Integrating a Knowledge-Grounded RAG Chatbot for Multi-Tenant Industrial Machinery Troubleshooting

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Software Engineering
April 2026
Dumindu De Silva

UNIVERSITY OF TURKU
Department of Computing

Dumindu De Silva: Integrating a Knowledge-Grounded RAG Chatbot for Multi-Tenant Industrial Machinery Troubleshooting

Master of Science (Tech) Thesis, 55 p.
Software Engineering
April 2026

Industrial organizations depend heavily on machinery for daily operations, where even minor failures can cause significant downtime and financial loss. A central challenge in troubleshooting is that the necessary knowledge is dispersed across lengthy manuals and technical documents, making it slow and inefficient for technicians to locate relevant instructions during urgent situations. While recent progress in large language models (LLMs) has demonstrated strong capabilities in understanding and generating natural language, their direct use in industrial environments remains limited. In addition, LLMs lack access to proprietary machine documentation and may produce inaccurate or overly generalized responses, which creates risks in safety-critical contexts. This thesis develops and investigates an intelligent chatbot powered by Retrieval-Augmented Generation (RAG) for end-to-end machinery troubleshooting, conducted in collaboration with Fatman Oy[1]. The proposed system integrates document retrieval with generative capabilities to provide machine-specific, context-aware responses grounded in technical manuals and operational documentation. A key focus of the work is the design of a multi-tenant retrieval architecture that prevents cross-contamination of information between documents belonging to different customers or machines by applying metadata-based filtering before the retrieval stage. In addition, the system incorporates a structured feedback integration mechanism that allows domain experts to refine the knowledge base, ensuring that outdated manual content can be corrected and that the most up-to-date troubleshooting guidance is returned to users. The study also investigates how such a RAG-based chatbot can be deployed using a scalable architecture and seamlessly integrated into an existing application at Fatman Oy, addressing challenges related to document ingestion, vector storage, model hosting, user session management, and workflow coordination. By grounding generative AI in continuously refined documentation within a scalable environment, the proposed approach demonstrates how industrial troubleshooting can be accelerated, information reliability improved, and operational downtime reduced.

Keywords: RAG, Generative AI, Industrial Machinery, LLM, Chatbot

---

[1] `https://fatman.fi/fi/`

Contents

Acknowledgement

I would like to express my sincere gratitude to all individuals and organizations who contributed to the successful completion of this thesis.

First and foremost, I´m extremely thankful to my academic supervisor, Tuomas Mäkilä, for his patience, expertise, and unwavering support. His guidance through key milestones, dedicated time for discussions, and thorough feedback were essential in maintaining the rigor and direction of this thesis. I would also like to thank Panu Puhtila for his support, insightful discussions, and valuable contributions during brainstorming sessions. His input and encouragement helped foster a productive and motivating research environment.

Also I am deeply grateful to Fatman Oy for providing the foundation for this research, including the thesis topic, necessary infrastructure, and continuous support throughout the project. Their commitment to innovation created an ideal environment to explore and develop the ideas presented in this work. I would like to extend my special thanks to Mugunthan Ravichandran from Fatman Oy for his constant guidance, valuable insights, and ongoing support during the course of this study. His mentorship played a significant role in shaping the direction and quality of this research.

Finally, I would like to acknowledge all those who directly or indirectly contributed to this work.

List of Figures

List of Tables

1 Introduction

In industrial environments, efficient troubleshooting of machinery is critical for maintaining productivity, minimizing downtime, and ensuring the longevity of equipment [1]. However, operators and technicians often face considerable challenges when dealing with breakdowns or routine maintenance tasks. Much of the knowledge required to resolve issues is buried in extensive user manuals and technical documentation. Locating the right troubleshooting steps in these documents under time pressure can be a frustrating and time-consuming process, often delaying recovery and increasing operational costs.

In recent years, conversational AI and chatbots have emerged as powerful tools for enhancing user interactions with digital systems. Modern large language models (LLMs), such as GPT-based systems, have demonstrated remarkable capabilities in natural language understanding and generation [2], [3], [4], [5]. They can provide coherent, context-aware, and human-like responses, making them highly suitable for question-answering and decision-support tasks. In industrial contexts, an LLM-powered chatbot could, in principle, allow operators to query machinery-related information in natural language instead of manually searching through documentation.

Despite their impressive language understanding and generation capabilities, large language models (LLMs) face significant limitations when applied directly to domain-specific tasks such as machinery troubleshooting [2]. Being trained on broad,

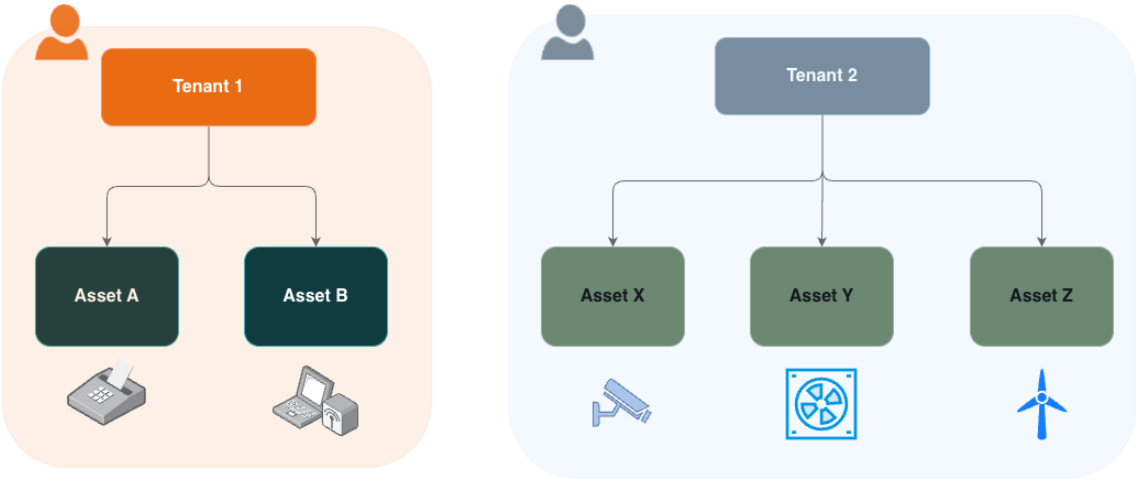

Figure 1.1: Multi-tenant, Multi-asset Industrial Environment.

internet-scale datasets, they lack access to specialized industrial knowledge. Consequently, their outputs, while fluent, may be factually incorrect, outdated, or inconsistent with specific machine manuals. To overcome this limitation, the integration of document-grounded retrieval mechanisms with generative models has gained attention. This approach, known as Retrieval-Augmented Generation (RAG), enables chatbots to first retrieve the most relevant passages from internal documentation and then generate responses grounded in that retrieved knowledge. For industrial troubleshooting, this means that a chatbot can not only converse naturally with the user but also ensure that its responses are accurate, machine-specific, and traceable to authoritative documents. Despite the strong performance of generative AI conversational models in many academic and commercial domains, their practical adoption in industrial manufacturing environments has progressed more slowly [6].

As illustrated in Figure 1.1, in multi-tenant, multi-machine industrial environments, additional challenges arise: responses must remain strictly machine-specific, avoiding cross-contamination between documents from different tenants or machinery, a problem that has seen limited exploration in prior research. Furthermore, LLMs are prone to "hallucinations," generating plausible but incorrect information

- a critical concern in safety-sensitive operations. To further mitigate risks associated with outdated knowledge, incorporating feedback from domain experts is essential as shown in Figure 1.2. This feedback loop allows the system to continuously refine and update its responses, maintaining reliability over time. Yet, few studies have investigated the integration of expert feedback specifically in industrial machinery chatbots, highlighting a clear gap that this thesis aims to address.

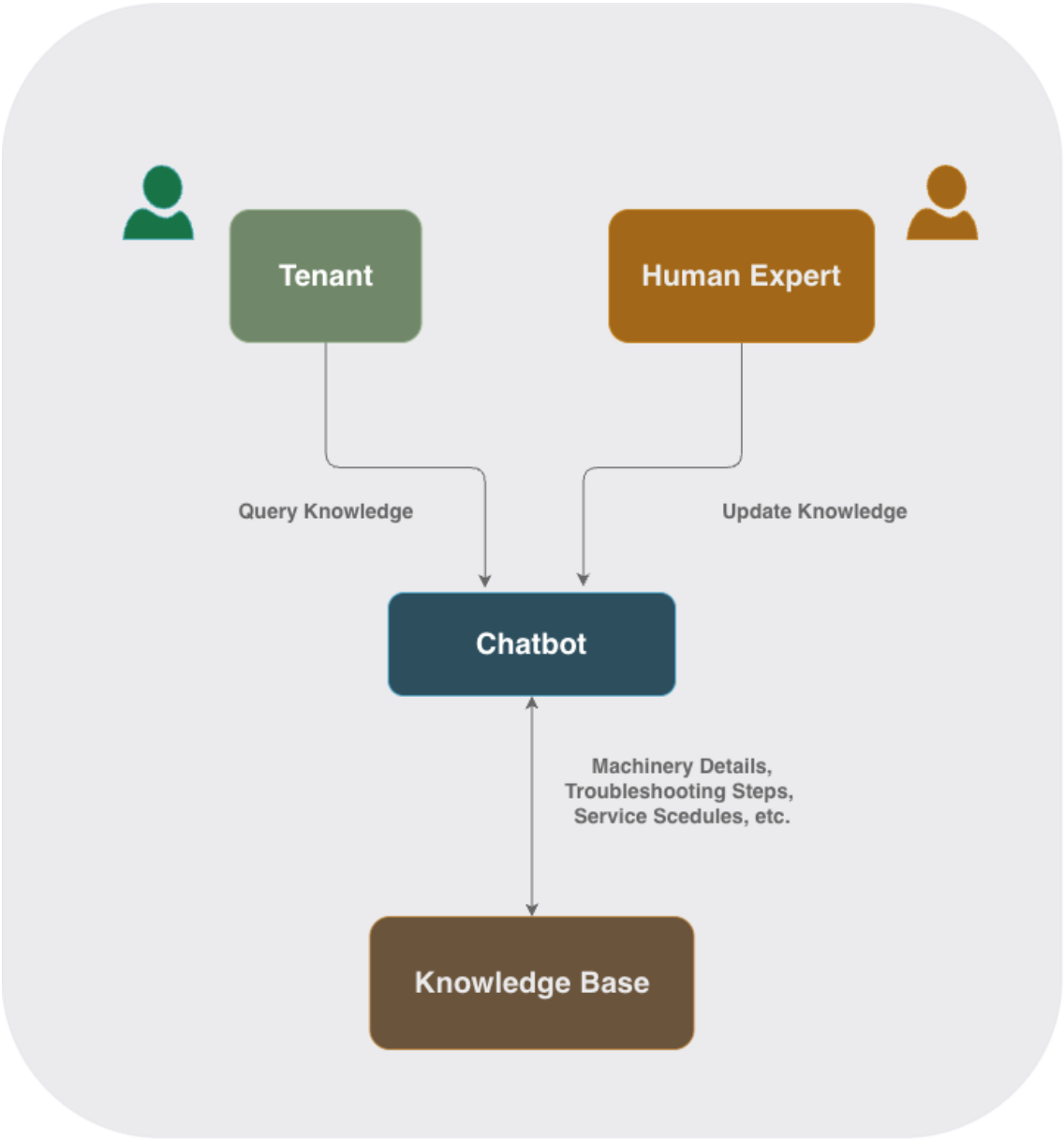

Figure 1.2: Experts update machinery knowledge and tenants query.

This thesis, conducted in collaboration with Fatman Oy, investigates how a Retrieval Augmented Generation (RAG)-based chatbot can be integrated into the machinery troubleshooting pipeline. The study aims to demonstrate how combining document-grounded retrieval with conversational AI can streamline troubleshooting, support maintenance activities, and reduce downtime. The research focuses on preventing cross-contamination between machine-specific manuals, incorporating structured expert feedback to ensure up-to-date guidance, and achieving seamless integration with cloud-based industrial workflows.

1.1 Problem Statement

In industrial settings, machinery troubleshooting knowledge is predominantly stored in extensive and complex user manuals. Operators and technicians often face significant challenges in quickly locating relevant information during unexpected breakdowns or scheduled maintenance. This leads to:

- **Extended downtimes**, as users manually search through documents for troubleshooting steps.
- **Reliance on outdated knowledge**, as manuals and procedures may not reflect recent updates or changes in machinery, increasing the risk of incorrect actions.

Consequently, critical knowledge remains underutilized, operational efficiency suffers, and recurring equipment failures are more likely. By leveraging an intelligent, document-grounded system that incorporates machine-specific guidance and expert feedback, these challenges can be mitigated. Such a system has the potential to reduce downtime, improve maintenance adherence, ensure knowledge remains current, and streamline the overall troubleshooting workflow.

1.2 Research Questions and Objectives

The complexity and volume of machinery documentation, along with the limitations of conventional troubleshooting methods, demand a systematic approach to enhancing maintenance efficiency and operational reliability. This study investigates the integration of a Retrieval-Augmented Generation (RAG)-based intelligent chatbot into an end-to-end machinery troubleshooting pipeline, with a focus on seamless architectural integration into existing applications. The chatbot is designed to provide context-aware guidance, prevent cross-contamination of information between machine-specific manuals from different tenants, and incorporate expert feedback to maintain an up-to-date knowledge base, thereby continuously refining its responses. To guide this investigation, the study is structured around the following research questions, which address both the technical and practical challenges of deploying such a system in industrial environments.

1.2.1 Research Questions

1. **RQ1:** What ***architectural design*** enables seamless integration of the chatbot into existing applications, considering challenges such as performance, document management, model hosting, user session handling, and workflow coordination?

2. **RQ2:** How can a Retrieval-Augmented Generation (RAG)-based chatbot be designed to provide ***machine-specific, context-aware responses*** while preventing cross-contamination of information between manuals of different machinery

3. **RQ3:** How can ***structured feedback from domain experts*** be effectively incorporated into the RAG framework to continuously refine the accuracy and relevance of machine-specific troubleshooting responses?

1.2.2 Research Objectives

Aligned with the research questions, the objectives of this study define specific outcomes that the research seeks to achieve. These objectives guide the design, implementation, and evaluation of the RAG-based chatbot system, ensuring that it addresses both technical challenges and operational needs in machinery troubleshooting.

1. To explore and design a **robust architectural framework** for integrating the chatbot into existing industrial applications, addressing challenges related to document handling, model hosting, user management, and workflow coordination to ensure seamless deployment and operation.

2. To develop a RAG-based chatbot capable of retrieving accurate, machine-specific troubleshooting guidance from manuals and other documentation, while **preventing cross-contamination** of information between different machines and tenants, ensuring that responses remain contextually relevant and verifiable.

3. To incorporate a **structured expert feedback loop**, allowing domain experts to correct or refine the chatbot's responses, thereby continuously improving the accuracy, reliability, and relevance of guidance across different machinery.

1.3 Declaration of Generative AI Usage

In the preparation of this thesis, a limited set of publicly available generative AI tools was used solely as supportive aids to expand the breadth of the literature review and to improve grammatical correctness.

Scholar Labs[1] was utilized to enhance the breadth and depth of the literature review by facilitating the discovery and aggregation of relevant research papers and academic sources beyond traditional keyword-based search methods. This enabled a more comprehensive exploration of existing work in the field.

Claude[2] and *ChatGPT*[3] were used to assist in improving the clarity and organization of the text, refining the articulation of research gaps, supporting the brainstorming of design strategies, and validating technical approaches. Additionally, it was used for grammatical refinement and language improvement.

All content presented in this thesis reflects the author's original ideas and understanding. The use of AI tools was limited to supportive functions such as language refinement and structuring. No content was directly copied or reproduced from AI-generated outputs. The author ensured that all final material maintains academic integrity and originality with critical review.

1.4 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 presents a focused theoretical background, positioning this work within the broader research landscape. It introduces the key concepts underlying the study, including Retrieval-Augmented Generation (RAG), multi-tenancy in AI systems, and mechanisms for incorporating expert feedback. In addition, it reviews relevant literature and existing approaches, highlighting their limitations and motivating the need for the proposed methodology.

Chapter 3 describes the system context and research methodology in detail. It first provides an overview of the existing system at Fatman Oy in which the solution is deployed. The chapter further explains the overall system design and architectural

[1] `https://scholar.google.com/scholar_labs/search`

[2] `https://claude.ai`

[3] `https://chatgpt.com`

approach by detailing the two primary flows of the proposed RAG system.

Chapter 4 presents the implementation of the proposed system. It explains how the theoretical concepts and methodological choices are translated into a working solution, covering the core components. In particular, the chapter explains how the implementation evolved from a local RAG system into an Azure-based cloud solution.

Chapter 5 evaluates the proposed approach with respect to the three research questions. The evaluation combines both quantitative and qualitative analysis, including performance metrics and coverage. The chapter covers the evaluation process along with the obtained results for the three research questions.

Finally, Chapter 6 concludes the thesis by reflecting on the key findings across the three research questions. It also discusses limitations that may affect generalizability and outlines directions for future research, including potential extensions and improvements to the proposed framework.

2 Theoretical Background

This chapter provides the theoretical foundation for positioning the work presented in this thesis. Each subsection aligns with one or more of the research questions, thereby supporting the overall scope and breadth of the study. The chapter first outlines the area of RAG-inspired industrial troubleshooting. Although this topic has received limited attention in the context of Retrieval-Augmented Generation, there are still several relevant studies and related approaches that provide useful insights and motivate the research presented in this thesis. Since one of the research questions focuses on handling multi-tenancy while avoiding cross-tenant contamination, the chapter also examines the challenges and strategies associated with supporting multi-tenant architectures in RAG applications. Particular attention is given to data isolation, and architectural considerations that help ensure safe retrieval of tenant-specific data. In addition, another research question addresses the incorporation of expert feedback to improve response quality and system accuracy. Therefore, this chapter also reviews relevant literature on human-in-the-loop systems, expert validation, and feedback-driven improvement mechanisms in RAG applications.

2.1 RAG inspired Industrial Troubleshooting

The integration of Retrieval-Augmented Generation (RAG) architectures into industrial applications represents a significant advancement in leveraging large language models (LLMs) for domain-specific problem-solving. RAG systems combine

semantic retrieval from structured or unstructured data sources with the generative capabilities of LLMs, enabling responses that are both context-aware and grounded in factual knowledge. Two recent contributions exemplify the use of RAG in distinct industrial domains: general industrial troubleshooting and additive manufacturing.

The authors in [7] propose a dual-component RAG framework that combines intelligent retrieval mechanisms with generative LLMs to support troubleshooting in complex industrial environments. The system retrieves relevant content from a heterogeneous corpus including technical manuals, sensor data, and maintenance logs using embedding-based semantic search (e.g., GTE models from Hugging Face). Incoming user queries are similarly embedded and matched against this corpus. Retrieved documents are then used to augment the user prompt before it is passed to an LLM such as GPT-4 or Gemini for final response generation. This architecture is particularly effective in bridging static knowledge (manuals and logs) with dynamic inputs (real-time telemetry), enabling condition-aware recommendations and adaptive guidance.

The AMGPT model proposed by [8] targets a highly specialized industrial subdomain metal Additive Manufacturing (AM). This domain-adapted LLM is built on the LLaMA2-7B model and enhanced through a RAG pipeline to overcome the limitations of general-purpose models like GPT-4 in providing accurate responses to materials science queries. Unlike end-to-end training, AMGPT leverages a dual-encoder architecture where user queries and documents are embedded using the sentence-transformers/all-mpnet-base-v2 model. Semantic retrieval is performed using cosine similarity within a vector database (VectorStoreIndex via LlamaIndex), drawing on a curated corpus of 50+ academic PDFs processed via Mathpix for structured embedding.

While these works demonstrate the effectiveness of RAG in industrial contexts, they do not address the challenges of building tenant-specific industrial chatbots

that enforce strict isolation of knowledge between machinery types, organizational units, or customers. In other words, existing solutions focus on general industrial knowledge retrieval but lack mechanisms to prevent cross-contamination of information in multi-tenant deployments, which is a key focus of this thesis.

2.2 Handling Multitenancy and Cross-Contamination

2.2.1 Multitenancy and Isolation Challenges

Retrieval-Augmented Generation (RAG) systems are increasingly adopted in enterprise and industrial environments where multiple tenants, users, or organizational units share a common AI infrastructure. While this shared deployment model improves scalability and cost efficiency, it introduces significant challenges related to data isolation and cross-contamination [9], [10]. In the context of RAG, cross-contamination occurs when retrieved context or generated responses incorporate information originating from document collections that are unrelated or unauthorized for a given query, thereby undermining correctness, safety, and trustworthiness.

Enterprise-scale RAG architectures commonly address multitenancy by enforcing data isolation through a combination of tenant-specific vector namespaces, query-time filtering, and document-level permission inheritance from source systems [10]. These approaches ensure that retrieval is restricted to content associated with the requesting tenant or organizational unit. However, existing approaches predominantly address authorization correctness, ensuring that users retrieve only documents they are permitted to access. They do not explicitly account for semantic correctness in cases where multiple, structurally similar document collections, such as manuals for different machines or configurations, coexist within the same tenant. As a result, retrieval can be authorization-correct yet contextually inappropriate.

Recent studies on multitenant RAG systems formalize isolation strategies through

patterns such as Silo, Pool, and Bridge [11], highlighting different ways to dedicate or share resources among tenants. Complementing these patterns, a four-plane taxonomy identifies the main system layers where isolation must be enforced: the data plane, which governs storage and access of tenant documents, chunks, and metadata; the vector plane, responsible for embeddings, vector storage, indexing, and similarity search, with design choices affecting both memory and query performance; the orchestration plane, which manages ingestion, query routing, retrieval, reranking, and context assembly, ensuring that tenant identity and filters propagate consistently across all workflow stages; and the LLM plane, covering prompt construction, inference, response filtering, and telemetry, where multi-tenancy relies on policy-based scoping and careful handling of tenant-specific prompts and retrieved context. Failure to enforce isolation consistently across any of these planes can result in risks such as cross-tenant embedding leakage, retrieval contamination, membership inference, or metadata inference. These findings collectively emphasize that ensuring secure, tenant-aware RAG operation requires end-to-end enforcement across storage, retrieval, orchestration, and generation layers, rather than relying on partial controls at individual components.

While these frameworks provide a comprehensive view of isolation at the tenant level, they largely assume that all documents within a tenant are contextually compatible. In industrial settings, however, manuals for different machines, configurations, or asset generations may be accessible to the same users but must not be mixed during retrieval, as doing so can produce misleading or unsafe troubleshooting guidance. This form of cross-contamination is semantic rather than strictly security-related and is not fully addressed by existing multitenant isolation models.

2.2.2 Strategies to handle Multitenancy

At the core of Retrieval-Augmented Generation (RAG) systems, vector databases play a decisive role in determining how effectively tenant isolation can be enforced during retrieval. Multi-tenancy in AI applications enables multiple users or organizations to share the same system while maintaining strict separation of data and operations, which is particularly critical for large language models where deploying per-user instances would be prohibitively expensive [12]. Implementing strong isolation involves multiple layers, including secure storage, reliable authentication mechanisms, logical access controls such as role-based (RBAC) and attribute-based (ABAC) permissions, and metadata-based filtering to enforce fine-grained access [12].

Existing approaches to multitenancy in vector databases typically rely on either a single shared index with metadata-based filtering or fully separate per-tenant indices [9]. Shared indices offer improved memory efficiency but require careful and consistent application of filters, whereas per-tenant indices reduce the risk of retrieval interference at the cost of increased memory consumption and operational complexity [11]. The Curator indexing approach proposes a hybrid solution by embedding tenant-specific clustering trees as compact substructures within a shared global index [9]. This design allows each tenant's vector distribution to be modeled independently while maintaining low memory overhead.

Data tagging is a common method for enforcing tenant-specific isolation in multi-tenant systems, where each data segment is labeled with unique tenant identifiers. These tags enable fine-grained access control by ensuring that only authorized users or the owning tenant can access the data, supporting secure and efficient retrieval. When combined with advanced query mechanisms, data tagging provides a scalable framework for privacy-aware applications. Existing work in metadata-aware RAG pipelines demonstrates that filtering retrieved chunks using tenant or asset-specific

metadata can prevent cross-contamination between documents [13].

To mitigate retrieval-time leakage and enforce fine-grained access control, several studies propose combining metadata-aware filtering with Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) [13]. In these systems, document chunks are enriched with structured metadata such as content categories, resource identifiers, or access attributes which are then used to constrain similarity search results. An LLM may additionally be employed to infer the intent or information type of a query, allowing the system to dynamically construct metadata filters that align retrieval with user permissions. Evaluations in sensitive domains such as healthcare demonstrate that this approach effectively prevents unauthorized data exposure and supports regulatory compliance.

In summary, existing literature demonstrates that metadata-based filtering and access-aware retrieval are effective mechanisms for enforcing security and compliance in multitenant RAG systems. However, these studies primarily focus on secure access control and do not evaluate or illustrate the integration of such metadata filtering in a machine-specific troubleshooting context. In this thesis, we adopt metadata-based filtering at the retrieval stage to guarantee that only the relevant machine-specific chunks are retrieved, ensuring that the generative model receives exclusively the appropriate context, thereby preventing cross-contamination in practice.

2.3 Incorporating Expert Feedback into RAG Systems

Ensuring the long-term reliability of Retrieval-Augmented Generation (RAG) systems requires more than accurate retrieval at deployment time; it demands continuous adaptation as knowledge sources evolve. As highlighted in [14], maintaining a vector database involves coordinated ingestion pipelines, preprocessing workflows,

embedding generation, efficient indexing, and mechanisms for handling incremental updates. When documents are modified or newly added, embeddings often need to be regenerated to preserve semantic accuracy, and selective re-indexing strategies are required to balance computational efficiency with system responsiveness. While such mechanisms ensure that the knowledge base remains synchronized and scalable, they primarily address structural and operational consistency rather than deeper integration of domain expertise.

To move beyond static knowledge maintenance, Human-in-the-Loop (HITL) paradigms introduce active human participation into the learning cycle [15]. HITL approaches enable experts to supervise, correct, and refine system outputs during operation, thereby improving accuracy, trustworthiness, and alignment with real-world standards. Unlike traditional methods that rely solely on labeled datasets, HITL frameworks aim to incorporate expert reasoning processes, allowing systems not only to receive correct answers but also to gradually internalize how domain specialists think and make decisions. In the context of RAG systems, this shift from static document updates to feedback-driven refinement has led to the emergence of knowledge editing techniques [16]. Broadly, these approaches can be categorized into model editing methods, which adjust the underlying language model parameters using expert feedback [16], and memory-based methods, which retain expert corrections externally and incorporate them during retrieval without modifying the core model. The following sections examine these two strands of research and position the present thesis within this evolving landscape.

2.3.1 Model Editing Techniques

Within HITL, *model editing* techniques focus on adjusting the underlying language model based on expert feedback. For instance, in a conversational agent designed for behavioral interventions, the system incorporates feedback from both clients and

therapists during interactions [17]. Experts provide structured guidance, such as encouraging empathetic responses or identifying critical behavioral cues, which is transformed into a reward model that fine-tunes the generative language model. This iterative optimization allows the model to improve over time while building a high-quality, domain-informed conversational dataset.

Similarly, the work in *Pistis-RAG* demonstrates a RAG-specific implementation of model editing [18]. In this system, user feedback is not applied at the level of individual documents but to the ordered set of examples used to generate a response. Each interaction which includes the query, retrieved candidates, generated answer, and user reaction is recorded and used to train a listwise ranking model. Over time, this enables the system to better select and order retrieved content according to user preferences, enhancing alignment between generated outputs and expert or end-user expectations.

The work in *ORCHID* framework [19] introduces a modular, agent-based architecture designed for high-risk property (HRP) classification tasks. In this system, human reviewers provide structured feedback on model decisions, which is systematically recorded for both auditability and iterative refinement. Rather than immediately altering the underlying model parameters, the feedback is stored in a dedicated database and associated with the corresponding data instances. This tagged feedback can then be leveraged in subsequent training or retrieval stages to improve classification consistency and accountability over time. By combining human oversight with persistent feedback storage, this research demonstrates how traceable, feedback-driven improvement can be embedded into retrieval-augmented decision systems operating in high-stakes domains.

In legal AI applications, a similar methodology has been employed to refine domain-specific RAG outputs [20]. Expert evaluations of generated answers are converted into weighted numerical scores, reflecting relevance, accuracy, and com-

pliance with regulatory standards. These scores are used as rewards in reinforcement learning, specifically via Proximal Policy Optimization (PPO), which gradually updates the model to favor higher-quality outputs. This approach ensures stable and reliable refinement, even in sensitive domains where erroneous outputs carry significant consequences.

In addition to reinforcement-based refinement, several model editing approaches seek to incorporate new knowledge directly into the internal parameters of a language model through structured fine-tuning. In one such research [16], synthetic training data is first generated using carefully designed prompts that guide the model to transform domain-specific text into high-quality question–answer pairs following a consistent format. These curated examples are then used to update selected subsets of the model's parameters, enabling the system to internalize the new information rather than depending solely on external retrieval at inference time. To maintain computational efficiency, parameter-efficient fine-tuning techniques are typically employed, modifying only limited components of the network while preserving the majority of pretrained weights. This strategy differs from conventional RAG pipelines, where knowledge remains entirely external and is retrieved on demand. By embedding selected domain knowledge within the model itself while retaining retrieval as a complementary mechanism such hybrid approaches aim to improve response accuracy, reduce dependency on repeated lookups, and support more stable long-term knowledge integration [16].

Overall, model editing techniques in HITL frameworks provide a mechanism for RAG systems to continuously improve by learning from structured expert feedback. They enhance response quality, incorporate domain expertise, and support adaptation to evolving knowledge bases. However, current approaches largely focus on general conversational or domain-specific contexts and have yet to fully explore integration into industrial, multi-tenant RAG systems where both tenant isolation

and domain-specific expert feedback must coexist. Addressing this gap is a central objective of the present thesis.

2.3.2 Memory-Based Feedback Integration

In contrast to model editing approaches that modify the internal parameters of a language model, memory-based techniques preserve expert feedback externally and incorporate it during retrieval or inference without altering the core model weights. These methods treat feedback as an evolving knowledge resource, stored in structured repositories, document collections, or auxiliary databases that can be queried and reused in future interactions. By maintaining a persistent memory layer, such systems enable continuous improvement while avoiding the computational cost and stability risks associated with repeated fine-tuning.

One representative approach is *STACKFEED* [21], which reframes knowledge base refinement as a structured search problem. Rather than adjusting the language model itself, this research iteratively improves the external documents that a RAG system retrieves from. Each version of the knowledge base is considered a system "state," and document edits are treated as "actions" that transition the system toward higher-quality states. Using Monte Carlo Tree Search (MCTS), the framework explores alternative document modifications efficiently. Individual editing agents are assigned to specific documents, while a centralized critic evaluates the collective impact of edits using counterfactual multi-agent reinforcement learning techniques. This design allows the system to attribute improvements or degradations in downstream performance to specific document changes. Experimental results show that coordinated, feedback-driven document refinement can substantially improve knowledge base quality and RAG performance, particularly in low-resource programming and factual question-answering settings. Compared to model editing, this approach offers greater transparency and controllability, but requires structured document

management and careful evaluation mechanisms.

Similarly, *MARK* [22] integrates instructor feedback directly into the external knowledge repository rather than the language model parameters. In this framework, instructors review generated answers and provide corrections or enhancements, which are incorporated into the raw document collection. Both sparse and dense vector indices are subsequently updated to reflect these additions. For moderately sized datasets, this re-indexing process remains computationally feasible, enabling practical deployment in educational environments. The advantage of this approach lies in its simplicity and auditability; however, performance improvements depend on timely index updates and effective retrieval alignment.

The framework presented in [23] demonstrates memory-based adaptation at the system orchestration level. This architecture combines intent-based predefined responses with a RAG pipeline, routing straightforward queries to static replies while forwarding complex cases to dynamic generation. User ratings and indirect signals, such as rephrased follow-up questions, are collected continuously. Frequently unresolved queries are clustered semantically, triggering the creation of new intents and responses once a threshold is reached. Confidence thresholds for routing are dynamically recalibrated based on aggregated feedback. Instead of retraining the model, the system improves performance by adjusting routing policies and expanding its structured response repository. This method enhances adaptability and scalability, though it relies on well-defined clustering and routing strategies.

In educational contexts, the research in [24] employ memory-based personalization by combining tagging mechanisms with structured prompts. Learner onboarding data and real-time feedback ratings are stored and used to influence subsequent retrieval and explanation strategies. Multiple parallel pipelines ranging from fully personalized retrieval to direct language model invocation are compared to measure the impact of feedback and personalization. By toggling components on and off, the

system quantifies the contribution of memory-driven adaptation. While effective for measuring personalization effects, this approach increases architectural complexity due to parallel evaluation pipelines.

The framework *ARIA* [25] further illustrates structured knowledge accumulation without weight updates. When the agent detects uncertainty, it queries human experts for clarification within a limited interaction budget. Extracted knowledge statements are stored in a timestamped repository, compared against existing entries, and reconciled in case of conflicts. Through iterative self-evaluation and targeted consultation, the system strengthens its external knowledge store over time. This design promotes transparency and continual learning, yet depends on accurate uncertainty detection and efficient knowledge reconciliation mechanisms.

In evaluation-oriented domains, the work in [26] adopts a similar memory-driven refinement strategy. The system compares model-generated grades with expert-labeled scores, identifies discrepancies, and refines grading rubrics accordingly. Human clarifications are stored and used to iteratively update structured grading rules through a multi-agent optimization pipeline. Rather than modifying the model directly, improvements arise from progressively refining the rubric and associated prompts. This approach enhances stability and interpretability, though it requires repeated analysis cycles and structured rule management.

Memory-based correction is also evident in recent approaches [27], where draft translations are revised using human or automated feedback. Correction examples are stored in a repository and later retrieved as in-context demonstrations for similar translation tasks. By leveraging accumulated correction cases during inference, the system gradually improves domain alignment without parameter updates. This method balances adaptability with computational efficiency but relies on effective similarity matching for retrieval of relevant correction examples. Similarly, the research in [28] introduces a Manual Correction System (MCS) that integrates expert

input directly into reasoning chains during inference. When the model exhibits uncertainty, humans intervene by correcting only specific erroneous reasoning steps rather than retraining the model or editing the knowledge base. These targeted textual modifications guide the model toward accurate conclusions while keeping human effort minimal. Although this approach does not persist corrections in a long-term memory store, it exemplifies inference-time memory augmentation that enhances reliability without structural model changes.

Across these works, memory-based feedback integration offers several advantages: transparency, auditability, computational efficiency, and reduced risk of destabilizing pretrained models. Unlike model editing, which internalizes knowledge through parameter updates, memory-based approaches maintain knowledge externally and incorporate it dynamically during retrieval, routing, or reasoning. In comparison to these approaches, the present thesis explores how structured expert feedback can be integrated within an industrial RAG chatbot setting while preserving tenant isolation and operational scalability, thereby addressing the practical constraints of multi-tenant deployment environments.

3 Methodology

This chapter introduces the system under study at Fatman Oy and describes the current context in which the proposed solution is deployed. It explains the existing challenges within the system and motivates why the proposed methodology is a feasible and suitable approach for addressing them. The chapter then presents an overview of the proposed methodology, showing how it is designed to resolve the key issues identified in the problem setting and answer the research questions of this thesis. In particular, it explains how the methodology supports industrial troubleshooting through a Retrieval-Augmented Generation (RAG) architecture while also addressing important concerns such as multi-tenancy, tenant isolation, and expert feedback integration. The chapter describes the proposed architecture in a step-by-step manner, using a generic RAG workflow as the foundation.

3.1 System Overview

The proposed solution is designed to enhance a core product at Fatman Oy, forming the foundation for improving machinery troubleshooting workflows. Fatman Oy is a Finland-based company providing enterprise solutions that support digital transformation in the industrial and service sectors. Its flagship platform offers digital lifecycle management for machinery manufacturers, enabling efficient asset management, scalable operations, and seamless integration of machine data with existing ERP systems.

Currently, troubleshooting machinery issues relies heavily on manual consultation of distributed documentation, including technical manuals, guidelines, and historical records. This process is time-consuming, prone to errors, and limits the speed at which maintenance personnel can respond to machine faults. There is a need for a more centralized, real-time solution that enables personnel to quickly access relevant troubleshooting knowledge without carrying or searching through physical documents. Specifically, the integration of a RAG-based chatbot is intended to streamline the troubleshooting process by providing fast, machine-specific guidance, reducing downtime, and minimizing reliance on manual inspection of documentation.

3.2 Architectural Design

The proposed system follows a modular Retrieval-Augmented Generation (RAG) architecture consisting of two primary flows: **Data Ingestion** and **Search and Response Generation** (Figure 3.1). The objective of this architecture is to transform machine-specific documentation into a structured, searchable knowledge base and to enable context-aware conversational interaction grounded in authoritative sources.

The design separates ingestion, retrieval, and generation components to ensure scalability, maintainability, and deployment flexibility. This abstraction allows the system to be implemented using different model hosting strategies (e.g., local or cloud-based deployments) without altering the underlying methodological framework.

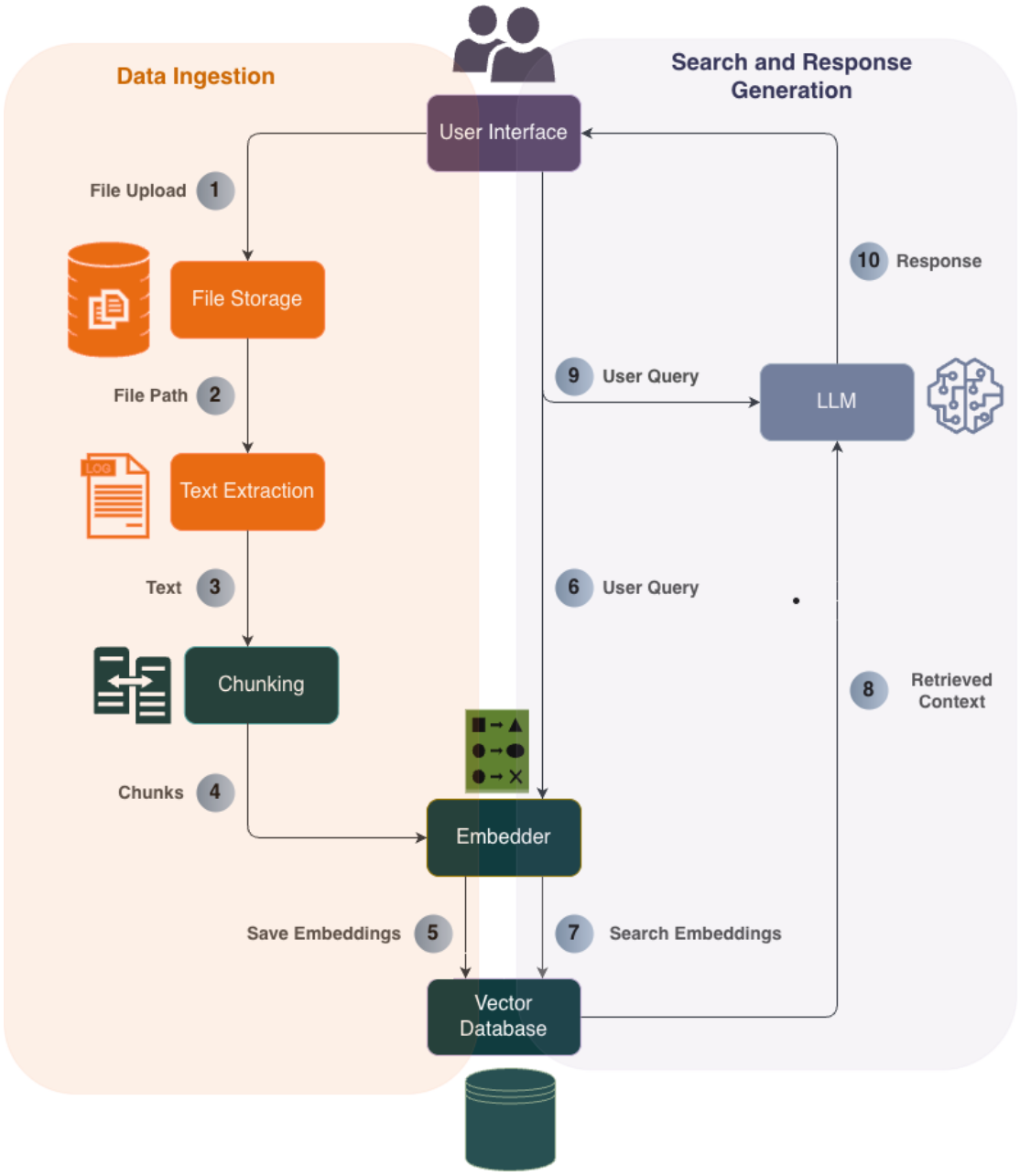

Figure 3.1: Overall RAG System Architecture.

3.2.1 Data Ingestion

The data ingestion pipeline begins when a user uploads a document (e.g., PDF technical manuals, troubleshooting guides, maintenance documentation) through the application interface. The document is processed by a backend ingestion service responsible for text extraction, segmentation, embedding, and indexing.

First, textual content is extracted from the uploaded file while preserving structural coherence. The extracted text is then segmented into smaller, semantically meaningful chunks. A sliding window overlap strategy is employed to maintain contextual continuity between adjacent segments. This ensures that important instructions or multi-step procedures are not fragmented in a way that degrades retrieval quality.

Each chunk is subsequently transformed into a dense vector representation using an embedding model. These embeddings encode semantic meaning, enabling similarity-based retrieval rather than relying solely on keyword matching. Alongside the vector representation, structured metadata is stored for each chunk. This metadata includes identifiers such as `tenant_id`, `asset_id`, document type, and timestamps, which are critical for ensuring data isolation and machine-specific retrieval.

The embeddings and metadata are stored in a vector-enabled indexing system capable of efficient similarity search and metadata filtering. This indexed knowledge base forms the foundation for retrieval during query processing.

3.2.2 Search and Response Generation

When a user submits a query through the conversational interface, the request is processed by a query handling service. The query undergoes embedding using the same embedding model applied during ingestion to ensure both documents and queries reside within a shared semantic vector space.

The resulting query embedding is used to perform similarity search against the indexed document embeddings. Retrieval is constrained by metadata-based filtering to ensure strict isolation between tenants and machine assets. Only document chunks associated with the appropriate tenant and asset identifiers are eligible for retrieval. This design directly addresses the research objective of preventing cross-contamination between manuals belonging to different machinery or customers.

The top-ranked chunks are assembled into a contextual payload. A prompt construction component integrates the retrieved context with the user's query in a structured format. This augmented prompt is then forwarded to a large language model (LLM), which generates the final response.

By conditioning the LLM on retrieved domain-specific content, the system ensures that responses are grounded in authoritative documentation rather than relying solely on the model's pre-trained knowledge. This retrieval-grounded generation mitigates hallucination risks and improves factual consistency in industrial troubleshooting scenarios.

3.3 Data Collection and Management

The knowledge base consists of machine-specific documentation provided by different industrial customers. Each uploaded document is associated with structured identifiers, including `tenant_id` and `asset_id`, which enable strict logical separation between different organizational entities and machinery types.

All ingested content is stored in a secure document storage system before preprocessing. Metadata fields are mapped into the retrieval index to enable filtering during search operations. This metadata-driven design is central to preventing cross-manual contamination and ensuring that generated responses remain machine-specific.

3.4 Chunking and Embedding Strategy

Effective chunking is essential to balance contextual richness with retrieval efficiency. In this study, chunk sizes are selected to maintain sufficient semantic completeness while remaining within language model input constraints. Overlapping windows are used to reduce the risk of splitting coherent instructions across chunk boundaries.

The embedding model encodes each chunk into a high-dimensional vector representation. These embeddings capture conceptual similarity, enabling the system to retrieve relevant content even when user queries use terminology different from the original manual text. Embedding consistency between ingestion and query stages ensures alignment within the semantic space, which is critical for retrieval precision.

3.5 Metadata-Based Isolation Mechanism

To address the risk of cross-contamination between manuals, the system incorporates metadata-based filtering at retrieval time. Each indexed chunk contains tenant and asset identifiers. During search operations, retrieval queries explicitly apply filters restricting results to the relevant tenant and asset scope.

This isolation mechanism ensures that retrieved knowledge fragments originate exclusively from the correct machine documentation. By integrating filtering directly into the retrieval stage rather than post-processing results, the system enforces strict contextual boundaries at the indexing layer.

3.6 Incorporating Expert Feedback

Industrial knowledge evolves over time due to software updates, hardware modifications, and operational experience. To accommodate this dynamic nature, the system incorporates structured expert feedback into the RAG pipeline.

In this approach, expert-provided corrections or additions are processed through the same chunking and embedding pipeline similar to original files. The updated knowledge is indexed with appropriate metadata, timestamps, and optional version indicators.

To prevent conflicts between outdated and corrected content, metadata flags (e.g., active status, version identifiers, or validity windows) can be applied. Retrieval queries then filter or prioritize the most recent or authoritative entries. This mechanism ensures long-term consistency and auditable knowledge evolution.

4 Implementation

Chapter 3 described a deployment-agnostic Retrieval-Augmented Generation (RAG) framework consisting of ingestion, embedding, retrieval, and generation stages. This chapter presents how that generic architecture was instantiated in practice in order to evaluate what type of architectural design enables seamless integration into existing industrial applications. In particular, the implementation focuses on practical challenges such as performance, document management, model hosting, user session handling, and workflow coordination.

4.1 Implementation Overview

Two concrete implementations were developed and evaluated:

1. A local deployment using an open-source LLM (Ollama) and a self-hosted vector database (Qdrant).

2. A fully managed cloud deployment using Azure OpenAI and Azure AI Search.

The transition from the local deployment to the cloud architecture was driven by empirical performance evaluation, explained in detail in Chapter 5, together with architectural trade-off analysis, with the goal of identifying a scalable and maintainable solution suitable for real-world industrial environments.

4.2 Initial Implementation: Local RAG Deployment

4.2.1 Local System Configuration

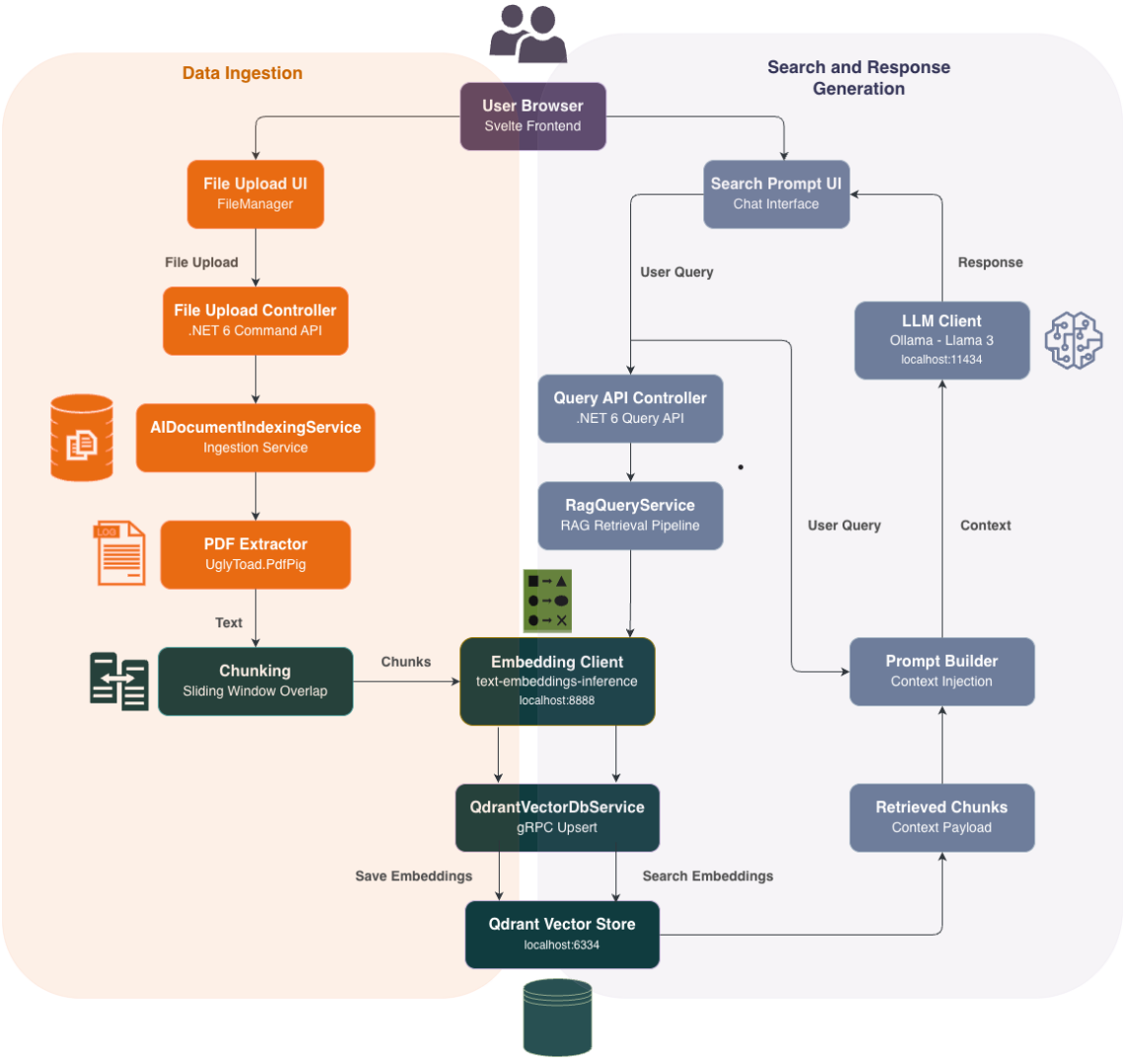

Figure 4.1: Local Deployment Architecture.

The first implementation instantiated the methodological RAG pipeline using entirely local infrastructure as shown in Figure 4.1. The objective of this design was to maintain full architectural control, avoid vendor dependency, and satisfy potential

data locality requirements relevant to industrial environments.

The ingestion pipeline followed the generic methodology:

- PDF text extraction from uploaded manuals.
- Sliding-window chunking strategy to preserve contextual continuity.
- Embedding generation using a locally hosted embedding service.
- Storage of embeddings and metadata in a vector database.

The vector index database (Qdrant/Quadrant Index DB) was deployed locally and accessed via gRPC. Each indexed chunk contained structured metadata fields, including tenant identifiers and asset identifiers, enabling machine-specific filtering as described in the methodological framework.

For the generation stage, a locally hosted LLaMA 3.1 model was served through the Ollama framework. The prompt construction process combined retrieved document chunks with the user's query before forwarding the augmented prompt to the LLM.

Thus, the local deployment represented a full instantiation of the RAG pipeline described in Chapter 3, with all components self-managed.

4.2.2 Operational Challenges

Although functionally correct, empirical testing revealed substantial operational constraints.

1. **Inference Latency.** Average response latency from query submission to completed response generation was approximately 180 seconds under typical workloads. This included embedding computation, vector retrieval, prompt assembly, and LLM inference. However, such latency hinders interactive industrial troubleshooting scenarios, where technicians require near real-time responses.

2. **Hardware Constraints.** The local LLaMA 3.1 deployment required significant computational resources. Without dedicated high-end GPU acceleration, inference performance degraded substantially. Scaling to multiple concurrent users would require proportional hardware expansion.

3. **Operational Complexity.** Maintaining embedding services, vector databases, and model serving infrastructure introduced engineering overhead. Service orchestration, dependency management, and monitoring required continuous maintenance effort.

While the local architecture validated the conceptual feasibility of the RAG framework, its performance characteristics necessitated architectural reconsideration.

4.3 Revised Implementation: Azure-Based RAG Deployment

The move toward a managed cloud-based deployment was motivated by practical limitations observed in the initial local setup. The local implementation introduced noticeable delays during response generation, and the available hardware restricted the system's ability to scale when handling larger document collections or concurrent user requests. In addition, maintaining the full infrastructure locally required considerable manual configuration and monitoring, increasing the overall operational burden. Considering the industrial context of the application, where stable performance and fast response times are essential, the design focus shifted from maintaining full control over the infrastructure to achieving higher reliability, easier maintenance, and better scalability through managed cloud services.

4.3.1 System Architecture in Azure

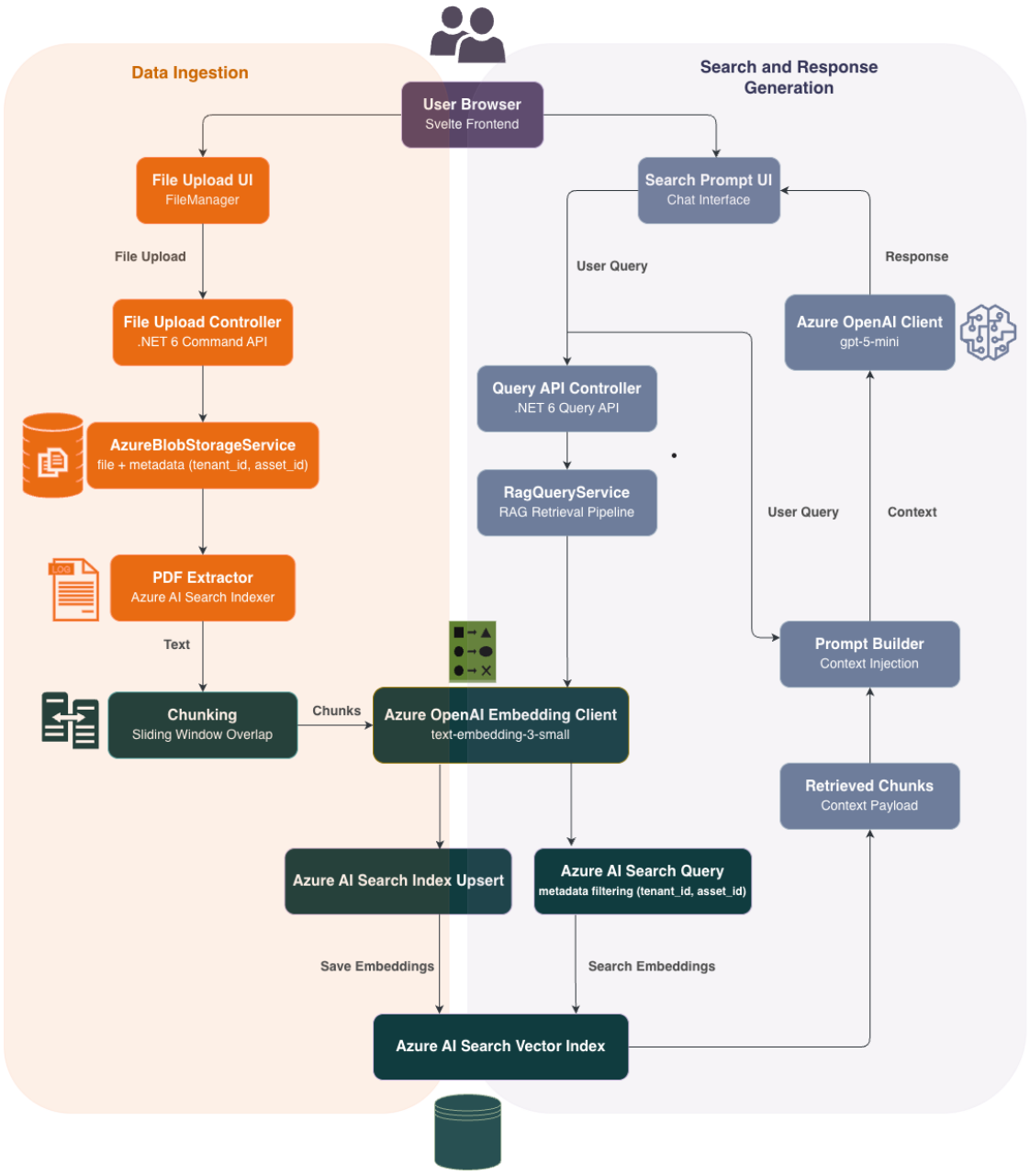

Figure 4.2: Azure-Based Deployment Architecture.

As illustrated in Figure 4.2, the revised implementation retained the same methodological RAG pipeline while replacing locally hosted components with managed services available in the Azure cloud environment. In this architecture, machine manuals and related documents are stored in Azure Blob Storage, which serves as the

centralized repository for all source data. Indexing, vector storage, hybrid retrieval, and metadata-based filtering are handled by Azure AI Search, enabling efficient and tenant-aware document retrieval. Response generation is performed using an Azure OpenAI deployment of the GPT-5 mini model, which produces context-aware answers based on the retrieved content. A .NET-based backend API coordinates the ingestion pipeline and query processing logic, ensuring controlled interaction between storage, search, and generation components. The user-facing interface is implemented using a Svelte frontend that provides a conversational chat environment through which technicians can interact with the system during troubleshooting tasks.

Document ingestion step uploads manuals to Azure Blob Storage. Azure AI Search indexes the documents, generates embeddings, and stores chunk-level metadata including `tenant_id` and `asset_id`. These metadata fields are mapped into index schema fields to enable strict filtering during retrieval.

During query execution, the user's input is first converted into a vector representation using the configured embedding model. This embedding is then used by Azure AI Search to perform either vector-based or hybrid retrieval, depending on the query configuration. To ensure correct data isolation in the multi-tenant environment, metadata filters are applied so that only documents associated with the relevant tenant and asset are considered during retrieval. The selected context is then forwarded to the Azure OpenAI deployment, where the language model receives both the user query and the retrieved document excerpts as input. Based on this grounded context, the GPT model generates the final response, ensuring that the answer remains aligned with the corresponding machine documentation.

Thus, the architectural logic remains identical to the methodological description, while infrastructure management is offloaded to Azure.

4.3.2 Rationale for Architectural Selection

The revised Azure-based architecture was ultimately selected due to its superior performance, scalability, and reduced implementation complexity. Given the industrial use case's requirement for near real-time interaction and consistent responsiveness, the cloud deployment demonstrated clear operational advantages. This architectural evolution reflects an iterative engineering process in which both quantitative and qualitative empirical evaluation, detailed in Chapter 5, guided design refinement. While both configurations employ RAG as a conceptual foundation, the deployment choice prioritizes performance and maintainability without compromising foundational design principles.

5 Evaluation

This chapter presents the experimental evaluation of the proposed RAG-based chatbot system. The objective of the evaluation is to determine whether the system satisfies the requirements defined by the research questions. The chapter covers both the evaluation procedure and the results obtained for each research question, combining quantitative metrics with qualitative analysis where relevant.

5.1 Evaluation of RQ1: Seamless Architectural Integration

Research Question 1 examines what architectural design enables seamless integration of the chatbot into existing applications, while considering practical constraints such as performance, document management, model hosting, user session handling, and workflow coordination. In an industrial setting, the chatbot must function as part of a larger system rather than as an isolated prototype, which makes deployment architecture a critical factor. To address this question, two implementations of the same Retrieval-Augmented Generation (RAG) pipeline were developed and evaluated both qualitatively and quantitatively: a fully local deployment using self-hosted components, and a managed cloud-based deployment using Azure services.

5.1.1 Evaluation Procedure

Both deployments were tested using the same document collection, ingestion pipeline, and query set representing typical machinery troubleshooting scenarios. The evaluation focused on characteristics relevant to real-world integration, including response latency, scalability under repeated use, operational complexity, and response quality. In addition to quantitative measurements, qualitative observations were recorded during development regarding deployment effort, infrastructure stability, and the ease of maintaining the system within an application workflow. Using identical data and query configurations ensured that the observed differences were caused by architectural design choices rather than variations in retrieval or prompting.

5.1.2 Results

The comparison revealed clear differences between the two architectural approaches. The local deployment provided full control over the environment but suffered from high latency, limited scalability, and increased operational overhead, which complicates integration into production systems. The cloud-based architecture, on the other hand, offered faster response times, automatic resource scaling, and simplified infrastructure management, making it more suitable for integration into existing industrial applications. The following subsections present a detailed evaluation of the two implementations in terms of performance, scalability, cost, and functional response quality.

1. **Performance and Latency Analysis**

   Latency measurements were collected for both deployments using a standardized query set representative of industrial troubleshooting tasks. The local deployment exhibited mean response latency of approximately 180 seconds. The Azure-based implementation consistently produced responses within 5–20

seconds under identical workload conditions. This represents an order-of-magnitude improvement in responsiveness.

2. **Scalability and Resource Management**

   The local architecture required manual provisioning of computational resources. Scaling to additional users would require hardware upgrades and configuration. The Azure deployment benefits from elastic resource provisioning and managed infrastructure. Embedding generation, indexing, and model inference scale automatically within service limits, reducing operational burden.

3. **Cost Considerations**

   The local solution avoids per-request API costs but incurs capital expenditure for hardware and operational maintenance costs. In contrast, the Azure deployment follows a usage-based billing model, eliminating infrastructure ownership but introducing recurring operational expenses. The selection between these models depends on projected workload, concurrency requirements, and available infrastructure resources.

4. **Functional Comparison**

   Qualitative evaluation indicated improved contextual coherence and reduced hallucination frequency in the Azure-based GPT deployment compared to the locally hosted LLaMA 3.1 model. This may be attributed to:

   - Larger-scale model optimization.
   - Mature semantic ranking in Azure AI Search.
   - Managed inference infrastructure with optimized GPU acceleration.

Table 5.1: RQ1 Comparison of Local and Azure-Based Implementations

Criterion	Local Deployment	Azure Deployment
Average Latency	∼180 s	∼5–20 s
Scalability	Hardware-limited	Elastic cloud scaling
Operational Complexity	High	Moderate to Low
Cost Structure	Capital + Maintenance	Usage-based
Maintainability	Self-managed	Provider-managed
Response Quality (qual.)	Moderate	High

5.2 Experimental Setup for RQ2 and RQ3

The evaluation was conducted using the production-integrated architecture described in the implementation chapter. The system consists of Azure OpenAI for response generation (GPT-5-mini deployment) and Azure AI Search for vector-based document retrieval. Machine manuals are stored in Azure Blob Storage and indexed with structured metadata attributes, specifically `tenant_id` and `asset_id`. These metadata fields are mapped to filterable index fields and applied as mandatory constraints during retrieval operations.

To support the systematic evaluation, the open-source benchmarking tool `Promptfoo`[1] was integrated into the experimental workflow. Promptfoo [29] is an open-source evaluation framework which allows automated execution of predefined test cases against a live API endpoint and supports structured assertions and LLM-based grading [30]. Its configuration-based approach enables reproducible experimentation and simplifies iterative validation when system modifications are introduced.

The evaluation follows the LLM-as-a-Judge rubric criteria[2]. In this paradigm, the Azure GPT-5-mini model is used to evaluate the outputs of another language model instance according to predefined criteria. This approach was selected for three reasons. First, it enables scalable evaluation across a large number of test cases without manual annotation. Second, it reduces subjective human bias in grading.

[1] `https://www.promptfoo.dev`

[2] `https://www.promptfoo.dev/docs/configuration/expected-outputs/model-graded/llm-rubric/`

Table 5.2: RQ2 Overlapping WaterJet specification fields used for adversarial queries.

Specification	WaterJet L	WaterJet H	WaterJet P
Positioning accuracy	±0,08 mm/m	±0,025 mm/m	±0,025 mm/m
Repeatability	±0,08 mm	±0,015 mm	±0,015 mm
Z-axis options	200/300 mm	200/300 mm	100/200/300 mm
X/Y max speed	15/40 m/min	15/40 m/min	40/100 m/min
Max cutting heads	4	6	6

Third, it allows evaluation criteria to be expressed in structured rubrics that focus on factual correctness and specification-level precision.

It should be noted that using the same model family as both the subject under evaluation and the judge introduces a potential methodological concern. To mitigate this, all 30 standard test cases were additionally verified manually against ground-truth values extracted directly from the machine datasheets. These manually verified results serve as a credibility anchor for the automated evaluation, confirming that the LLM-as-a-Judge outcomes are consistent with human assessment.

Three machine manuals were used in the evaluation: WaterJet P and WaterJet H belonging to Tenant A, and WaterJet L belonging to Tenant B. The manuals intentionally contain partially overlapping terminology - for example, *positioning accuracy*, *repeatability*, *Z-axis stroke*, and *maximum cutting speed* appear as named specification fields in all three datasheets - but differ in their numerical values. This overlap creates a realistic and challenging test environment for detecting cross-contamination during retrieval, since vector similarity alone may match a query to any of the three manuals. Table 5.2 illustrates the key overlapping specifications and their machine-specific values.

Each machine manual covers approximately five pages of technical content. Consequently, the number of genuinely distinct, numerically verifiable facts per manual is limited. Attempting to generate 30 questions per machine would force repetitive or artificially varied queries that do not represent meaningful evaluation cases. In-

Table 5.3: Summary of the RQ2 evaluation dataset.

Machine	Tenant	Standard queries	Adversarial queries
WaterJet L	Tenant B	10	5
WaterJet H	Tenant A	10	5
WaterJet P	Tenant A	10	5
Total		30	15

stead, 10 factual questions were constructed per machine, each mapping to a unique and directly verifiable specification extracted from the datasheet. This yields a standard test suite of 30 queries across the three machines. An additional 5 adversarial cross-contamination queries per machine were constructed targeting the overlapping specification fields identified in Table 5.2, resulting in 15 adversarial queries and a total evaluation dataset of 45 queries. Table 5.3 summarises the dataset structure.

The standard test questions were constructed manually from the datasheets to guarantee that each question has exactly one correct answer traceable to the correct machine's documentation. The Promptfoo test format used for the standard queries is retained from the original implementation. An example test case for WaterJet L is shown below:

Listing 1 Example standard query test case.

```
{yaml}
- vars:
    question: What is the positioning accuracy of WaterJet L?
    assetId: 707a0ae8-2b8f-4e80-ad4e-7f83e8460914
  assert:
    - type: llm-rubric
      value: The answer must contain "+/- 0,08 mm/m".
  metadata:
    machine: WaterJetL
    tenant: TenantB
```

The adversarial queries use the same format but are constructed from terminology shared across all three manuals. The `asset_id` field determines which machine-specific value constitutes the correct answer for each adversarial query. An example

adversarial test case is shown below:

Listing 2 Example adversarial test case definition.

```
{yaml}
- vars:
    question: What is the repeatability of this machine?
    assetId: 707a0ae8-2b8f-4e80-ad4e-7f83e8460914
  assert:
    - type: llm-rubric
      value: The answer must contain "+/- 0,08 mm".
  metadata:
    machine: WaterJetL
    tenant: TenantB
```

The same adversarial query issued with the `asset_id` of WaterJet H or WaterJet P requires the answer to contain ±0,015 mm instead. This design makes adversarial queries the most sensitive indicator of cross-contamination: a correct response in the filtered configuration requires the retrieval mechanism to have constrained context to the right machine, while an incorrect response in the unfiltered configuration reveals that vector similarity alone retrieved content from a semantically adjacent but incorrect manual.

5.3 Evaluation of RQ2: Machine-Specific Context Isolation

RQ2 investigates whether the RAG-based chatbot can provide machine-specific, context-aware responses while preventing cross-contamination between manuals belonging to different machines and tenants. In a multi-asset industrial environment, incorrect contextual blending may lead to technically inaccurate or unsafe recommendations. Therefore, strict isolation of retrieval context is a critical requirement.

5.3.1 Evaluation Procedure

For each test case, Promptfoo invoked the chatbot through the backend API. The backend then executed the RAG pipeline consisting of vector retrieval via Azure AI Search followed by response generation using GPT-5-mini.

To evaluate the effect of metadata-based contextual isolation, the experiment was conducted under two configurations. In the *filtered* configuration, both `tenant_id` and `asset_id` filters were applied during retrieval, restricting candidate documents strictly to the manual belonging to the requesting tenant and machine. In the *unfiltered* configuration, metadata filtering was disabled, allowing retrieval to rely solely on vector similarity without structural constraints. By executing the identical set of 45 test cases under both configurations, it was possible to directly observe the impact of metadata filtering on contextual correctness and cross-contamination.

In each run, the retrieved context was passed to GPT-5-mini for answer generation. The generated response was then evaluated using the rubric defined in the Promptfoo test configuration. Standard and adversarial queries were reported separately to distinguish between baseline retrieval performance and performance under semantic pressure from overlapping terminology.

Performance was measured using binary accuracy:

$$\text{Accuracy} = \frac{\text{Number of Correct Responses}}{\text{Total Test Cases}}$$

In addition, a set of boundary tests was conducted by passing a deliberately mismatched `asset_id` - one that does not correspond to any indexed manual - alongside a valid query. This tests whether the system fails safely by returning a graceful no-result response rather than silently falling back to the nearest semantically similar manual.

Table 5.4: RQ2 Accuracy results by query type and retrieval configuration.

Query type	Configuration	Correct	Accuracy
Standard (30)	Filtered	30 / 30	100.00%
Standard (30)	Unfiltered	26 / 30	86.66%
Adversarial (15)	Filtered	15 / 15	100.00%
Adversarial (15)	Unfiltered	9 / 15	60.00%
All (45)	Filtered	45 / 45	100%
All (45)	Unfiltered	35 / 45	77.78%

5.3.2 Results

Table 5.4 and Figure 5.1 summarise the results for RQ2 across all configuration and query types.

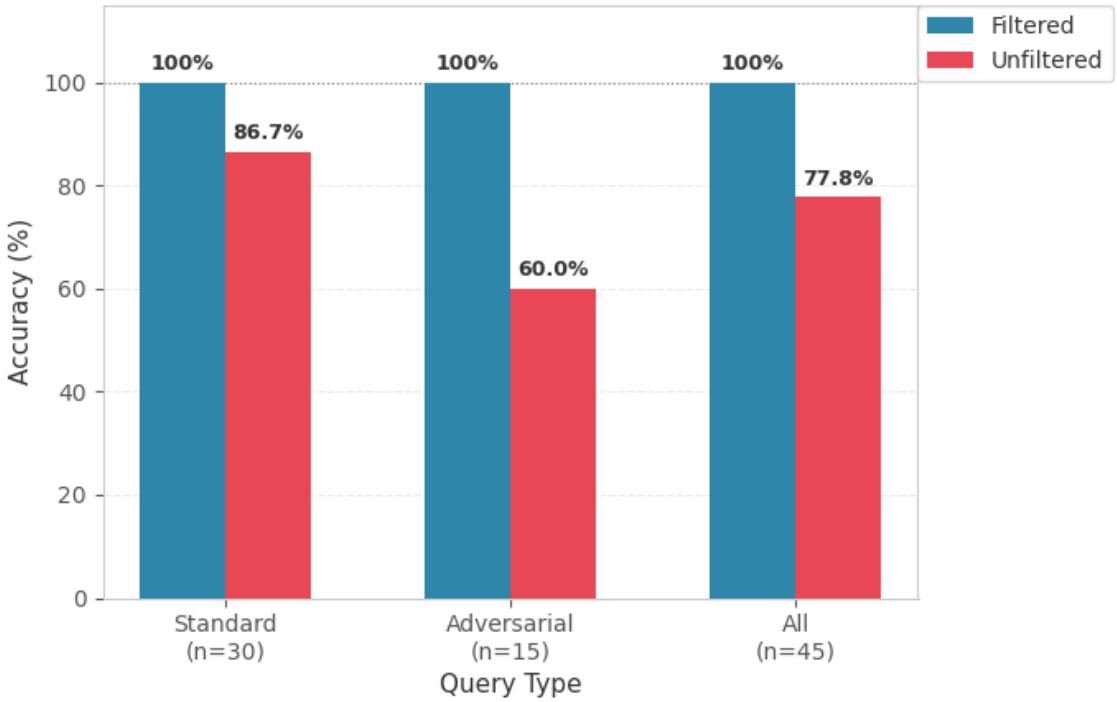

Figure 5.1: RQ2: Retrieval Accuracy.

1. **Standard queries - filtered configuration.**

   All 30 standard test cases were answered correctly under the filtered configuration, showing an accuracy of 100%. No cross-asset or cross-tenant contamination was observed in any response.

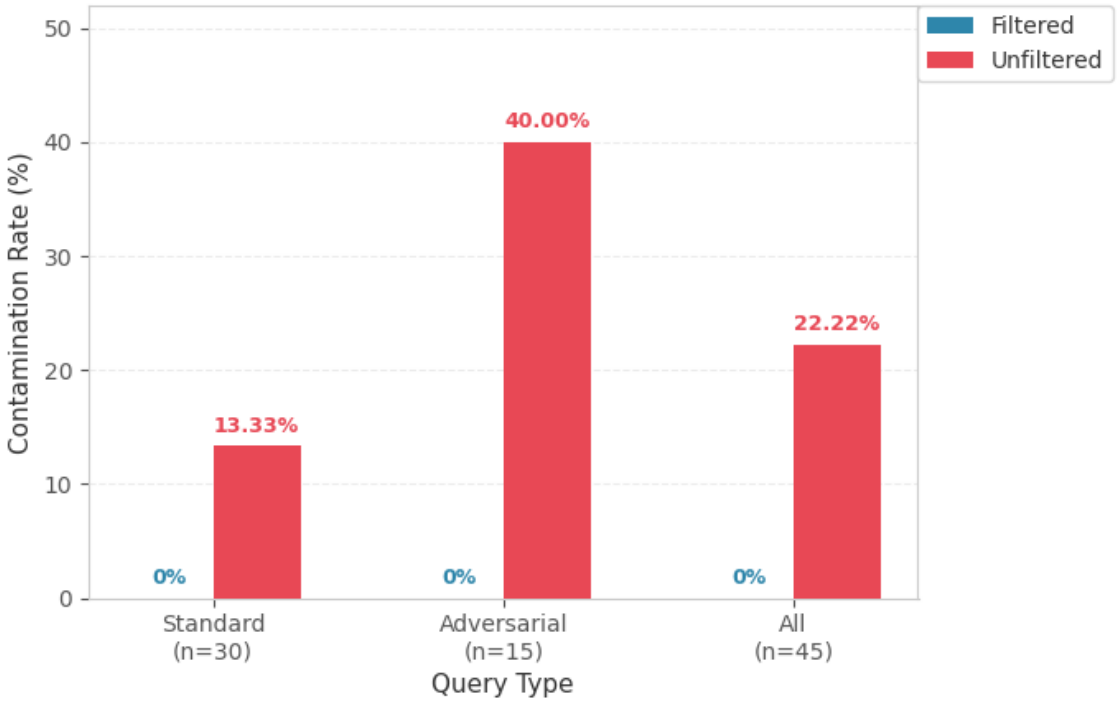

Figure 5.2: RQ2: Contamination Rate.

2. **Standard queries - unfiltered configuration.**

   When filtering was disabled, 4 of the 30 standard test cases produced contaminated responses (contamination rate: 13.33%, see Figure 5.2). Contaminated responses predominantly involved numerical tolerances and performance values drawn from semantically similar but incorrect manuals. Notably, several contaminated responses were fluent and technically plausible, which underscores the operational risk of relying on vector similarity alone, since an incorrect specification value would not be immediately identifiable to a technician unfamiliar with the particular machine.

3. **Adversarial queries - filtered configuration.**

   All 15 adversarial test cases were answered correctly under the filtered configuration. Each response contained the specification value corresponding exclusively to the machine identified by the `asset_id` parameter, confirming

that the metadata constraint holds under semantic pressure from overlapping terminology.

4. **Adversarial queries - unfiltered configuration.**

   When filtering was disabled, the adversarial queries produced a higher contamination rate than the standard suite (40% versus 13.33%, Figure 5.2), consistent with the expectation that shared terminology increases the likelihood of incorrect chunk retrieval when no structural constraint is applied.

5. **Boundary tests.**

   When a mismatched `asset_id` was passed, the system returned no results from the vector index and the language model responded with an appropriate message indicating that no relevant documentation was found for the specified asset. No incorrect machine specifications were returned, confirming that the filter logic does not silently fall back to broader retrieval.

5.4 Evaluation of RQ3: Expert Feedback Integration

RQ3 asks how domain expert feedback can be incorporated into the RAG pipeline to keep machine-specific responses accurate as specifications evolve. The mechanism implemented for this thesis differs from a straightforward chunk-replacement approach - rather than deleting the chunk containing an outdated value, the system augments it by appending a clearly-marked correction section while leaving the original text intact. Alongside this, a follow-up query is executed to retrieve this feedback with maximum priority, ensuring that corrections are emphasized. This design appends expert feedback to the original chunk, preserving its surrounding contextual information that might otherwise be inadvertently overwritten. The

Table 5.5: RQ3 Simulated errors and corresponding expert corrections.

Machine	Specification	Introduced error	Correct value
WaterJet L	Positioning accuracy	±0.05 mm/m	±0.08 mm/m
WaterJet L	Abrasive feeder range	0–400 g/min	0–600 g/min
WaterJet H	Repeatability	±0.020 mm	±0.015 mm
WaterJet H	Z-axis max speed	3 m/min	5 m/min
WaterJet P	Linear motor force	5000 N	7000 N
WaterJet P	X/Y max speed	40/80 m/min	40/100 m/min

evaluation of RQ3 focuses on the observable property of whether corrected facts are actually surfaced in the responses after submission by the expert.

5.4.1 Evaluation Procedure

Expert Feedback Experiment: As shown in table 5.5, six specification values were selected across three machines and deliberately corrupted in the index to simulate entries that had become outdated. Before any correction was applied, queries targeting each value were confirmed to return the wrong value. An expert correction was then submitted for each entry through the `POST /ai/chat/expert-feedback` endpoint. The submission triggers two writes: the matching index chunks are augmented with an `"EXPERT CORRECTION"` section appended to their existing content chunk, and a standalone chunk carrying the expert-provided value is upserted with a unique identifier derived from the `tenant_id` and query topic. In the index database, this standalone chunk is saved with the `doc_type` set to `ExpertFeedback`.

Three metrics were recorded for each scenario:

- **Expert Correction Uptake**: whether the corrected value appears in the response after submission.
- **Conflict Resolution Accuracy**: whether the system consistently returns the corrected value rather than the outdated one during the coexistence window when both the augmented original chunk and the new expert chunk are present in the index simultaneously.

Preservation of surrounding context: To verify that augmentation does not strip unrelated information from the same chunk, each corrected chunk was manually inspected after submission to confirm that the original text preceding the correction marker remained intact.

End-to-end re-evaluation: After all six corrections were applied, the standard 30-query test suite from the RQ1 evaluation was re-executed. This checks whether the feedback mechanism degrades performance on queries that were not targets of any correction.

5.4.2 Results

The following results summarise the effectiveness of the expert feedback integration mechanism in terms of correction uptake, consistency, and impact on end-to-end system performance.

1. **Correction uptake and conflict resolution.**

   Of the 6 correction scenarios, all 6 were answered correctly in the post-correction query, yielding a correction uptake rate of 6/6. Conflict resolution accuracy across was 6/6. Average time-to-correction was around 12 seconds across all successful cases.

2. **Context preservation.**

   Manual inspection confirmed that in all cases the chunk text preceding the correction marker was unchanged after submission. The expert correction section was cleanly appended and clearly delimited, and the surrounding technical content remained unchanged in the retrieved context.

3. **End-to-end re-evaluation.**

Table 5.6: RQ3 Evaluation results.

Experiment	Metric	Result
Correction uptake	Uptake rate	6 / 6
	Conflict resolution accuracy	6 / 6
Context preservation	Chunks with intact original	6 / 6
End-to-end	Corrected queries now correct	6 / 6
	Unaffected queries correct	24 / 24

The 30-query standard suite produced 6 correct responses after feedback integration, compared to outdated failing 6 before. The 6 queries targeting corrected facts now returned the updated values. All remaining 24 queries continued to return correct responses, indicating that the correction mechanism does not interfere with retrieval for unrelated topics. Table 5.6 summarises the results.

6 Conclusion

This chapter presents the final conclusions of the thesis. In particular, it discusses how the three research questions have been addressed, summarizes the main achievements of the work, and reflects on the practical implications and trade-offs of the proposed approach. In addition, the chapter outlines the main limitations of the study and highlights potential directions for future work and further improvements to the proposed solution.

6.1 Evaluation of Research Questions

This thesis addressed three research questions focusing on the design, integration, and continuous improvement of a Retrieval-Augmented Generation (RAG)-based chatbot for industrial machinery troubleshooting.

6.1.1 RQ1 Summary

With respect to RQ1, which explores the architectural design required for seamless integration into existing applications, the study compared a local deployment with a cloud-based implementation. The results indicate that the Azure-based architecture provides substantial advantages in terms of latency, scalability, and maintainability. Both quantitative and qualitative evaluations show that the managed cloud solution enables efficient document handling, reliable model hosting, and smooth workflow coordination, making it more suitable for real-world industrial deployment.

The managed Azure-based architecture demonstrated clear advantages in responsiveness, automatic scaling, and reduced maintenance effort. The use of provider-managed services for storage, indexing, and model inference simplified system integration and enabled stable operation under varying workloads. These characteristics are particularly important in industrial settings, where the chatbot must operate as part of a larger application pipeline involving document ingestion, user session handling, and coordinated workflows.

From a cost perspective, the local deployment may be suitable for small-scale or experimental use, while the cloud-based approach is more appropriate for production systems that require consistent performance and minimal operational overhead. The evaluation therefore suggests that seamless architectural integration is best achieved through a managed cloud deployment, where infrastructure, model hosting, and retrieval services can scale automatically while maintaining predictable performance.

Overall, the findings of RQ1 support the conclusion that the choice of architecture has a direct impact on the practicality of deploying RAG-based chatbots in industrial environments, and that cloud-based solutions provide a more suitable foundation for real-world integration.

6.1.2 RQ2 Summary

In terms of RQ2, the 100% accuracy achieved under the filtered configuration across both standard and adversarial query types confirms that metadata-constrained retrieval effectively prevents cross-contamination in this multi-tenant, multi-asset deployment. The higher contamination rate observed for adversarial queries in the unfiltered configuration is an important finding. The adversarial queries were constructed from specification field names - such as positioning accuracy and repeatability - that appear identically across all three datasheets. Vector similarity is most likely to fail precisely in this scenario, since the query embedding matches all three

manuals equally well and the retrieved chunks may originate from any of them. This represents the realistic case in industrial deployments, where machines from the same manufacturer share a common technical vocabulary but differ in critical numerical values. Metadata filtering is therefore most necessary in exactly those situations where it is hardest to enforce through similarity alone.

The boundary test results confirm that the system fails safely: an unrecognised `asset_id` produces an empty retrieval result and a transparent no-result message, rather than a fallback to an adjacent manual. This is the correct behaviour in safety-sensitive industrial contexts.

These findings directly address RQ2, by demonstrating that machine-specific contextual isolation can be reliably achieved through metadata-constrained retrieval within Azure AI Search, and that the absence of such constraints introduces measurable, operationally significant risks.

6.1.3 RQ3 Summary

For RQ3, concerning the incorporation of structured expert feedback into the RAG framework, the results show that the augmentation-based approach addresses the core concern that motivated this design. By preserving the original text and appending a correction section, the system avoids discarding that context. From the LLM's perspective the prompt still contains the surrounding technical detail, with the expert's value appearing as an explicit override immediately following the original passage.

A practical consequence of the augmentation design is that chunks can grow longer each time a correction is applied to the same document passage. This is a known limitation and would require a chunk-splitting strategy in a production deployment operating at scale.

A second limitation is that the conflict resolution window: which is the brief

period after submission during which both the augmented chunk and the new expert chunk are present but index propagation may not yet be complete, was observed to vary by a few tens of seconds. For safety-critical corrections this latency is non-trivial.

Overall, the evaluation provides evidence that structured expert feedback can be incorporated into the RAG pipeline without disrupting existing retrieval behaviour. This addresses RQ3, in the context of this prototype. Broader generalisation to real-world deployments with multiple concurrent experts, conflicting corrections, and higher query volumes remains an open question for future work.

6.2 Limitations of the Study

Despite the promising results, this study has several limitations. First, the architectural evaluation was conducted using a single cloud provider, namely Microsoft Azure, as it aligns with the company's existing infrastructure. As a result, the findings may not fully generalize to other cloud environments with different service capabilities or cost structures.

Second, the evaluation dataset was based on a limited set of queries generated synthetically with the assistance of large language models. Although care was taken to ensure diversity and relevance, the dataset may not fully capture the complexity and variability of real-world user queries. The absence of a production deployment prevented the collection of authentic user interaction data or large-scale empirical validation.

Third, the validation of generated queries and evaluation results required human involvement. Domain experts from the company reviewed the generated question–answer pairs to ensure correctness and relevance. Additionally, assessment of cross-contamination and response quality involved manual inspection, which introduces subjectivity and limits scalability.

Finally, an additional limitation relates to the integration of expert feedback within the system. In the current implementation, expert corrections do not overwrite existing content but are appended as new entries, forming a separate expert feedback document type within the Azure index. While this approach preserves traceability and avoids loss of original information, it increases the overall size of the indexed data. Over time, this may lead to higher memory consumption and potential increases in retrieval latency. However, the use of hybrid retrieval (combining semantic and keyword-based search) helps mitigate these effects by maintaining efficient and relevant retrieval performance despite the growing index size.

6.3 Future Work

Future work can extend this research in several directions. One key area is the evaluation of the proposed architecture across multiple cloud providers to better understand trade-offs in performance, cost, and portability. This would improve the generalizability of the findings and support broader deployment scenarios.

Another important direction is the integration of real-world user feedback through production deployment. Collecting interaction data from technicians and domain experts would enable more robust evaluation and support the development of automated feedback loops for continuous system improvement. Additionally, incorporating more sophisticated evaluation techniques, including large-scale user studies and statistical validation, would strengthen the reliability of the results.

While the proposed approach to incorporate expert feedback is effectively within the scope of this prototype, its applicability to large-scale real-world deployments remains an open area for further investigation. In particular, future work should explore scenarios involving multiple concurrent domain experts, potentially conflicting feedback, and significantly higher query volumes. Evaluating the system under such conditions would provide deeper insights into its scalability, robustness, and ability

to maintain consistent performance in complex operational environments.

Finally, extending the system to handle more complex multi-turn interactions, cross-system integrations, and proactive maintenance recommendations could further enhance its practical value in industrial environments.

References

[1] M. Vitale et al., "Harnessing generative AI for interactive system failure diagnostics: A user-centric approach to streamlined problem solving and maintenance", in *In Proc. Abu Dhabi International Petroleum Exhibition and Conference (ADIPEC)*, Nov. 2024.

[2] H. Wang and Y.-F. Li, "Large language model empowered by domain-specific knowledge base for industrial equipment operation and maintenance", in *In Proc. 5th International Conference on System Reliability and Safety Engineering (SRSE)*, 2023, pp. 474–479.

[3] S. Jose, K. T. P. Nguyen, K. Medjaher, R. Zemouri, M. Lévesque, and A. Tahan, "Advancing multimodal diagnostics: Integrating industrial textual data and domain knowledge with large language models", *Expert Systems with Applications*, vol. 255, p. 124 603, 2024.

[4] L. G. Di Maggio, "Toward autonomous LLM-based AI agents for predictive maintenance: State of the art, challenges, and future perspectives", *Applied Sciences*, vol. 15, no. 21, p. 11 515, 2025.

[5] B. Chen, "Application of retrieval-augmented generation for interactive industrial knowledge management via a large language model", *Computer Standards and Interfaces*, vol. 94, p. 103 995, 2025.

[6] K. S. Kiangala and Z. Wang, "An experimental hybrid customized AI and generative AI chatbot human machine interface to improve a factory troubleshoot-

ing downtime in the context of Industry 5.0", *The International Journal of Advanced Manufacturing Technology*, vol. 132, no. 5, pp. 2715–2733, 2024.

[7] A. Narimani and S. Klarmann, "Integration of large language models for real-time troubleshooting in industrial environments based on retrieval-augmented generation (RAG)", in *In Proc. 7th European Industrial Engineering and Operations Management Conference*, Jul. 2024.

[8] A. Chandrasekhar, J. Chan, F. Ogoke, O. Ajenifujah, and A. B. Farimani, "AMGPT: A large language model for contextual querying in additive manufacturing", *Additive Manufacturing Letters*, vol. 11, p. 100 232, 2024.

[9] Y. Jin, Y. Wu, W. Hu, B. M. Maggs, X. Zhang, and D. Zhuo, *Curator: Efficient indexing for multi-tenant vector databases*, arXiv:2401.07119. [Online]. Available: `https://arxiv.org/abs/2401.07119`, [Accessed: Feb. 20, 2026].

[10] M. Y. Kurra, "Generative AI agents at enterprise scale: Architecting RAG-enhanced LLM systems for production deployment", *International Journal of Computational Mathematical Ideas (IJCMI)*, vol. 17, no. 1, pp. 17 361–17 377, 2025.

[11] R. Kumar, "Silo, pool, and bridge for multi-tenant RAG: Measuring isolation, noisy-neighbor effects, and cost in SaaS microservices", *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 7, no. 1, pp. 30–47, 2026.

[12] J. Jeyaraman, *Vector Databases Unleashed: Isolating Data in Multi-Tenant LLM Systems*. India: Libertatem Media Private Limited, 2025.

[13] B. Chen and T. Taipalus, "Metadata-aware RAG for enforcing access control and metadata-based filtering: Proof of concept and evaluation", in *In Proc. RACS '25*, New York, NY, USA, 2026, 20:1–20:7.

[14] A. R. Bairi, *Vectorizing the Cloud: Advanced RAG Solutions*. Libertatem Media Private Limited, 2024.

[15] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He, "A survey of human-in-the-loop for machine learning", *Future Generation Computer Systems*, vol. 135, pp. 364–381, Oct. 2022.

[16] X. Liu, S. Qiao, and S.-H. Na, *Retrieval-augmented editing generation: Impact of knowledge editing and fine-tuning on RAG*, 2025. [Online]. Available: `https://openreview.net/forum?id=R2OzZWOkjz`, [Accessed: Dec. 8, 2025].

[17] X. Sun, J. A. Bosch, J. D. Wit, and E. Krahmer, "Human-in-the-loop interaction for continuously improving generative model in conversational agent for behavioral intervention", in *In Companion Proc. 28th International Conference on Intelligent User Interfaces (IUI)*, Sydney, NSW, Australia, 2023, pp. 99–101.

[18] Y. Bai et al., *Pistis-RAG: Enhancing retrieval-augmented generation with human feedback*, arXiv:2407.00072. [Online]. Available: `https://arxiv.org/abs/2407.00072`, [Accessed: Apr. 8, 2026].

[19] M. Mahbub et al., *ORCHID: Orchestrated retrieval-augmented classification with human-in-the-loop intelligent decision-making for high-risk property*, arXiv:2511.0495 [Online]. Available: `https://arxiv.org/abs/2511.04956`, [Accessed: Dec. 8, 2025].

[20] S. Nasir, Q. Abbas, S. Bai, and R. A. Khan, *A comprehensive framework for reliable legal AI: Combining specialized expert systems and adaptive refinement*, arXiv:2412.20468. [Online]. Available: `https://arxiv.org/abs/2412.20468`, [Accessed: Apr. 6, 2026].

[21] S. Kirtania et al., *STACKFEED: Structured textual actor-critic knowledge base editing with feedback*, arXiv:2410.10584. [Online]. Available: https://arxiv.org/abs/2410.10584, [Accessed: Dec. 8, 2025].

[22] Y. Lian, *Machine assistant with reliable knowledge: Enhancing student learning via RAG-based retrieval*, arXiv:2506.23026. [Online]. Available: https://arxiv.org/abs/2506.23026, [Accessed: Nov. 8, 2025].

[23] P. Pattnayak, A. Agarwal, H. Meghwani, H. L. Patel, and S. Panda, *Hybrid AI for responsive multi-turn online conversations with novel dynamic routing and feedback adaptation*, arXiv:2506.02097. [Online]. Available: https://arxiv.org/abs/2506.02097, [Accessed: Apr. 6, 2026].

[24] B. Tarun, H. Du, D. Kannan, and E. F. Gehringer, *Human-in-the-loop systems for adaptive learning using generative AI*, arXiv:2508.11062. [Online]. Available: https://arxiv.org/abs/2508.11062, [Accessed: Apr. 6, 2026].

[25] Y. He et al., *Enabling self-improving agents to learn at test time with human-in-the-loop guidance*, arXiv:2507.17131. [Online]. Available: https://arxiv.org/abs/2507.17131, [Accessed: Apr. 8, 2026].

[26] Y. Chu, H. Li, K. Yang, Y. Copur-Gencturk, and J. Tang, *LLM-based automated grading with human-in-the-loop*, arXiv:2504.05239. [Online]. Available: https://arxiv.org/abs/2504.05239, [Accessed: Apr. 8, 2026].

[27] X. Yang, R. Zhan, D. F. Wong, J. Wu, and L. S. Chao, *Human-in-the-loop machine translation with large language model*, arXiv:2310.08908. [Online]. Available: https://arxiv.org/abs/2310.08908, [Accessed: Apr. 8, 2026].

[28] Z. Cai, B. Chang, and W. Han, *Human-in-the-loop through chain-of-thought*, arXiv:2306.07932. [Online]. Available: https://arxiv.org/abs/2306.07932, [Accessed: Apr. 8, 2026].

[29] Promptfoo Contributors, *Introduction to Promptfoo: Test-driven prompt engineering*, [Online]. Available: `https://www.promptfoo.dev/docs/intro/`, [Accessed: Mar. 2, 2026], 2023.

[30] K. Thirumalaisamy, "Survey of public red-teaming frameworks for LLM: Techniques, coverage, and gaps", *International Journal of Science and Research (IJSR)*, vol. 14, no. 12, pp. 1788–1793, Dec. 2025.