section describes the applied, artifact-oriented research process and the sequential steps that guided the design, implementation, and qualitative assessment of an API connecting an industrial maintenance management context (Manusis4/Hive4 as reference), a PostgreSQL-based persistence and retrieval layer, and external LLM services accessed via GroqCloud and Hugging Face. The research is structured following a design-oriented research perspective, in which a software artifact is developed to address an identified interoperability problem and subsequently evaluated in terms of feasibility and functional adequacy. The methodology prioritized accessible, free, and easily replicable solutions, and followed sequential stages with iterative refinement based on testing.

3.1 Research organization and overall approach

The work was organized as a stepwise development process to enable efficient integration between an industrial maintenance management context and generative AI models. This process can be interpreted as an artifact development cycle comprising problem identification, design, implementation, and evaluation stages. The sequence

begins with conceptual grounding, proceeds through environment setup and API implementation, and then incorporates Retrieval Augmented Generation (RAG), model access, and prompt formulation strategies. Finally, the artifact is tested and documented to support reproducibility. This organization reflects an applied integration workflow in which architectural decisions, implementation choices, and evaluation activities are progressively refined through iterative testing.

3.2 Technical study and conceptual grounding

The project began with a focused technical review of core concepts relevant to the proposed solution: generative AI, prompt formulation, RAG, and maintenance management systems, with emphasis on Manusis4 as the reference platform [8] and on literature concerning evidence-based asset management, smart maintenance, and maintenance digital transformation [19] [20] [21] [22]. This stage supported the identification of information requirements, representative queries, and the types of recommendations that would be most useful in an industrial maintenance setting. The conceptual grounding also ensured alignment with established directions in language model usage and retrieval methods. Additionally, this phase contributed to the explicit definition of the problem context and design requirements guiding the development of the proposed artifact.

3.3 Development environment and materials

The development environment was configured in Python due to its broad adoption and its ecosystem of libraries for integration, data manipulation, and AI workflows. FastAPI was selected as the API framework because it is lightweight, free, and provides automatic endpoint documentation that facilitates testing and integration [3]. To support portability and replication, the implementation relies on widely available open-source components. Docker packaging is used to standardize runtime environments and reduce dependency-related inconsistencies across machines [12]. These choices were guided by the requirement of reproducibility and ease of adoption in industrial and research contexts.

3.4 API implementation and architectural organization

The API was implemented as a modular interoperability service that sits between the maintenance management environment represented by Manusis4 workflows and data structures, a persistence and retrieval layer implemented with PostgreSQL, and external LLM services accessed via GroqCloud and Hugging Face. The design goal is to expose a simple, standardized interface for downstream applications while encapsulating the complexity of retrieval, prompt composition, and model interaction.

As an integration boundary, the API receives requests describing a maintenance question and any relevant context (e.g., an asset identifier, symptoms, or a short issue description). From the AI perspective, the API assembles the full model input by combining the user request with retrieved contextual information and explicit instructions. This boundary reduces coupling, since changes in the LLM provider or retrieval strategy can be implemented within the API without requiring changes in the maintenance management system.

Internally, the API follows a modular organization separating concerns such as request validation, database access, retrieval orchestration, prompt construction, and model invocation. This modularity supports maintainability and testing; retrieval can be assessed independently to verify that it returns relevant context before being used in end-to-end generation tests. Endpoints were created to receive workflow requests, validate inputs, interact with PostgreSQL, and communicate with LLM services. The API design emphasizes standardized input/output schemas so that downstream systems can integrate without ad hoc conventions, and database communication was implemented using standard Python libraries to preserve portability. The artifact was then assessed through incremental unit and integration tests focused on interface behavior, retrieval relevance, and response adherence to retrieved evidence.

4 Results

The results presented in this section describe the implemented software artifact and the initial feasibility-oriented validation activities for the proposed API. The contribution is not only architectural but operational: the prototype integrates CMMS-oriented data representations, embedding storage and retrieval, and LLM inference behind a stable REST interface. Qualitative tests with representative maintenance queries suggested response times compatible with interactive use and the retrieval of semantically relevant segments through the RAG pipeline. In addition, preliminary observations suggest that retrieval-grounded generation may reduce the time needed to consolidate maintenance information when compared with manual consultation of dispersed records. These observations are based on exploratory evaluation scenarios and should be interpreted as indicative evidence rather than statistically validated results. These findings should be interpreted as system-level evidence of feasibility and initial applicability rather than as a controlled operational performance study.

4.1 Architecture: interconnections, modules and technologies

The system architecture was defined after an initial conceptual study of generative AI, prompt formulation, and Retrieval Augmented Generation (RAG), which guided the main tooling and modular design decisions. In parallel, the Manusis4 CMMS platform was examined to understand typical maintenance information flows, how assets, events, and historical records are organized, and to identify where AI-assisted querying and recommendation could generate practical value in an industrial maintenance context.

Based on these requirements, a Python development environment was established with FastAPI as the primary framework for exposing standardized REST endpoints and structuring the application into decoupled components. Its automatically generated documentation facilitated manual testing and debugging during implementation. The adopted organization also supports future expansion through new endpoints and integration modules while maintaining consistent input validation and response schemas. Core endpoints were implemented to receive client requests, validate inputs, and orchestrate interactions with two key subsystems: the persistence/retrieval layer and the LLM inference layer. PostgreSQL connectivity was implemented using standard Python

libraries, allowing the same database to store structured maintenance metadata and the vector representations required for semantic search. Within this design, maintenance knowledge is treated as a combination of structured CMMS fields and unstructured narratives, requiring a representation step that transforms relevant maintenance artifacts into text segments that preserve domain meaning and context, such as compact narratives of historical maintenance events (asset identity, symptom, failure, action taken, outcome) and segmented documentation (procedures, checklists, manuals). Because retrieval quality depends on how well embeddings capture these semantics, this representation step is treated as an engineering activity rather than as a purely mechanical extraction.

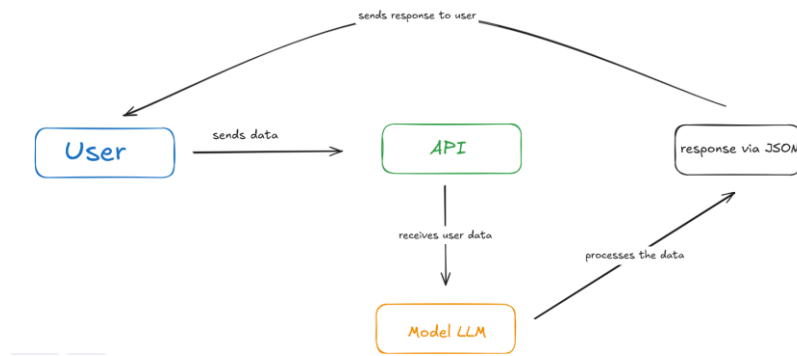


Fig. 1. Application Flow.

To implement RAG within an accessible infrastructure, the project created a mechanism for vector storage and semantic retrieval directly in PostgreSQL using open-source extensions. Relevant records and documents are converted into text segments, embedded into vectors, and stored in dedicated PostgreSQL tables together with associated metadata. At query time, the API generates an embedding for the incoming query and performs semantic similarity searches over stored vectors using the pgvector extension, retrieving the most relevant segments and injecting them into the prompt to ground generation in updated, domain-specific evidence rather than generic assumptions. Maintenance queries in industrial environments are frequently scoped (e.g., asset-specific), so the architecture stores each vector with metadata describing the segment origin, such as asset association, document type, or time period, so retrieval can remain traceable and avoid mixing unrelated equipment or documentation sets.

The implemented retrieval behavior aims to return context useful for diagnosis and decision support, potentially combining complementary perspectives such as recent incidents, procedural steps, and domain explanations.

After retrieval, the system assembles a context package and constructs a prompt to guide maintenance-relevant behavior. In practice, the prompt combines explicit task instructions, the retrieved context, and the user question, with ordering and formatting treated as factors influencing response consistency. The resulting prompt is then forwarded to the inference provider layer, and the generated output is returned to the client system. Because interoperability is an explicit objective, the interface emphasizes predictable output structure, even though the internal generation process remains probabilistic.

To avoid requiring robust local infrastructure for inference, the architecture integrates with Meta’s LLaMA family models through GroqCloud and Hugging Face services. In this setup, the API acts as the orchestration layer: it receives the request, performs retrieval when applicable, forwards the constructed prompt to the cloud inference service, and returns the generated text in a standardized response schema. The use of cloud inference is motivated by practical scalability and latency considerations for interactive use, while also supporting accessibility and reproducibility because researchers can replicate the same access pattern without hosting models locally.

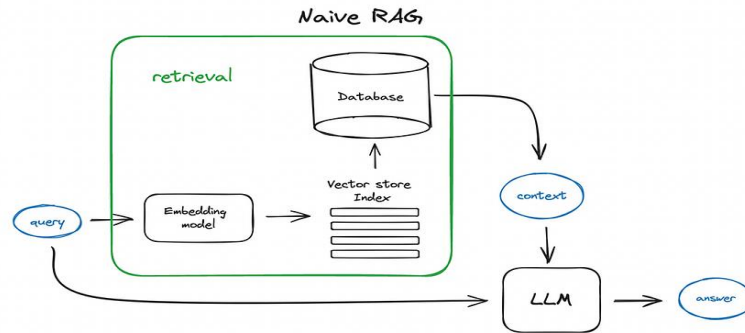


Fig. 2. RAG Process.

4.2 Model/Product: API contract and reproducibility artifacts

In interoperability-oriented research, the “product” is not limited to the executable prototype itself; it also includes the specification and traceability mechanisms that allow other researchers to reproduce the same integration pathway and verify observed behavior. In this work, the API contract was treated as an explicit scientific artifact: a stable interface that mediates between CMMS-derived maintenance information and LLM services, enabling external systems to submit maintenance questions under consistent input assumptions and to receive outputs in predictable structures. This emphasis on contract clarity reduces ambiguity during integration and strengthens the reproducibility of maintenance-oriented experiments built on top of the proposed solution.

To support replicability beyond a single execution environment, the project systematically recorded the decisions and resources required to reconstruct the

operational stack. The documentation includes architectural rationales, environment setup scripts, usage examples, data structures, and the outcomes of the validation activities conducted during implementation. By covering the full path from configuration to execution and analysis, these artifacts function as a methodological scaffold: they allow independent replication, facilitate controlled modifications, and provide a baseline for subsequent comparative studies that may evaluate alternative retrieval settings, prompt strategies, or model providers while preserving the same integration contract.

A key element of this reproducibility strategy is the availability of an inspectable interface specification that can be used both for integration and for systematic testing. During development, the use of FastAPI enabled the automatic generation of endpoint documentation, which served not only as a practical debugging aid but also as an auditable representation of the implemented contract. This strengthens methodological transparency because it exposes the concrete request/response structures used during the validation activities, enabling other researchers to reproduce comparable calls under similar constraints.

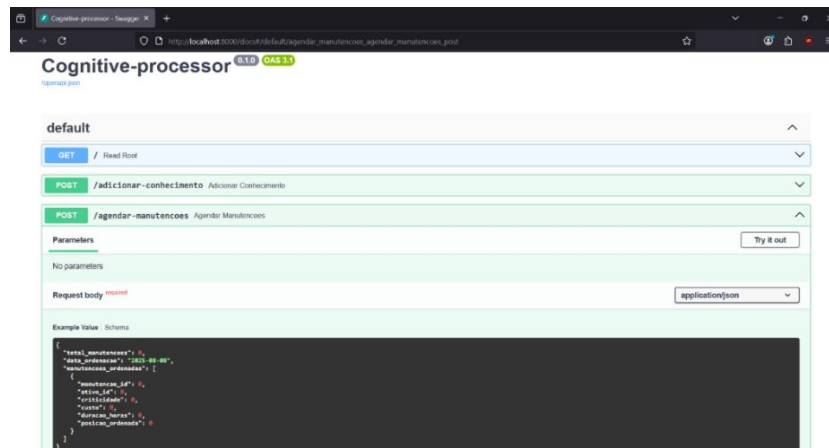


Fig. 3. Documentation for the API developed with FastAPI.

Fig. 3 shows the interactive API documentation automatically generated by FastAPI (Swagger/OpenAPI), which was used to inspect and test the interoperability service during development. The screen illustrates the available endpoints (e.g., knowledge ingestion and maintenance scheduling), together with request/response schemas and example JSON payloads, enabling validation of input structures, parameter constraints, and expected outputs.

4.3 Application/Use case: behavior and observed outcomes

From an applied perspective, the proposed API is analyzed as an interoperability mechanism that can support contextual queries and maintenance-oriented recommendations in industrial maintenance settings, reconciling data heterogeneity and infrastructure constraints. In this scenario, the objective is not only to produce fluently

coherent responses, but to examine whether generation can be anchored in organizational evidence so that the resulting guidance remains auditable and potentially useful in operational routines. Qualitative tests with representative scenarios indicated that the RAG-based approach retrieved semantically relevant segments and that response consistency in relation to the retrieved context appeared to improve when prompt formulation imposed clear constraints and structures. These observations provide preliminary evidence of applicability rather than a controlled benchmark comparison.

To make this evaluation observable and reproducible, the CMMS API flow can be described through simulated interaction scenarios, in which typical maintenance-management intentions are translated into structured inputs and expected output patterns. Rather than reproducing execution logs, the Table summarizes illustrative API use patterns from an applied validation perspective: each scenario connects a CMMS-side request to the type of structured response expected from the interoperability layer.

Table 1. CMMS–API interaction scenarios and expected responses (with example input/output).

Scenario (CMMS intent)	Example input (CMMS → API)	Expected API response
Maintenance prioritization/organization request	List of maintenance tasks to be sorted with attributes (e.g., maintenance_id, asset_id, criticality, cost, duration_in_hours) and reference date (sorting_date).	Return with ranking/sorting (e.g., sorted_maintenances + sorted_position) and metadata (e.g., total_maintenances)
Weekly scheduling request	Set of tasks/activities (with duration and priority) + available resources (technical/hours) + calendar constraints	Structured weekly allocation per technician/day (blocks), totals, and breakdown by activity.

Beyond interoperability, an additional outcome of this work is the use of an iterative protocol for formulating and refining prompts to support domain adaptation at the interaction level without adjusting model weights. Instead of conventional fine-tuning, which tends to increase computational cost and governance complexity, a refinement cycle based on domain examples and systematic evaluation of outputs is adopted: collection of representative cases, composition of the prompt with explicit instructions and demonstrations (few-shot), execution, adherence verification, and incremental adjustment of the structure and order of information. This strategy helps increase the likelihood that the model will appropriately incorporate the context retrieved by RAG into maintenance-oriented reasoning and recommendations, consistent with prior prompt programming literature [18].

Overall, the results suggest complementarity between RAG and prompt formulation: while RAG provides contextual evidence, prompt design may improve the adherence of responses to that evidence and to the intended industrial setting. The validation combined

unit tests (endpoint behavior, schema validation, and database operations) and integration tests with representative queries, focusing on the relevance of retrieved material and the adherence of responses to the returned evidence. This feasibility-oriented evaluation links transparency, traceability, and practical utility, and supports initial applicability in industrial environments. Broader quantitative benchmarking, controlled before-and-after comparisons, and user-centered evaluations with maintenance planners remain important directions for future work.

5 Conclusion

The implemented prototype supports the argument that integrating CMMS environments with generative AI can add practical value to maintenance and asset management by improving access to dispersed information and indicating the potential for more contextualized and traceable support than non-grounded generation or manual consultation of records alone. By combining FastAPI-based interoperability services, PostgreSQL with pgvector, LangChain orchestration, and cloud-hosted LLaMA-family models, the system establishes a reproducible end-to-end pathway for retrieval-grounded maintenance querying and recommendation generation. From a research perspective, this work can be understood as the design and implementation of an interoperability artifact aimed at addressing information fragmentation and contextualization challenges in maintenance environments.

The main contribution of this work is the modular interoperability architecture that shows how the combination of LLMs with Retrieval-Augmented Generation (RAG) can strengthen information access and decision support in maintenance environments by grounding responses in organizational records, manuals, and historical maintenance data. By structuring the integration through a standardized API layer, the proposed approach also supports flexibility, reproducibility, and adaptability across different industrial contexts and model providers. In this sense, the proposed architecture offers a practical deployment layer that makes contextual AI services more usable, explainable, and adaptable in industrial settings [20] [22].

Although the evidence reported here is limited to prototype implementation and qualitative validation, the results provide preliminary evidence of feasibility and initial applicability, suggesting that RAG-based generation may provide more relevant and consistent outputs than non-grounded approaches in maintenance information workflows. These findings should be interpreted as indicative of functional feasibility rather than as statistically validated performance improvements. Future studies should expand this contribution through controlled comparisons, broader benchmarking, and user-centered evaluation in real maintenance workflows.

References

1. T. B. Brown et al., “Language Models are Few-Shot Learners,” arXiv:2005.14165, 2020.
2. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” arXiv:1810.04805, 2018.

3. FastAPI Documentation. Available at: <<https://fastapi.tiangolo.com/>>. Accessed: 13 Feb. 2026.
4. J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification," arXiv:1801.06146, 2018.
5. Meta AI, "Introducing LLaMA: A foundational, 65-billion-parameter language model." Available at: <<https://ai.meta.com/blog/large-language-model-llama/>>. Accessed: 13 Feb. 2026.
6. V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," arXiv:2004.04906, 2020.
7. P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv:2005.11401, 2020.
8. Manusis4, "Maintenance and Asset Management Software." Available at: <<https://manusis4.com/>>. Accessed: 13 Feb. 2026.
9. Hugging Face, "meta-llama/Meta-Llama-3-8B model card." Available at: <<https://huggingface.co/meta-llama/Meta-Llama-3-8B>>. Accessed: 13 Feb. 2026.
10. H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv:2302.13971, 2023.
11. A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.
12. Docker Documentation, "What is Docker?." Available at: <<https://docs.docker.com/get-started/docker-overview/>>. Accessed: 13 Feb. 2026.
13. pgvector, "pgvector: Vector similarity search for Postgres," GitHub repository. Available at: <<https://github.com/pgvector/pgvector>>. Accessed: 13 Feb. 2026.
14. LangChain, "LangChain Documentation." Available at: <<https://docs.langchain.com/>>. Accessed: 13 Feb. 2026.
15. Groq, "Groq API Reference." Available at: <<https://console.groq.com/docs/api-reference>>. Accessed: 13 Feb. 2026.
16. L. Stazić, N. Račić, T. Stanivuk, and Đ. Dobrota, "Determination of Benefits of the Application of CMMS Database Improvement Proposals," Applied Sciences, vol. 13, no. 4, 2731, 2023. doi:10.3390/app13042731.
17. L. Kohl, S. Eschenbacher, P. Besinger, and F. Ansari, "Large Language Model-based Chatbot for Improving Human-Centricity in Maintenance Planning and Operations," in Proc. European Conf. of the PHM Society 2024, vol. 8, 2024. doi:10.36001/phme.2024.v8i1.4098.
18. L. Reynolds and K. McDonell, "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm," arXiv:2102.07350, 2021.
19. M. Herrera Galán, "Evidence-based asset management applied to maintenance function control," Journal of Quality in Maintenance Engineering, vol. 25, no. 4, pp. 635–644, 2019. doi:10.1108/JQME-05-2018-0045.
20. J. Bokrantz, A. Skoogh, C. Berlin, T. Wuest, and J. Stahre, "Smart Maintenance: a research agenda for industrial maintenance management," International Journal of Production Economics, vol. 224, 107547, 2020. doi:10.1016/j.ijpe.2019.107547.
21. L. Silvestri, A. Forcina, V. Introna, A. Santolamazza, and V. Cesarotti, "Maintenance transformation through Industry 4.0 technologies: A systematic literature review," Computers in Industry, vol. 123, 103335, 2020. doi:10.1016/j.compind.2020.103335.
22. J. Bokrantz and A. Skoogh, "Adoption patterns and performance implications of Smart Maintenance," International Journal of Production Economics, vol. 256, 108746, 2023. doi:10.1016/j.ijpe.2022.108746.