Leveraging Generative AI for Predictive Maintenance:

Building a Knowledge Base for Fault Diagnosis

Master's Program in Systems, Control and Mechatronics
Master's Program in Complex Adaptive System

XINYING WANG
XIAOYING LIU

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Leveraging Generative AI for Predictive Maintenance:

Building a Knowledge Base for Fault Diagnosis

XINYING WANG
XIAOYING LIU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials science
Division of Production Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Leveraging Generative AI for Predictive Maintenance:
Building a Knowledge Base for Fault Diagnosis
XINYING WANG
XIAOYING LIU

Supervisor: Siyuan Chen, Chalmers University of Technology
Examiner: Anders Skoogh, Chalmers University of Technology

Master's Thesis 2025
Department of Industrial and Materials Science
Division of Production Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: System architecture diagram illustrating the integration of manual documents, event logs, and synthetic sensor data into a retrieval-augmented generation pipeline for predictive maintenance using Large Language Model.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Abstract

Fault diagnosis is a complex challenge for industrial production. This thesis develops and evaluates a predictive maintenance assistant integrating large language models (LLM) with retrieved-generation techniques (RAG). By constructing a unified knowledge base comprising sensor data, event logs, and equipment manuals, the system enhances fault diagnosis in industrial settings.
The system analyzes sensor data and links it to event logs, matching sensor data with faults. Meanwhile, it connects faults with equipment manuals via RAG, forming a unified knowledge framework. It generates readable and accurate fault diagnostics via LLMs and searching relevant technical documents. It is adaptive, transferable, and capable of integrating specific knowledge.
Experiments conducted using a simulated drone assembly production line demonstrate significant improvements in diagnostic accuracy, interpretability, and reliability, effectively addressing common issues such as hallucinations and unsupported claims found in traditional LLM applications. The findings highlight the practical feasibility of deploying advanced AI-driven predictive maintenance solutions, emphasizing the importance of semantic richness and structured knowledge integration.

Keywords: Predictive Maintenance, Large Language Models, Retrieval-Augmented Generation, Knowledge Base, Fault Diagnosis, Industrial AI, Data Integration, Knowledge Graph.

Acknowledgements

We would like to express our deepest gratitude to our supervisor, Siyuan Chen, and examiner, Anders Skoogh, for the continuous support, valuable feedback, and insightful guidance throughout the course of this thesis.
We also wish to thank Silvan Marti, whose advice and encouragement greatly helped us in shaping our work.
Our heartfelt thanks go to our families and friends for their constant support, understanding, and motivation during this demanding period.
Finally, we are sincerely thankful for the collaboration between us as co-authors — working together has made this thesis not only more manageable, but also more enjoyable and rewarding.

Xinying Wang and Xiaoying Liu, Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis:

AI	artificial intelligence
LLM	Large Language Model
RAG	Retrieval Augmented Generation
PdM	Predictive maintenance
KAG	Knowledge Augmented Generation
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
MoE	Mixture-of-Experts
RL	Reinforcement Learning
KG	Knowlege Graph

Contents

List of Figures

List of Tables

1
Introduction

This chapter introduces the conceptual framework and research objectives of the thesis, which studies the assistance of smart decision making in industrial predictive maintenance with artificial intelligence (AI). It focuses on the synergistic integration of generative AI technologies, specifically large language models (LLM), with retrieval augmented generation (RAG) techniques to achieve smart decision making.

1.1 Background

In industrial production, to keep high efficiency and safety in operation, it is crucial to conduct equipment failure diagnosis and maintenance. In the era of Industry 4.0, the data-driven predictive maintenance (PdM) proves to be more effective compared with conventional fixed schedules and breakdowns waiting. It excels at promptly forecasting equipment failures and scheduling maintenance[1]. Modern PdM systems constantly monitor asset health via some physical parameters such as vibration, temperature, or acoustic sensors. These data can be subsequently used for analytics to detect anomalies and predict future failures[2]. In this manner, PdM optimizes the performance and improves the lifespan of equipment, thus reducing unplanned downtime and maintenance costs[3]. In practice, it benefits the technicians by providing real-time information about potential equipment faults [4]. The data-driven approach contrasts with traditional preventive schedules by its sensitive reaction to the actual condition of assets[5]. However, in most cases, the automotive decision making of PdM is restricted by various types of uncurated data sources and knowledge that are stored in unstructured documents, which limits [6].

Most recently, several studies reported that AI techniques (from machine learning to today's generative AI) enable more advanced PdM solutions. AI assists PdM by fusing rich sensor and diagnostic data, which further improves machine uptime and reliability. As an example, PdM has been successfully applied in the automotive industry, by predicting component breakdowns and scheduling pre-failure repairs based on analysis of vehicle sensor data. occur[7][8].Recent advances in generative AI, notably LLMs, suggest new possibilities for improvement of PdM. Despite originally designed for natural language processing (NLP), LLMs can be repurposed for applications in industrial settings[9]. For example, LLMs can be utilized to process unstructured maintenance logs or technical manuals and subsequently generate human-readable notes. LLMs have already been applied in the diagnostics and predictive maintenance tasks in the automotive sector. Moreover, generative

models like LLMs can fuse sensor streams and textual knowledge (e.g. past failure reports) to produce a more comprehensive maintenance schedule. However, LLMs alone have several limitations that may hamper the industrial application. For the most important one, they tend to "hallucinate" facts or get trapped by date domain specific knowledge. For application in the maritime industry, LLMs often require considerable modifications to meet the specific demands[10].

To address the limitations listed above, retrieval augmented generation was proposed, which combines a standard LLM with a retrieval step. In RAG, before generating an answer, the LLM "looks up" relevant information from an external knowledge base or document set [11]. As for application in industry, the external queries become the database of equipment manuals, maintenance logs, or sensor archives, which can be utilized to refine a PdM system with RAG. This ensures that the decision of LLM is grounded in real technical data.[9]. This makes RAG especially an attractive scheme to deploy LLM for predictive maintenance. Notably, RAG allows the model to incorporate the latest domain knowledge (e.g., component specifications or historical failure modes) without the need for retraining. Besides RAG, knowledge augmented generation models have been proposed more recently, which further integrate structured domain knowledge (such as knowledge graphs) with LLMs[12]. In contrast with RAG, which utilizes unstructured data, KAG combines the reasoning capabilities of knowledge graphs, such as numerical relations, rules, and temporal logic, with the language understanding of LLMs. The combination of LLM with RAG or KAG approaches is a promising way to fuse sensor data, engineering knowledge, and natural language, building the theoretical foundation for this study[13].

1.2 Purpose

The purpose of this research is to develop and evaluate a novel model that integrates a large language model with retrieval augmented generation techniques for industrial fault diagnosis. The model aim to enhance maintenance decision making by enabling more accurate fault diagnosis, context rich diagnostics, and proactive recommendations tailored to an industrial setting. By combining an LLM's reasoning ability with domain specific knowledge retrieval, the model provides maintenance engineers with interpretable and trustworthy insights that improve operational reliability while respecting practical constraints such as data privacy and limited computing resources.

1.3 Aims

This thesis aims to construct an end to end pipeline that converts troubleshooting logs and sensor measurements into readable maintenance reports, and deploy an interactive LLM based assistant that ingests those reports along with manual book documents to provide actionable diagnostics.

1.4 Research Question

This thesis seeks to answer the following research questions (RQs):

- RQ1: How can a comprehensive knowledge database be constructed by integrating heterogeneous data sources from various types of machines for application of generative AI?
- RQ2: How can large language models and retrieval-augmented generation techniques be effectively integrated to enhance fault diagnosis support in complex systems?

1.5 Scope and Delimitations

This research was conducted over a brief period (February–June 2025), imposing practical constraints on experiment scope and system development. As a result, certain design decisions (e.g., model selection and parameter tuning) had to be made under time pressure, foregoing extended iterative testing or optimizations.

A key challenge was the unavailability of real-world industrial event logs due to data access and confidentiality issues. To mitigate this, synthetic event logs were generated using Siemens Plant Simulation to approximate factory operations. However, the synthetic events from the simulation did not directly correspond to the structure or semantic context of the real measurements and required preprocessing to be mapped.

Another limitation arises from the computing environment typical of many factories. Industrial plants often rely on modest hardware. This constraint limits the size and complexity of deployable language and retrieval models. Designing for these modest resources requires trade-offs between model performance and computational cost, prioritizing lightweight architectures and optimized workflows.

1.6 Outline

This thesis is organized into six main chapters, followed by appendices, to systematically address the research objectives. Chapter 1 introduces the research, outlining the background, problem statement, purpose, aims, research questions, and scope. Chapter 2 provides a comprehensive theoretical background, reviewing key concepts in predictive maintenance, Large Language Models, Retrieval-Augmented Generation, Knowledge-Augmented Generation, the LangChain framework, and the current state-of-the-art in LLMs for predictive maintenance. Chapter 3 details the methodology, describing the system overview, the overall research framework, data collection and preprocessing techniques for equipment manuals, event logs, and sensor data,

and the implementation of the RAG and KAG pipelines. Chapter 4 presents the results of the study, including findings on data integration and preprocessing, the performance evaluation of the RAG and KAG systems, and a comparative analysis of their effectiveness. Chapter 5 discusses these results, providing interpretations, answering the research questions, considering the implications and limitations of the study, and suggesting directions for future work. Finally, Chapter 6 concludes the thesis by summarizing the key findings, reiterating the contributions to the field of AI-driven predictive maintenance, and highlighting the overall significance of the research. The appendices offer supplementary materials, including system configurations, illustrative examples, and data processing code.

2

Theoretical Background

2.1 Maintenance

In the industrial context, maintenance is broadly defined as "the combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function"[14]. While maintenance inherently encompasses repairs, this definition highlights the proactive measures to keep equipment operating soundly and safely. Modern industrial maintenance strategies are typically categorized as corrective or preventive maintenance. The former refers to reactive fixes after failures, and the latter refers to scheduled maintenance to avert failures. Recently, there has been a growing emphasis on predictive maintenance. It is a proactive approach using condition monitoring and data analytics to predict equipment failures early[15]. By constantly acquiring sensor data and assessing machine health, predictive maintenance schedules interventions just-in-time. It minimizes unplanned downtime and maintenance costs, thus maximizing asset availability.

Figure 2.1 A schematic comparison of reactive (action only after failure), periodic preventive (scheduled actions), and data-driven predictive (intervene based on health estimates) approaches.
The timeline diagram (Figure 2.1) displays how reactive fixes follow a failure event, preventive tasks recur at fixed intervals, and predictive maintenance leverages sensing (iconized by the magnifier) to schedule repairs before failure. These strategies form a continuum balancing downtime risk and cost.

2.1.1 Current Practices

In predictive maintenance, a key component is effective fault diagnosis, which generally consists the following four stages[17]: (1) fault detection – recognizing the occurrence of a deviation or anomaly indicative of a fault event; (2) fault isolation – pinpointing the originating component or subsystem that causes the fault event; (3) fault identification – determining the exact nature and characteristics of the fault, usually including type, size, and timing; and (4) fault evaluation – assessing the severity and implications of the fault, especially its potential effect on system performance or lifespan.

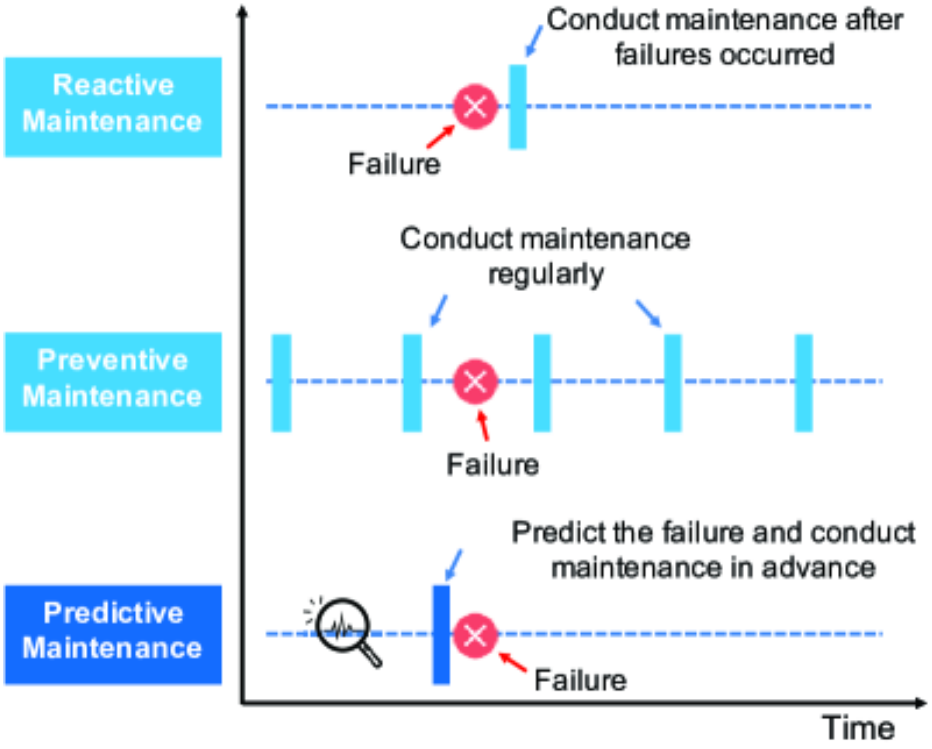


Figure 2.1: Schematic timeline of Reactive, Preventive and Predictive maintenance actions relative to machine operation. [16].

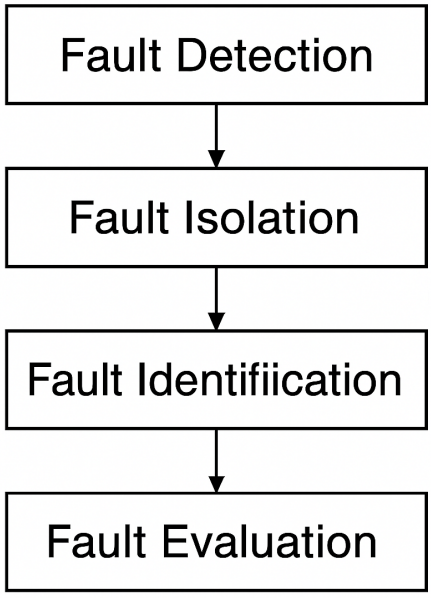


Figure 2.2: Flowchart illustrating the sequential stages in fault diagnosis.

For decades, fault diagnosis has mainly employed analytical and signal-processing methods. Model-based (analytical) approaches build mathematical models of system behavior. Typical methods include observers or parameter estimation schemes that analyze the differences between measured and expected signals as the criterion to detect anomalies[18][19]. In contrast, signal-based methods (data-driven or feature-based) explore the statistics in raw sensor data to extract fault signatures. Representative analysis tools in signal-based methods include time-frequency analysis, spectrum analysis, and wavelet transforms. Besides the two methods, emerging knowledge-based techniques encode expert rules or use fuzzy logic to reason about faults. Ji et al. systematically summarized and compared the three mainstream diagnostic methods mentioned above. For well-characterized systems, analytical models and classical signal-processing methods function effectively. However, when dealing with nonlinear systems and high-dimensional data, the two traditional methods would be incompetent. The reason is that they typically focus on specific fault patterns without extensive domain knowledge. This problem can potentially be solved by knowledge-based methods. [20][21].

2.1.2 AI in Fault Diagnosis

Over the past two decades, machine learning (ML)[22] and deep learning (DL) have transformed predictive maintenance and fault diagnosis. Data-driven ML models such as support vector machines, random forests, and gradient boosting are successfully applied in fault diagnosis. After training on historical sensor and operational data, these models can recognize fault patterns or predict failures with high accuracy[20]. More advanced deep neural networks learn hierarchical features from raw time-series data seamlessly and significantly improve the performance on large datasets. Serradilla et al. reasoned that the prevalence of DL methods in predictive maintenance as their capabilities in handling the growing volume of industrial data. This is particularly beneficial for addressing tasks such as anomaly detection, root-cause analysis, and remaining lifespan. [23]. These AI-based models have achieved state-of-the-art results across several fault diagnosis domains. However, they require a large amount of labeled data and lack interpretability. Consequently, researchers have been exploring hybrid and explainable approaches combining physical models with learning algorithms, to address the problem mentioned above[24].

In recent years, large language models were proposed, which are basically very large neural networks pretrained on vast corpora. LLMs have opened new frontiers for AI-assisted maintenance. Initially, LLMs designed for the language task were not a directly applicable tool for sensor-based diagnostics. However, their ability to capture complex patterns and knowledge has sparked exploration in maintenance contexts. Recently, researchers have begun to fine-tune or modify LLMs for specific fault diagnosis tasks. LLMs were even fed with time-series sensor readings by encoding signals as text tokens[25]. Early results show that certain LLMs can achieve competitive or even better fault detection accuracy compared with specialized deep learning. The advantages of LLMs originate from their capacity to integrate multiple data sources

and contextual information[26]. Moreover, LLMs are more human-friendly, as they produce intelligible maintenance reports and anomaly interpretations. This greatly assists engineers in the industry[27]. This trend paves a broader road toward AI-augmented maintenance, where advanced models provide both the fault prediction and its explanation and contextualization.

Nonetheless, the application of LLMs for industrial maintenance is still immature. The solution of key challenges like adapting them to specific domains and ensuring reliable, trustworthy outputs needs more effort. These issues drive the work presented in the rest of this thesis.

2.2 Large Language Models

Large Language Models have emerged from decades of natural language processing (NLP) research. Early work at the mid-20th century developed statistical language models (LMs) such as n-grams to predict word sequences[28]. The 2000s saw neural LMs, using embeddings and recurrent neural networks (RNNs) [29][30] to learn word probability distributions. A watershed was the Transformer architecture, mainly composed of the attention operation that enabled massive pre-trained LMs [31][32]. Pre-training on web-scale text and fine-tuning on tasks (the PLM paradigm) produced flexible models. Starting around 2018, OpenAI's GPT series exemplified the "foundation model" approach: GPT-1 and GPT-2 showed the value of scaling law, and the subsequent GPT-3 (175B parameters) demonstrated few-shot learning. More recently, GPT-4 (Mar 2023) and other similar models integrate data on a larger scale and reach near-human proficiency on many benchmarks[33]. In summary, LLMs trace a path from statistical n-grams in the 1990s through neural architecture like RNNs in the early 2000s, to modern Transformer-based models since 2017, culminating in today's large pretrained models. These models can generate coherent text and follow complex instructions[34].

2.2.1 Application Areas of LLMs

LLMs now power a wide range of tasks[34]. Key applications include the following areas. Text Generation and Dialogue: Generative chatbots (e.g., ChatGPT) can compose essays, summaries, and converse with users. They excel at tasks like question answering and text summarization[35]. Knowledge Reasoning: Advanced LLMs perform multi-step reasoning on complex questions. For example, GPT-4 scores at the top 10% on a simulated bar exam[33], and models can solve logic/math problems via chain-of-thought prompting[36]. Coding Assistance: Models fine-tuned on code (e.g., OpenAI's Codex) translate natural language to programs. Codex (derived from GPT-3) powers GitHub Copilot and can generate code snippets from prompts[37]. Specialized Domains [38]: LLMs assist in technical fields by summarizing information or drafting reports. They can also handle multimodal tasks (processing images alongside text in models like GPT-4[33] or Google's Gemini[39]).

These capabilities derive from pretraining on vast datasets and the models' generality, enabling them to be adapted to diverse tasks by prompt design or fine-tuning.

2.2.2 Comparative Overview of Selected LLMs

Table 2.1: Major LLMs (until 2025): Metadata Summary

Model	Release Date	Company	Country	Type	Open Source?
GPT-4	Mar 2023	OpenAI	USA	General	No (closed)
Llama 3	Apr 2024	Meta	USA	General	Yes (open)
DeepSeek-R1	Jan 2025	DeepSeek-AI	China	Specialized (reasoning)	Yes (MIT License)
Claude 3	Mar 2024	Anthropic	USA	General	No (closed)
Gemini	Dec 2023	Google DeepMind	USA	Multimodal	No (closed)
PaLM 2	May 2023	Google	USA	General	No (closed)
Mixtral	Dec 2023	Mistral AI	France	Sparse MoE	Yes (Apache 2.0)
Qwen 3	Apr 2025	Alibaba Cloud	China	General	Yes (Apache License)
Falcon	Mar–Nov 2023	TII	UAE	General	Yes (permissive)

Table 2.2: Major LLMs (until 2025): Technical Capabilities and Performance

Model	Main Capabilities	Reasoning Ability	Accuracy / Benchmarks
GPT-4	Multimodal input (text + image); broad NLP skills	Very high – near-human level[33]	SOTA on several benchmarks (e.g., bar exam top 10%)
Llama 3	Text generation, coding, long-context reasoning	Improved over Llama 2[40]	Matches GPT-4 on many tasks
DeepSeek-R1	Reasoning, math, code	Very high – comparable to GPT-4[41]	GPT-4 level on logical reasoning and coding[41]
Claude 3	Content creation, safe alignment, document analysis	Very high – human-level on MMLU[42]	SOTA on many expert-level tasks

Model	Main Capabilities	Reasoning Ability	Accuracy / Benchmarks
Gemini	Multimodal (text, image, audio); multilingual	Very high– 90%+ on MMLU[39]	Surpasses GPT-4 on some benchmarks
PaLM 2	Multilingual understanding, code, logic/math	High – expert-level logic[43]	Top on many language/coding tasks
Mixtral	Long-context, multilingual, sparse MoE	High – exceeds GPT-3.5[44]	Strong cost/performance tradeoff
Qwen 3	Code, math, multilingual (119 langs)	High – strong on reasoning[45]	Best open-source on many leaderboards
Falcon	Large-scale generation, open-source tuning	High – near GPT-4 level[46]	Among top-performing open models

2.2.3 Model Selection: Llama3 and DeepSeek-R1

We select Llama 3 and DeepSeek-R1 in this study, considering their complementary advantages. Both models are open-source, enabling inspection and on-premise deployment. Meta's Llama 3 (8B or 70B parameters) demonstrates enhanced reasoning and coding abilities[**noone2024Llama3**] and is released with permissive licenses for research use. It serves as a strong general-purpose model with multi-task capabilities. DeepSeek-R1 is explicitly designed for enhancing logical reasoning. It achieves competitive performance with GPT-4-level models on math and coding tasks[41] and is released under an MIT license. Moreover, DeepSeek-R1 can be distilled into much smaller sub-models (e.g., 7B or 14B parameters) with uncompromised chain-of-thought ability, which is crucial in our study. Therefore, the combination of Llama 3 and DeepSeek-R1 provides a balance of high reasoning accuracy and lightweight deployment. These features are suitable for fine-tuning and embedding in edge or industrial systems, which this study focuses on.

2.2.4 Introduction of Deepseek

DeepSeek-R1 is a large Mixture-of-Experts (MoE) language model built on the DeepSeek-V3 base[47]. It contains 671B total parameters, of which 37B are active per token generation[41]. The model is composed of a Transformer architecture with multiple expert sub-networks, enabling parameter up-scaling while only activating a subset of experts for each input. This design yields a very high capacity (with up to 671B learned weights) but keeps a high inference efficiency (computation is on 37B parameters per token).

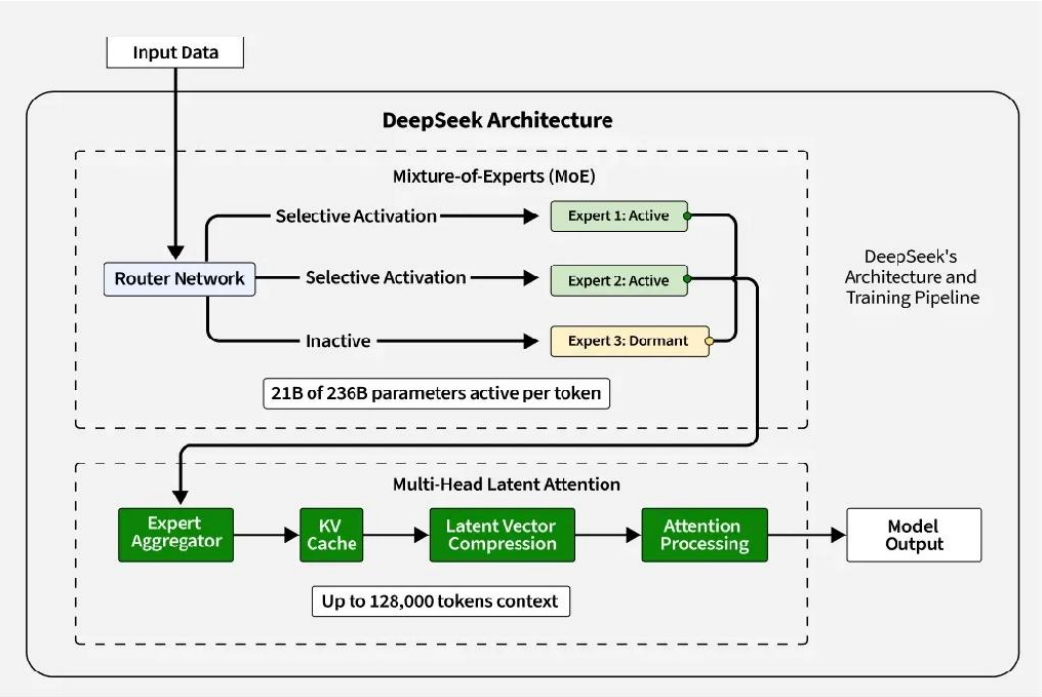


Figure 2.3: DeepSeek-R1 architecture overview. The model uses a Mixture-of-Experts (MoE) routing structure, where only a subset of expert networks are activated per input. This reduces compute while maintaining high model capacity.

DeepSeek-R1 supports an extended context window of 128K tokens, which is achieved by a two-phase context length expansion after pre-training. This feature allows it to process long documents or dialogues[48]. It was pre-trained on a massive multi-domain corpus on the order of 14–15T tokens, and then underwent a specialized multi-stage training pipeline emphasizing reinforcement learning (RL) to enhance reasoning.

Notably, DeepSeek-R1-Zero was first trained purely with RL without supervised fine-tuning to encourage chain-of-thought reasoning. Subsequently, DeepSeek-R1 incorporated a "cold start" phase with curated data before the RL fine-tuning stage. The use of RL with rule-based rewards for accuracy, format, and language consistency led to emergent behaviors like self-verification and long reasoning traces. DeepSeek-R1 achieves state-of-the-art performance on complex tasks comparable to OpenAI's proprietary models (OpenAI-o1 series)[49].

In summary, DeepSeek-R1 leverages a massive MoE Transformer with hundreds of billions of parameters, 61 layers of experts. It is augmented by RL-based training, yielding a model highly specialized for logical reasoning and problem-solving. Figures 2.3 and 2.4 illustrate the DeepSeek architecture and training pipeline, respectively, highlighting its MoE layers and RL fine-tuning stages.

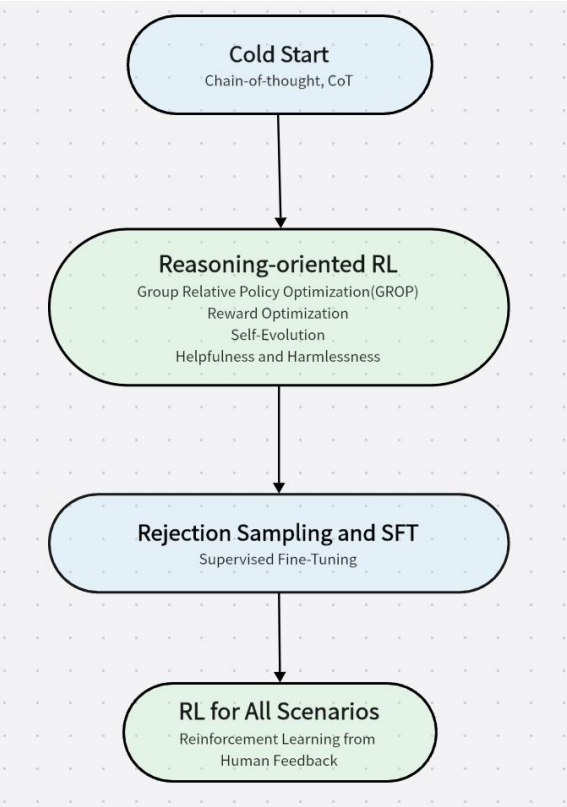


Figure 2.4: DeepSeek-R1 training pipeline. The multi-stage training includes Cold Start, Reasoning-oriented RL, supervised fine-tuning (SFT), and RLHF across scenarios.

2.2.5 Introduction of Llama3

Llama3 is Meta AI's third-generation LLM, designed as a dense Transformer without MoE to maximize training stability. The largest Llama3 model has 405B parameters and was trained with a context window up to 128K tokens. Its capacity is a significant jump from Llama2's 4K context[**meta2024Llama3**].

Its architecture is a decoder-only Transformer with 126 layers, a model dimension of 16,384, and 128 attention heads. Unlike MoE models, Llama 3's dense architecture activates all the weights for each token, which requires a massive computational cost and advanced parallelism techniques for training. To train Llama3, thousands of GPUs across multiple pods were used.

Llama 3 was pre-trained on a diverse, large-scale corpus of about 15.6T tokens of text that includes multilingual data and code[**grattafiori2024Llama**]. An initial training with 8K token sequences was followed by a continued pre-training stage to extend context to 128K tokens. During this stage, a strategy that gradually adapts the model to longer sequences without loss of performance was applied. The model was then "post-trained" with supervised fine-tuning, preference alignment (e.g., direct preference optimization), and instruction tuning, as well as tool-use integration for issues like code execution and reasoning.

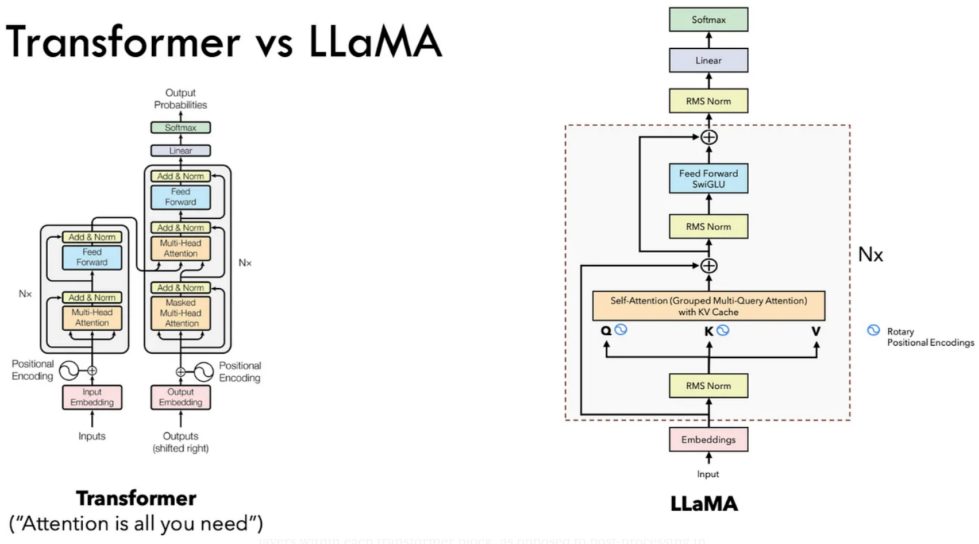


Figure 2.5: Comparison of Transformer vs Llama-3 architecture. Llama replaces softmax normalization with RMSNorm, adds SwiGLU feedforward blocks, and uses rotary positional embeddings with grouped attention for scalability.[50]

Llama 3 primarily aims at multilingual communication support, high-quality coding assistance, reasoning ability, and even basic tool use (via text-based tool APIs) out-of-the-box of Huggingface. Indeed, evaluations show Llama 3.1's 405B-parameter model achieves performance on par with GPT-4 across a broad range of tasks. It can solve complex math and logic problems, answer in multiple languages, and follow user instructions effectively[51].

2.3 Introduction of Retrieval-Augmented Generation

Retrieval Augmented Generation is a framework that combines LLMs with external knowledge bases to improve the accuracy and knowledge richness of the generated results[52]. RAG models consist of parameterized and non-parameterized memory, with the former denoting the knowledge learned by the model itself and the latter denoting external knowledge bases that can be queried[53].

Specifically, RAG introduces a retriever component into the language text generation process. In the retrieval process, after receiving user input such as a question, the model retrieves relevant information fragments from external knowledge sources such as document collections or databases. Then, the external knowledge is provided as an additional context to the generation model. With the retriever scheme, RAG is able to break the limitations of LLMs that solely rely on the implicit knowledge of model parameters. RAG enables the incorporation of the latest, domain-specific,

or more detailed explicit knowledge into the answer generation process.

In practice, this retrieval-generation fusion architecture demonstrates several significant advantages: (1) Reduction of hallucinations: Since the model references retrieved real data during generation, the risk of fabricating false content termed as "hallucinations" is significantly reduced[54]; (2) Enhancement of the controllability: Retrieved evidence fragments can serve as the basis and references for the model's responses, thereby providing traceability and trustworthiness of results; (3) Facilitation of the knowledge updating: Non-parameterized knowledge bases can be dynamically expanded or modified, and models can acquire new knowledge without retraining; (4) Reduction model size: With external knowledge support, even smaller LMs can perform complex tasks in specific domains since the required information can be found externally[55].

2.3.1 Development of Retrieval Augmented Generation

The idea of RAG originates from the gradual evolution of the "retrieval-reading" paradigm in tasks such as open-domain question answering. Early open-domain question-answering systems (also called Open-Domain QA) typically adopted a two-stage framework. First, information retrieval techniques were used to find relevant document segments from large-scale corpora (e.g., Wikipedia). Afterwards, reading comprehension models were employed to extract or generate answers from these segments. As a classic example, Chen et al. proposed the DrQA system in 2017. It uses TF-IDF retrieval to obtain candidate Wikipedia article paragraphs. Then, a neural network reader is used to locate the answer text[56]. Such systems have demonstrated the effectiveness of combining retrieval with language understanding models. Subsequently, with the rise of pre-trained LMs, researchers began to explore end-to-end training that combines pre-trained question-answering models with differentiable retrieval modules. For example, the ORQA[57] and REALM[58] models that appeared in 2019-2020 integrated pre-trained models such as BERT with semantic vector retrieval, enabling the model to update its internal representations during the retrieval process. They achieve better results in open-domain question answering. Afterwards, RAG was formally proposed by Lewis et al. in 2020[52].

Compared to previous works, RAG integrates generative pre-trained models with retrievers for the first time to handle knowledge-intensive generative tasks. RAG models use pre-trained sequence-to-sequence generation models as "parameterized" memory and dense vector indexes pre-built from large-scale text corpora as "non-parameterized" memory. The neural retriever functions through the calculation of the inner product similarity. Lewis et al. incorporated the retrieved documents as latent variables into the generation process through a probabilistic model. They achieved state-of-the-art results on tasks such as natural question answering, dialogue, and generating yearbooks. This series of advancements laid the foundation for modern RAG technology: under the "large model + retrieval" paradigm, by flexibly incorporating external knowledge, the model's performance in open-domain and domain-specific question-answering tasks was significantly improved.

2.3.2 Multiple Applications of Retrieval Augmented Generation

Medical question-answering and clinical decision support are key applications of RAG. For example, Mashatian et al. developed a conversational assistant for diabetes foot care in 2024. They deployed a RAG model that provides patients with medical knowledge question-answering services[58]. The system utilizes a retrieval module to extract relevant facts based on the data source of medical literature and guidelines. This information is then processed by a large-scale generative model to generate clear and understandable explanatory answers for patients. By incorporating evidence retrieval, the assistant can provide more accurate and contextually relevant medical advice compared with the previous methods. It mitigates the risk of the model's incorrect feedback of medical information based solely on training memories. Experiments demonstrate that compared to models without external knowledge, RAG models can provide more reliable medical knowledge to non-expert users in a more controllable manner.

Legal scenarios demand extremely high accuracy and answers grounded on evidence, making RAG a suitable candidate for applications such as question-answering and assistants in the legal context. LegalBench-RAG benchmark [59] constructed a searchable knowledge base of laws, regulations, and case law, and combined it with a generative legal LLM. This enables the system to locate precise legal provisions or case law references when answering legal queries and generate conclusive answers with citations. Such systems significantly reduced the probability of large models misquoting legal provisions or making unfounded assumptions, ensuring that answers are both legally grounded and explanatory. Additionally, retrieving fine-grained legal text fragments rather than entire documents can help reduce forgetting or confusion caused by overly long context while maintaining contextual clarity.

Besides, RAG has also demonstrated significant potential in the industrial sector for predictive maintenance and knowledge management. Conventionally, engineers often need to quickly retrieve the most critical information from extensive technical manuals and maintenance records to maintain and diagnose complex equipment. The RAG model has greatly assisted and simplified this process. For example, a smart maintenance question-answering system at a nuclear power plant employs an RAG architecture for maintenance assistance[60]. It is built by converting a large volume of internal maintenance documents into embedded vectors, which are then indexed offline. The system's retrieval module extracts relevant sections from the knowledge base if inquired, releasing on-duty engineers from complex selection. Subsequently, an LLM is employed to generate professional responses. This system works by combining a privately deployed large model with local document retrieval. This approach eliminates the risk of exposing sensitive data to external APIs. Simultaneously, this approach bridges knowledge gaps arising from the attrition of expert experience. In a practical industry context, when compared to conventional methods reliant on manual document searches, the integration of RAG into maintenance assistance systems greatly reduces the total response times. By delivering

solutions grounded in verifiable references, it bolsters the reliability of outcomes while improving the efficiency of intelligent decision-making processes..

2.3.3 Retrieval Augmented Generation's System Structure

The RAG system typically consists of several functional modules that collaborate to complete the retrieval and generation process. The overall process of retrieval-generation collaboration is vividly shown in Figure 2.6.

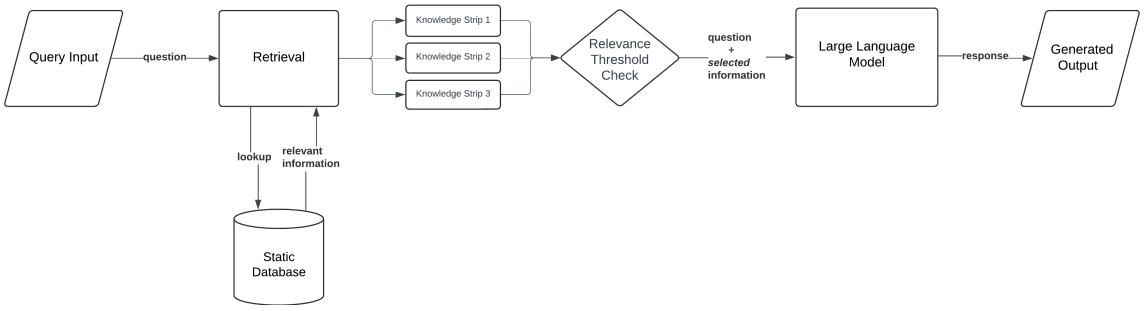


Figure 2.6: High-level architecture of the RAG system.[61]

The framework initially receives the user's natural language query and subsequently conducts semantic retrieval to extract pertinent text fragments from a vector database, which serve as contextual references. These fragments, along with the original query, are fed into an LLM to generate a response. The generated output is then processed further and returned to the user as feedback.

The retriever is responsible for transforming the input query into a vector representation and identifying the K candidate texts with the highest semantic similarity in the index. This index is typically hosted by a pre-built vector database, which stores vector embeddings of all document fragments in the knowledge base. These vectors are usually encoded by a pre-trained embedding model (Embedder, such as BERT or Sentence Transformer), ensuring that semantically related texts are positioned closely in the vector space. The retriever then conducts an efficient nearest-neighbor search using inner product or cosine similarity across the query and document vector spaces to locate the most relevant text passages. Next, the generator takes the retrieved K texts, which are usually concatenated with the original question with additional context. Then they are used as input to generate answers that incorporate external knowledge[61].

In contemporary RAG implementations, the generator typically leverages pre-trained sequence-to-sequence Transformer large models (e.g., BART, T5, etc.) proceedings.neurips.cc. During end-to-end training, the retrieval module and the generator module are jointly optimized, enabling the model to learn optimal selection of the most relevant documents and effectively incorporate the retrieved content into the responses. Overall, RAG systems seamlessly integrate information retrieval techniques with generative models: the retrieval module expands the model's "view," while the generation module synthesizes the retrieved knowledge into coherent, natural lan-

guage outputs. In this process, indexing and embedding models ensure efficient access to knowledge, the retriever selects pertinent information, and the generative model provides language understanding and generation capabilities. These modules complement each other to achieve high-quality knowledge-enhanced text generation.

2.3.4 Introduction of Knowledge-Augmented Generation

Knowledge-Augmented Generation is a newly proposed framework developed on the basis of RAG. It is designed to further empower generative models to leverage logical reasoning and structured knowledge. Unlike RAG, which primarily relies on unstructured text fragments, KAG explicitly integrates knowledge graphs (KGs) and rule-based reasoning mechanisms. This advancement stems from the higher requirements for logical consistency and domain-specific expert rigor. RAG retrieval relies on vector similarity, which poses challenges in ensuring the logical coherence of retrieved results. This approach frequently encounters difficulties when addressing knowledge logic challenges, such as numerical computations, temporal inferences, and hierarchical relationships. As an example, in scenarios such as financial statement analysis and legal contract review, answering user queries necessitates models with capabilities for mathematical calculations, relationship matching, and adherence to business rules, to ensure good accuracy. RAG, which relies solely on text retrieval, may fail to ensure compliance with these complex logical constraints. To mitigate this, KAG introduces structured knowledge representation and reasoning components into its framework. KAG represents domain knowledge as a graph structure where nodes represent entities/concepts, and edges represent relationships. The domain knowledge is then combined with domain-specific rules and constraints to provide an enhanced solution that balances semantic relevance and logical accuracy. [12]

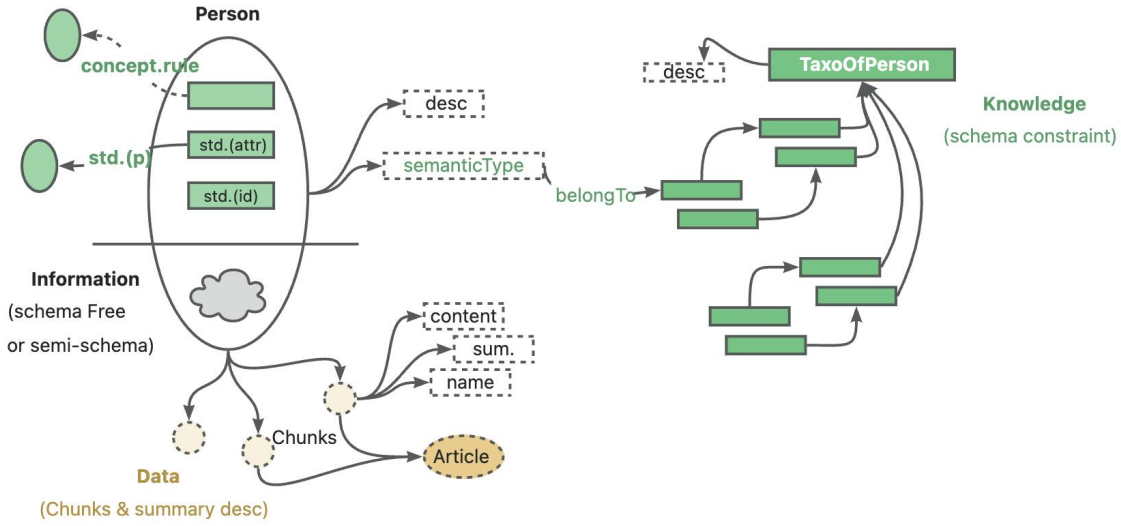


Figure 2.7: KAG framework: Builder, Solver and Model components (adapted from[12]).

The KAG framework typically consists of three core components: (1) Knowledge Construction Module (KAG-Builder), which is responsible for transforming raw domain knowledge into a graph format and establishing a bidirectional indexing system. For example, information extraction from professional documents yields entity-relation triples, which are used to construct a domain knowledge graph while maintaining links between graph nodes and the original text segments. This enables both the retrieval of textual evidence from graph nodes and the localization of knowledge units in the graph from text. (2) Knowledge Solving Module (KAG-Solver): This module handles online reasoning by combining retrieval and symbolic processing. On one hand, it retrieves relevant knowledge from the graph and text indexes, similar to a RAG retriever. On the other hand, it leverages a logical reasoning engine to process the relationship chains and rules in the knowledge graph. Notably, KAG employs a logical-form-guided reasoning scheme, generating symbolic expressions (e.g., SPARQL queries, inference rule sequences) for queries and executing them on the knowledge graph to derive implicit answers. This symbolic reasoning addresses the limitations of vector retrieval, enabling the system to tackle multi-step reasoning-oriented tasks, such as numerical comparisons and temporal inference, with greater accuracy. (3) Knowledge-Augmented Model (KAG-Model), which is the knowledge-integrated generative model component. KAG-Model typically builds upon large pre-trained models and enhances their capacity to utilize knowledge graph information through specialized training strategies. For example, it represents graph information in a format amenable to LLM understanding (e.g., through format conversion or annotated knowledge embeddings) and aligns model outputs with graph reasoning results during training. This ensures that generated responses are grounded in the evidence provided by the knowledge graph.

When generating responses, the KAG-Model synthesizes inputs from the retrieval module, the reasoning engine, and the natural language query to produce answers with both rich content and logical consistency. In summary, the KAG architecture is more sophisticated but rigorous than RAG. KAG unifies unstructured text and structured knowledge, retaining the ability to retrieve rich corpus details. Simultaneously, it introduces formal knowledge constraints through knowledge graphs to ensure that model outputs align with the objective rules. This ensures that model outputs adhere to the objective rules and logical requirements of domain-specific knowledge.

2.4 Introduction of LangChain

To implement RAG and KAG pipelines efficiently, developers should rely on specialized development frameworks. This project employs LangChain, which is a popular platform. LangChain is an open-source toolkit that provides standard components for building applications empowered by LLMs[62]. It was created to simplify the development of complex “LLM chains” including sequences of prompt processing, model calls, and integration with external data. The core idea is that rather than working with a raw LLM API in isolation, LangChain offers abstractions such as:

Prompt templates (parametrized prompts for consistency and reusability), Memory (to remember past interactions in a conversation), Retrievers and Vector Stores (to do the document search part of RAG), Tools/Agents (to allow an LLM to invoke external tools like a calculator or database), and Chains (pipelines that tie these components together in logic flows)[62].

Specifically, LangChain's RAG process first involves loading data such as manuals, logs, and other multi-format documents. Then it splits the loaded documents into chunks, embedding the split data into a vector space using an embedding model, and finally storing it in a vector store for subsequent retrieval. Figure 2.8[63] shows the specific steps of this process:

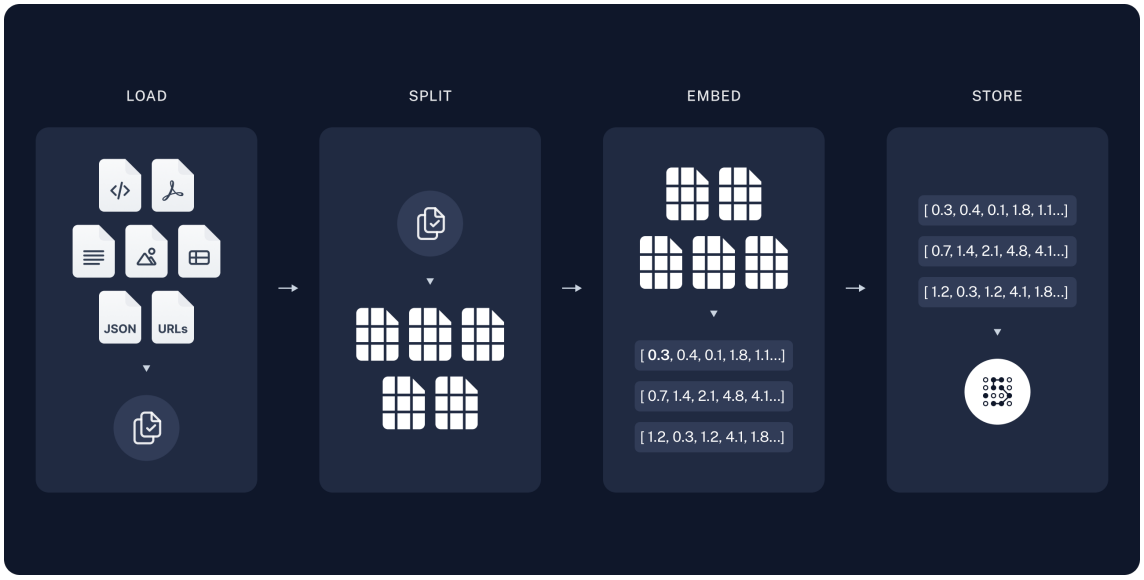


Figure 2.8: LangChain indexing pipeline: loading, splitting, embedding, and storing data in vector stores.

When a user submits a query, LangChain calls the retriever to extract the most relevant document fragments from the vector store. Then it combines this information into a context that is passed to the LLM to generate an accurate response. Figure 2.9[63] describes this typical query and response process:

For example, using LangChain, developers can rapidly construct a "retrieval-augmented QA chain" that handles the user query. It employs an embedding model to retrieve the most relevant documents, and feeds both the query and the retrieved documents into an LLM prompt to generate a grounded, context-aware answer. LangChain abstracts away the boilerplate of these steps, allowing developers to focus on higher-level design. This framework has gained prominence in the LLM ecosystem because it streamlines the integration of LLMs with organizational data[62]. Instead of fine-tuning an LLM on a proprietary dataset, which is infeasible if the model is closed or the data is frequently updated, LangChain enables runtime data injection via retrieval. This approach is widely adopted in "LLM-powered data assistant applications" applications, offering a practical solution for leveraging domain-specific

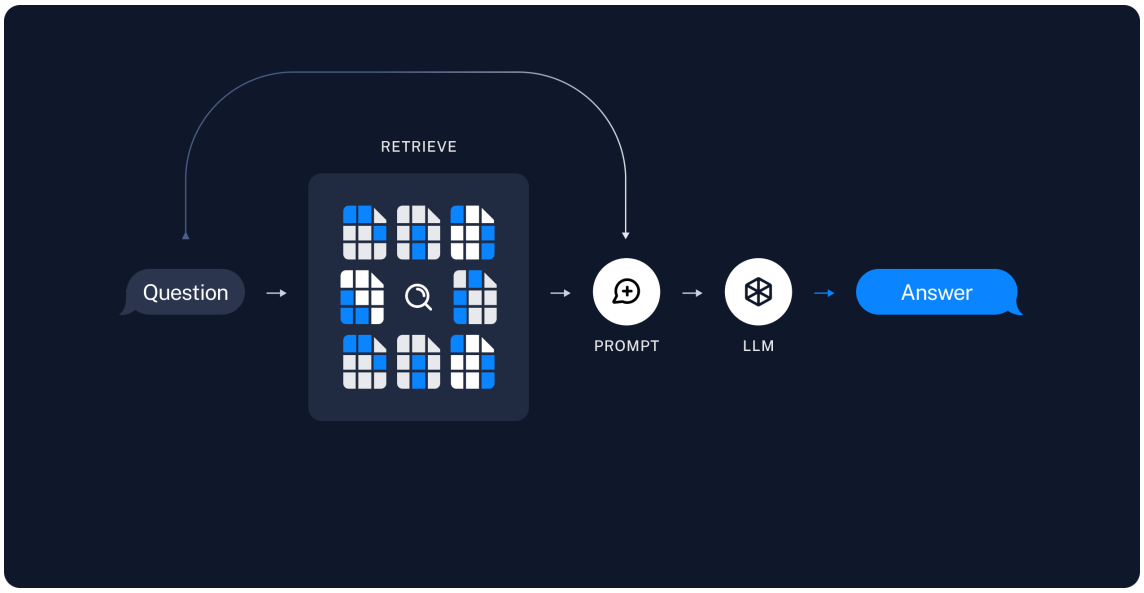


Figure 2.9: LangChain retrieval-augmented generation workflow: retrieving relevant document chunks based on the query and producing answers with an LLM.

knowledge. It effectively implements RAG patterns, as AWS highlights, LangChain "allows LLMs to access new datasets without retraining" and to build context-aware workflows like RAG that reduce hallucination and improve accuracy[62].[64]
The LangChain architecture is inherently modular with its core primary components comprising: (1) Models (LLMs or chat models, which LangChain can interface with uniformly, be it OpenAI's API or an open local model), (2) Prompts (including prompt templates and output parsers to structure model I/O), (3) Indexes or vector stores (for efficient similarity search on document embeddings), (4) Retrievers (the logic to query those indexes and fetch documents), (5) Memory (short-term memory of interactions, useful in dialogue), and (6) Agents (which involve the LLM making decisions to use tools in an iterative loop). These components can be assembled into Chains, representing a full application pipeline from input to output[62]. For instance, a maintenance chatbot chain might accept a user's fault description. Then it retrieves relevant manual sections via a vector store, and feeds this context to an LLM to generate a step-by-step troubleshooting guide. LangChain was selected as the developing platform for this project since it drastically reduces the development effort required to build such pipelines and is well-supported by an active community.

By adopting LangChain, we ensure that our implementations of RAG/KAG for predictive maintenance are built on standardized frameworks rather than ad-hoc solutions. Furthermore, LangChain's extensibility allows us to integrate custom components if needed (for example, a custom knowledge-graph reasoning tool for KAG). In summary, LangChain serves as the glue between LLMs and maintenance data sources (equipment logs, knowledge bases). It enables the creation of an intelligent assistant capable of answering technical queries reliably by combining trained knowledge with retrieved factual informati[65].

2.5 State of the Art in Large Language Models for Predictive Maintenance

Research on leveraging LLMs for industrial predictive maintenance and fault diagnosis is gaining traction, despite it remains in a primary stage. Traditional predictive maintenance systems have predominantly relied on classical ML/DL models (e.g., random forests, SVMs, CNNs, LSTMs) for anomaly detection and prognostics tasks[25]. The recent advent of LLMs offers new opportunities in this field. Particularly, LLMs hold great potential in enhancing the intelligence of maintenance systems by enabling them to interpret unstructured text, integrate multimodal data, and generate explainable responses. Below, we survey several emerging applications of LLMs in this domain.

2.5.1 Anomaly Detection and Fault Classification

Researchers have investigated the use of LLMs to analyze raw sensor signals for analyzing raw industrial sensor signals by encoding them as token sequences compatible with LLM processing. For example, Qaid et al. (2024) introduced FD-LLM, a framework that adapts pre-trained LLMs to ingest time-series vibration data for machine fault diagnosis[25]. In this approach, vibration signals are encoded either as strings or as statistical feature tokens, enabling fine-tuned LLMs to classify fault types. Notably, their results showed that LLMs such as Llama 3 (and an instruction-tuned variant) achieved strong fault detection performance, surpassing state-of-the-art deep learning models in diverse scenarios. Moreover, there is a remarkable finding: it suggests that the rich representations learned by LLMs can transfer to industrial sensor data analysis when properly encoded. In another study, Hang et al. (2023) combined an open Chinese LLM (ChatGLM2-6B) with fuzzy logic for electrical equipment fault diagnosis. This integration aimed to leverage the LLM's contextual understanding alongside fuzzy reasoning for uncertainty handling[66]. Early reports indicate that integrating an LLM improves the adaptability and accuracy of fault diagnosis, as the LM can better interpret nuanced patterns and linguistic descriptions of faults than traditional models. Similarly, in the spacecraft domain, researchers used fine-tuned LLMs to diagnose satellite faults, treating telemetry streams as a form of language. This approach allowed the model to consider historical fault descriptions (as text) alongside live telemetry, mimicking an expert's ability to recall similar past incidents[20]. These early works enhance the potential of LLMs in anomaly detection: beyond pure classification accuracy, LLMs can provide rationales for detected faults (e.g., pointing to which sensor reading behavior influenced the decision). This transparency addresses a key limitation of deep black-box models, offering insights that are often critical for industrial decision-making[67].

2.5.2 Maintenance Report Generation

Another promising application avenue of LLMs lies in their ability to automate the generation of machine condition reports[68][69]. In industrial settings, technicians

manually draft narrative reports from inspections or parse through alarm logs to synthesize incidents. LLMs, with their strength in language generation capabilities, are well-suited to streamline the process here. For instance, an LLM can be prompted to produce a summary of a week's worth of sensor alarms, highlighting critical anomalies and recommended actions. Proof-of-concept studies have demonstrated that LLMs like GPT-4 can be fed with a chronological list of events or sensor readings. Then the LLMs can be asked to produce a concise incident report, coherent incident reports akin to human-written maintenance logs, thanks to their training on vast textual data. A notable example is the use of multimodal LLMs for maintenance logging. Alsaif et al. (2023) proposed a framework where GPT-4 (multimodal preview) integrates numeric time-series data and contextual text (like operator notes) to generate a fault explanation and maintenance suggestions[20]. The model can contextualize faults (e.g., "Pump 2 shows an abnormal vibration spike around 14:00, likely indicating a bearing wear issue") and schedule next steps. It demonstrates how LLMs might auto-generate draft maintenance reports for engineers to refine. While such generative usage holds promise, it also requires caution. The LLM might hallucinate plausible-sounding causes or recommendations unsupported by data. To mitigate this, researchers advocate grounding LLM outputs. For instance, coupling generation with RAG ensures claims are backed by retrieved sensor data or known failure modes. Nonetheless, LLMs' ability to transform raw data into structured narratives can significantly reduce documentation overhead. This also ensures that insights from predictive maintenance systems are effectively communicated to human decision-makers.

2.5.3 Interactive Maintenance Assistants

Perhaps the most exciting application of LLMs lies in developing conversational assistants for maintenance personnel[70]. Such assistants could answer questions like "What does error code E17 mean on Machine X?" or "How do I troubleshoot a temperature spike in the hydraulic system?" by synthesizing information from equipment manuals, historical maintenance records, and real-time data. This approach represents a specialized case of retrieval-augmented QA, where the knowledge base comprises technical documentation and the LLM generates natural-language responses. Research in this area often employs RAG pipelines with LLMs (as discussed in section 2.3). For example, Microsoft researchers have explored using ChatGPT to assist factory technicians by augmenting it with a library of standard operating procedures. This enables step-by-step guidance through repairs with references to official guidelines. In the aviation industry, a "virtual co-pilot" was developed using an LLM to help pilots or maintenance crews rapidly access procedures. Given a query about a warning indicator, the system retrieves the relevant checklist from the aircraft manual, and the LLM presents it in an easy-to-follow dialog format[20][71]. In another case, an LLM-based chatbot for an automotive assembly line was designed to answer engineers' questions about machine malfunctions by correlating error logs (parsed by the LLM) with known issues in a troubleshooting database[72].

These systems demonstrate that LLMs can serve as an intuitive interface between

humans and complex technical knowledge. Instead of searching through massive PDF manuals, a technician can simply query the AI assistant. Early user studies suggested this can significantly speed up problem resolution and also help less-experienced staff in adhering to best practices (the assistant can ensure no diagnostic step is missed, for instance). However, a critical challenge is ensuring the assistant's responses are trustworthy—hallucinated instructions in safety-critical procedures could pose significant risks. To address this, state-of-the-art implementations maintain "human-in-the-loop" oversight, using the LLM to draft responses that are then verified against source documents (often with sources displayed to the user to foster transparency and trust).

3

Methodology

3.1 System Overview and Scenario Design

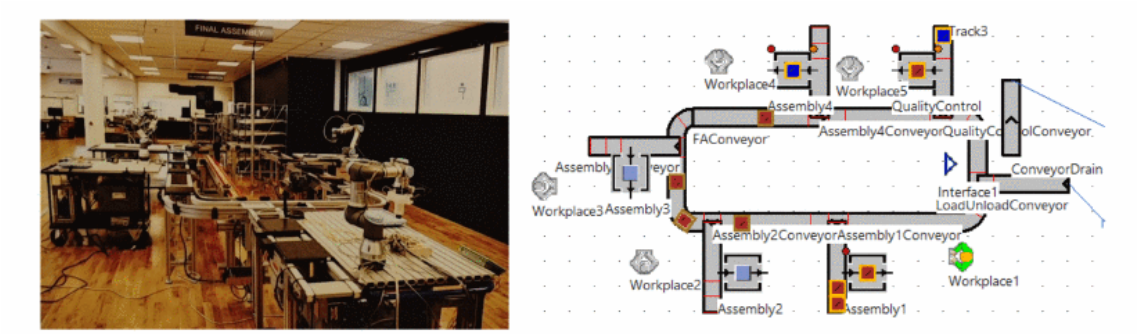
This study is conducted using the lab-scale drone assembly line at the Stena Industry Innovation Lab (SII-Lab), a national testbed for industrial digitization located at Chalmers University of Technology [73]. Instead of a real production line, we employ a virtual experimental setup: a digital twin of the drone assembly process is built in Siemens Tecnomatix Plant Simulation to mirror the physical SII-Lab system[74]. Figure 3.1 illustrates this hybrid setup, where the left side shows an actual SII-Lab workstation and the right side displays its Tecnomatix simulation model.The assembly line model includes four automated stations (three assembly machines and one quality-inspection station) linked by a main conveyor.

To create a realistic maintenance scenario, failures are deliberately introduced in three assembly machines during the simulation. It often suffer from slow fault diagnosis and repair, especially when newly hired technicians lack sufficient experience. Maintenance knowledge is typically scattered across different sources, including historical event logs, sensor data, and manuals to making it difficult for less experienced staff to act promptly.

Our method integrates a LLM with a RAG approach to analyze and respond to these issues. In this LLM and RAG pipeline, the LLM continuously ingests event log data from the digital twin and retrieves relevant information from equipment manuals as needed.

To address the lack of annotated maintenance data, we constructed a hypothetical smart maintenance scenario. This scenario is also used to test the capabilities of the LLM+RAG hybrid model in a controlled environment. The major objective is to investigate how a generative model could assist new technicians in the aspect of integrating fragmented information sources, including manuals, event logs, and sensor data. The test is also conducted even when trained on incomplete or synthetic datasets.

Instead of trying to optimize production KPIs, the demonstrator produced by the model focuses on providing a transferable methodology. First, it shows the method to assemble a maintenance knowledge base that unifies multiple data sources. Second, it illustrates emulation of the complexity of real-world troubleshooting through



(a) Drone assembly stations.

(b) PlantSim digital twin model.

Figure 3.1: (a) SII Lab Production Line (b) Tecnomatix PlantSimulation Virtual Model [74][75].

simulation. Finally, it embeds the gathered knowledge into an LLM powered question answering assistant. This simulation-driven approach offers a pragmatic compromise, that is, it circumvents the scarcity of comprehensive industrial data while still yielding an explainable and testable reasoning pipeline.

Moreover, the simulation-driven strategy also provides a practical compromise between real-world constraints. This issue originates from the lack of access to full industrial data and the need for explainable and testable maintenance reasoning pipelines. In essence, we aim to answer whether a technician could ask an AI assistant questions about a malfunctioning system. Subsequently, we investigate how the AI should reason, retrieve, and respond using a combination of expert manuals, event logs, and synthetic failure cases.

3.2 Overall Framework

To contextualise the proposed methodology, we first sketch the end-to-end information flow of the predictive-maintenance assistant. Figure 3.2 visualises this pipeline and clarifies how raw shop-floor data are progressively transformed into traceable answers for technicians. The workflow begins with a data-conditioning phase in which sensor streams, event logs and PDF manuals are normalised into timestamped tables or cleaned text snippets, enabling uniform downstream handling. Once curated, these artefacts are channelled into a dual-store knowledge layer: dense embeddings populate a vector database that excels at semantic similarity search, while key entities and relations are lifted into a lightweight domain graph that captures equipment hierarchy and fault causality. On top of this foundation sits an edge-deployable generative model (a quantised blend of DeepSeek R1 and Llama3) whose retrieval module can query either store depending on the nature of a user prompt. When a technician issues a question, the utterance is embedded, matched against the vector store, optionally expanded via the graph, and the retrieved evidence is injected into the model's context window; the model then produces an answer explicitly anchored to that evidence, together with source citations.

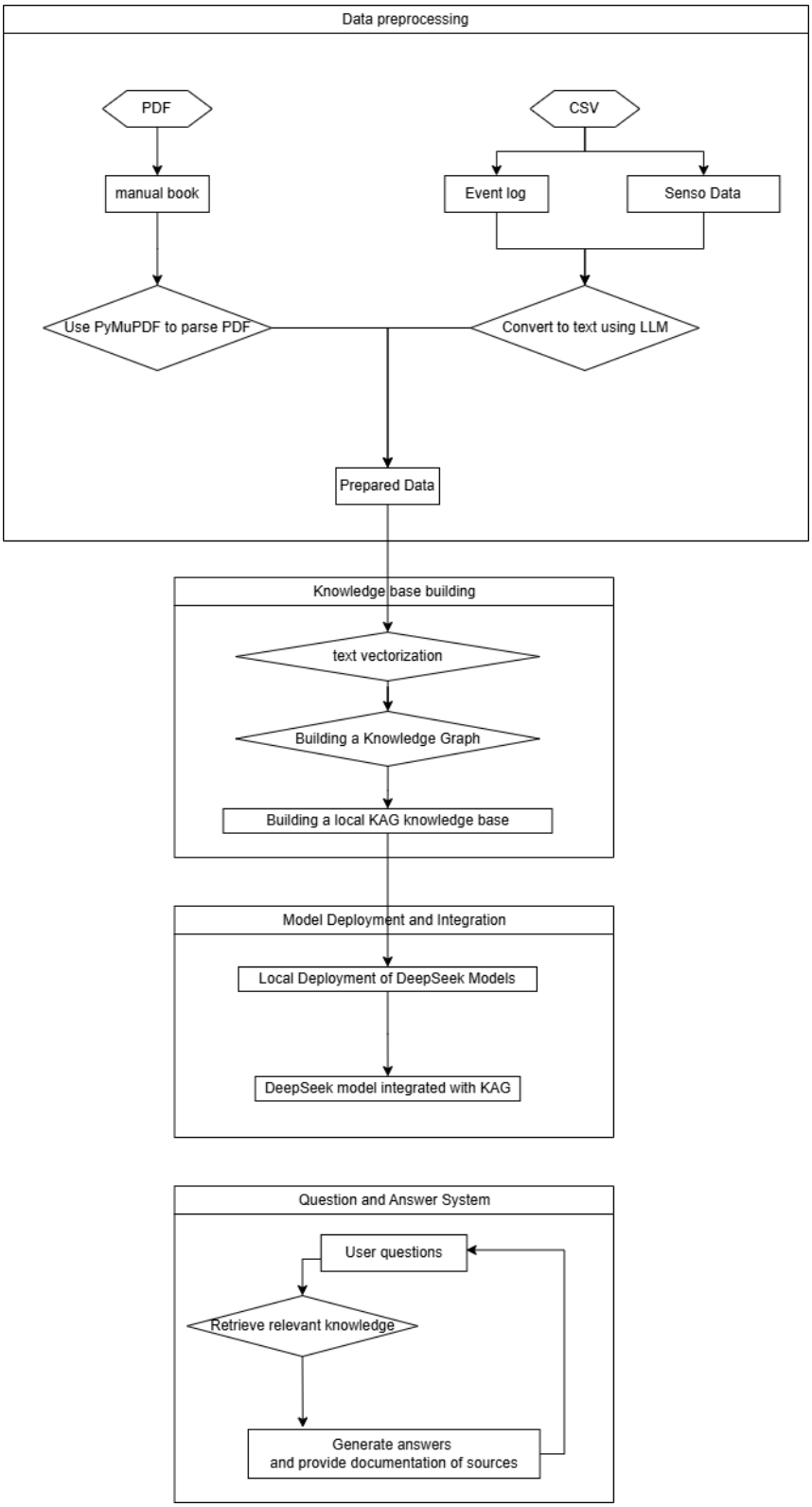


Figure 3.2: Workflow diagram of the proposed predictive-maintenance methodology.

A comparable retrieval-augmented pattern has recently proved effective in maintenance advice systems—Han et al. (2024), for instance, convert turbine manuals into structured chunks before retrieval to ground their LLM outputs [60]. Building on that precedent, the sections that follow unpack each stage of Figure 3.2 in turn, from preprocessing choices to on-maual inference constraints, and discuss how the design supports explainability and low-latency deployment in industrial settings.

3.3 Data Collection and Data Preprocessing

3.3.1 Equipment Manual Book

This study utilizes the manual equipment data that originates from official maintenance documents that can be found in the Haas CNC machine manufacturer[76]. These documents, primarily in the PDF format, include installation guides, troubleshooting instructions, maintenance checklists, and component replacement procedures. Each document covers a specific system module of the industrial equipment and contains both textual descriptions and schematic references.

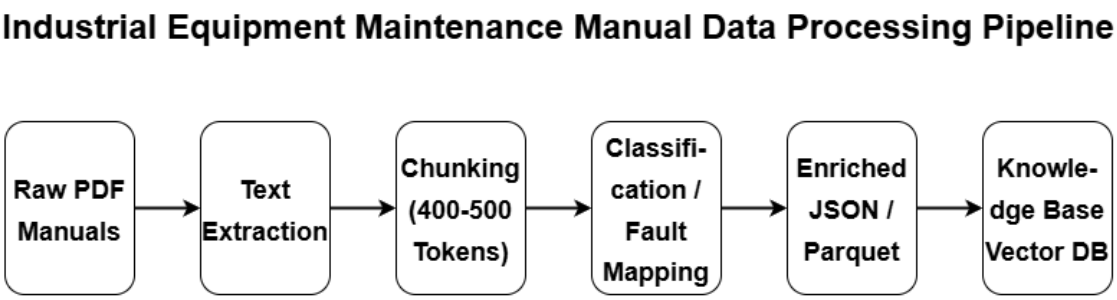


Figure 3.3: Manual book preprocessing

To prepare this data for downstream processing, all manual pages were first converted into plain text while preserving their original layout, section headers, and page numbers. Non-textual elements, such as images, were extracted and stored separately to maintain their association with the corresponding text segments. The extracted text was then split into coherent chunks of approximately 400–500 tokens, ensuring that logical paragraphs were not broken across boundaries. Each chunk was annotated with metadata including the original file name, page index, and identified component type.

To enhance structure and usability, the content was automatically classified into broader system categories (e.g., mechanical drive, control electronics, and lubrication) and associated with failure-related entities such as components and fault types. Representative maintenance suggestions and fault-handling procedures were extracted from the descriptive sections and normalized into brief solution statements. These were further grouped by subsystem, forming a semantically tagged corpus suitable for retrieval or downstream question–answer generation.

Table 3.1: DataFrame Row

Column	Value
file_name	`BMT65_75 - Turret Indexer Assembly - Troubleshooting Guide.txt`
title	BMT65_75 – Turret Indexer Assembly – Troubleshooting Guide
part_name	Turret Indexer Assembly
category	Spindle & Drive
failure_type	1
solutions	`[array of solution steps]`

Table 3.2: Brief Solution Statements

Part Name	Turret Indexer Assembly
Solutions	1. Check the incoming air supply. 2. Check the voltage to the turret clamp or unclamp solenoid. 3. ...

The result of this preprocessing phase was a structured repository of annotated, chunk-level manual data, indexed by component, failure theme, and document context. This corpus was used to support both knowledge base construction and model training throughout the rest of the project.Detailed results can be found in the Results section.

3.3.2 Event logs

The event logs used in this project are synthetically generated through a simulation environment, as real-world industrial logs are inaccessible due to confidentiality restrictions. The generated simulated logs contain sequences of machine-level fault events over a virtual production timeline. Each entry includes timestamps, component identifiers, and failure types. However, since the simulated log is independent of the equipment manuals, the log entries are not initially aligned with any known troubleshooting procedures or solution patterns.

To bridge the gap between simulation and practical scenarios, each fault event is mapped to a relevant solution category derived from the processed equipment manuals, to enrich the data. The classification is decided through shared labels such as failure mode and affected subsystem. This enables each log entry to be associated with a general maintenance recommendation. Detailed results can be found in the Results section.

The implementation details of the data preprocessing pipeline are documented in Appendix A.0.0.0.4.

Table 3.3: Failure Event Record

Column	Value
model_name	`*.Models.Model.Assembly3`
failure_profile	FailureA
start_time	8:56:00.3331
end_time	9:12:02.0050
failure_interval	1:04.4022
failure_type	1
solution_method	5
is_fixable	0

3.3.3 Operator knowledges

Due to the absence of practical operator feedback, synthetic operator knowledge was generated to reflect responses from technicians in the real world. For each enriched log entry, we automatically generated a natural language description. This was done by reformulating two key elements, including the failure condition and suggested action. The output was structured as a narrative, mimicking an operator report. These descriptions included important contextual details, such as symptom duration, predicted recurrence, and required interventions.

See Appendix A.0.0.0.4 for the implementation of operator-style feedback generation using large language models.

3.3.4 Sensor data

Data collection and preprocessing

The dataset includes sensor measurements and faults of the virtual production line based on the SII Lab real-world setup and its corresponding digital twin built with Siemens Tecnomatix Plant Simulation. It contains continuous variables such as the temperature and pressure of different conponents, discrete variables such as the open/closed sockets and the fault texts. The data covers a continuous 20-day period of three machines, with each record collected once a minute.

The texts of the raw data mixed English and Swedish, and have been mapped accordingly. If the data in the text columns is empty or marked as NaN, it is considered that the machine is operating normally.

Continuous variables of sensors measurements are selected to reduce computational complexity.

Table 3.4: Sensor Data Overview

Attribute	Value	Description
Total records	86,400	3×20 days$\times$ $1,440$ minutes/day
Features (columns)	71	Sensor measurements (21), System status flags (6), Fault texts (5), Machine alarm flags(5)
Unique fault types	41	Categorized from `fault_text_2` to `fault_text_5`
Time span	20 days	Continuous monitoring period
Sampling frequency	1/min	Per-minute recordings
Missing values	0.7%	Forward-filled from adjacent records

Key sensors include: Hydraulic pressure (`pn7071`), spindle temperature (`tp3237`), vibration intensity (`sp1 veff_10hz_5khz`)
Fault examples: Trajectory errors (65%), communication failures (22%), safety circuit faults (13%)

Dynamic threshold

Machine Learning is a good method to detect and classify faults. However, normal running data of the machines are much more than failure. It is imbalance between the amounts of data. Thus conventional statistical methods is selected to calculate the sensor thresholds. To simulate the real world complex industrial scenarios, this method creates adaptive thresholds for each sensor measurements. It combines Statistical Process Control (SPC) and actual factory needs, as demonstrated in RAAD-LLM [9].
For each machine $s \in \{1, 2, 3\}$ and sensor i, the dynamic thresholds are derived as follows:
Mean calculation:

$$\mu_{s,i} = \frac{1}{N_s} \sum_{t=1}^{N_s} x_{s,i}(t) \tag{3.1}$$

where N_s is the number of data points in machine s, and $x_{s,i}(t)$ denotes the sensor i's value at time t.
Standard deviation:

$$\sigma_{s,i} = \sqrt{\frac{1}{N_s} \sum_{t=1}^{N_s} \left(x_{s,i}(t) - \mu_{s,i}\right)^2} \tag{3.2}$$

Threshold definition:

$$\text{Upper Threshold } (\text{UT}_{s,i}) = \mu_{s,i} + 1.5\sigma_{s,i} \tag{3.3}$$

$$\text{Lower Threshold } (\text{LT}_{s,i}) = \mu_{s,i} - 1.5\sigma_{s,i} \tag{3.4}$$

1.5σ multiplier is a tighter bound than traditional 3σ control limits, enhancing sensitivity to anomalies. Compared with static thresholds, dynamical thresholds can

adjust the operating baseline of each machine.

Table 3.5: Corpus statistics after preprocessing

Corpus	Entries	Chunks
Equipment manuals	55	1 361
Event logs	2 316	2 316
Sensor Data	135	135

3.4 Retrieval Augmented Generation

As shown in Figure 3.4, this system follows a Retrieval-Augmented Generation workflow that turns each user query into an answer backed by explicit knowledge.

First, a knowledge base is pre-built from maintenance-event logs and technical manuals. All documents are vectorised and indexed so they can be efficiently retrieved. When a query arrives, the retrieval module selects the most relevant document snippets and supplies them as context, together with the original query, to a large-language model.

By injecting verified source material at generation time, the RAG pipeline keeps the LLM's output consistent with domain knowledge, lowers the risk of hallucination, and improves both the accuracy and the timeliness of the answers.

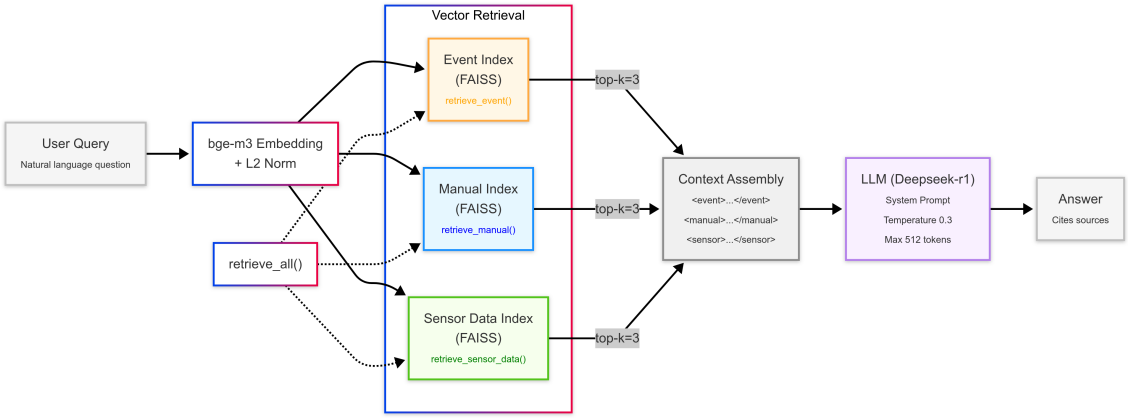


Figure 3.4: Overall Retrieval-Augmented Generation (RAG) workflow adopted in this study.

3.4.1 Knowledge Base Construction

3.4.2 Embedded Models and Vector Databases

During the knowledge base construction phase, the system utilised high-performance embedded models BAAI/bge-m3 to convert document semantics into vector representations. Specifically, each document fragment is encoded as a 1024-dimensional vector. To improve batch processing efficiency, embedding calls are made in batches of 5 fragments, thereby efficiently utilising the model API. A total of 3,111 vector embeddings were generated during the maintenance of the knowledge base, including 123 sensor data fragments, 2,317 event log fragments and 794 manual fragments.

These vectors and their metadata are stored in a FAISS-based vector database[77]. A Flat IP index is used, which keeps all vectors in memory to guarantee full recall. All vectors are first L2-normalised to unit length; similarity is then computed with inner product, which is mathematically equivalent to cosine similarity for unit vectors[78].

First, both the document vector $\mathbf{v}$ and the query vector $\mathbf{q}$ are subjected to L2 normalization:

$$\widehat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2}, \quad \widehat{\mathbf{q}} = \frac{\mathbf{q}}{\|\mathbf{q}\|_2}, \quad \text{where } \|\mathbf{v}\|_2 = \sqrt{\sum_i v_i^2}$$

This operation projects all vectors onto the unit hypersphere, retaining only their directional information and eliminating magnitude differences. The similarity score is then computed as the dot product of the two unit vectors:

$$s = \widehat{\mathbf{q}} \cdot \widehat{\mathbf{v}} = \cos\theta$$

The resulting score $s \in [-1, 1]$ equals the cosine of the angle θ between the two vectors, with larger values indicating higher semantic similarity.In practice,scores in the range $[0.85, 1.00]$ typically indicate near-synonymy or minimal syntactic variation. Scores in $[0.60, 0.85]$ suggest the same topic expressed differently and may be retained depending on downstream tasks. Scores in $[0.30, 0.60]$ imply only weak relevance and are often filtered in large-scale retrieval to reduce noise. Scores in $[0, 0.30]$ reflect irrelevance and are often used as negative samples. Scores below zero indicate opposite directions, often due to poor or corrupted embeddings.

Note that thresholds are not universal: general-purpose embedding models (e.g., OpenAI's 'text-embedding-3' series) often yield more compact vector distributions and may require slightly lower thresholds. In domain-specific retrieval tasks such as document or code search, a cutoff of 0.6 is often considered a strong match. If higher recall is desired, a relaxed threshold like 0.4–0.5 may be used; if precision is prioritized, values above 0.75 are preferable.

Table 3.6: Interpretation of cosine similarity score s

Range s	Semantic Meaning	Typical Action
0.85–1.00	Nearly identical meaning or minor syntactic variation	Considered a strong match; often top-ranked in retrieval
0.60–0.85	Same topic with different wording	Retain based on task needs; often included in RAG context if $s \geq 0.7$
0.30–0.60	Weak relevance or partial context overlap	Usually filtered to reduce noise in large corpora
0–0.30	Irrelevant	Treated as negative samples, useful for contrastive learning
< 0	Opposite directions (rare)	Indicates complete mismatch or embedding error

3.4.3 Model Deployment and Integration

3.4.4 Retrieval Module and Document Filtering

The system provides three dedicated retrieval functions for selecting relevant document fragments from the vector database, as summarized in Table 3.7. The `retrieve_event()` function restricts the search to maintenance event log entries, supporting queries that require historical failure context. The `retrieve_manual()` function targets only technical manuals, useful for retrieving procedural guidance from official documentation. The `retrieve_all()` function applies no type filter or invokes both sources jointly to provide a comprehensive reference set.

These functions distinguish between document categories through metadata fields, particularly the `kind` attribute embedded in the vector index. This allows selective filtering for either `"event"` or `"manual"` types, ensuring that returned content matches the intended knowledge source.

This modular design allows the system to adaptively retrieve content based on the query's nature: event logs are prioritized for questions about fault history, while manuals are emphasized for inquiries involving standard operating procedures. When needed, multi-source retrieval can be used to enrich the response with both contextual background and authoritative instructions.

Retrieved document fragments are then formatted into structured context blocks. Event log entries provide insights into related maintenance history, whereas manual snippets contribute normative steps and technical references. These blocks are organized by source—e.g., historical failure logs followed by manual summaries—to ensure clarity and logical coherence in the final assembled context.

Table 3.7: Retrieval utility functions and their roles

Function	Filter Condition	Primary Purpose
`retrieve_event()`	`kind == "event"`	Query historical failure logs
`retrieve_manual()`	`kind == "manual"`	Obtain official maintenance-manual steps
`retrieve_all()`	– (no type filter)	Combined retrieval to assemble final context

3.4.5 Large Language Model Integration and Answer Generation

Upon receiving the retrieved context, the system utilizes a customized large language model to generate the final response. A carefully crafted system prompt guides the model to follow a logical reasoning path: it first analyzes the historical background and prior maintenance actions using event log snippets, then refers to manual excerpts to extract authoritative repair procedures.
The model selectively cites sources when incorporating content from manuals, appending references when providing actionable guidance to enhance answer credibility and traceability. To meet the demands of industrial engineering tasks, the generation process is configured with a low temperature setting (`temperature = 0.3`) to reduce output variability and ensure consistency. Moreover, the output is limited to 512 tokens to keep answers concise and focused.
Powered by semantically relevant, retrieved context, the LLM synthesizes comprehensive and coherent responses. These responses combine historical diagnostics with procedural solutions grounded in official documentation, showcasing the strengths of Retrieval-Augmented Generation in delivering interpretable and high-accuracy answers in industrial fault diagnosis scenarios.

3.4.6 LangChain Framework Integration

To implement the RAG pipeline described above, this project leverages the LangChain framework, which provides modular components for constructing data processing and question-answering chains. LangChain's document loader modules are used to ingest heterogeneous data sources—including PDF-format equipment manuals and structured maintenance logs—and convert them into standardized text document objects for downstream processing[62].
Long documents are then segmented using LangChain's semantic-aware text splitters, which divide the content along natural boundaries such as paragraphs or sections. Each segment is typically kept within a few hundred words to remain within the LLM's context window.
The segmented text fragments are embedded using a pretrained model, `BAAI/bge-m3`, producing 1024-dimensional vector representations. These vectors, along with metadata such as `kind` and source file name, are stored using LangChain's FAISS vector store interface to build a searchable vector database.
In the query phase, LangChain's retriever module encodes the user query using the same embedding model and performs similarity search over the vector database. The

custom functions `retrieve_event()` and `retrieve_manual()` interoperate with the retriever to apply document-type filters—for example, retrieving only documents labeled `"event"` or `"manual"`—or query separate indices accordingly for finer-grained control.

Retrieved fragments are assembled into a structured prompt using LangChain's chaining mechanism. A predefined prompt template is populated with the user query and the context from event logs and manuals. With LangChain's high-level abstraction classes such as `LLMChain` and `RetrievalQA`, the entire RAG pipeline—from retrieval to generation—can be encapsulated in a single callable chain.

Figure 3.5 illustrates the working logic of this pipeline. Upon receiving a query, the system first retrieves the most relevant documents using the vector-based retriever, then passes these along with the original query to the `Llama3` model, which generates a final answer. Source attributions are included where applicable.

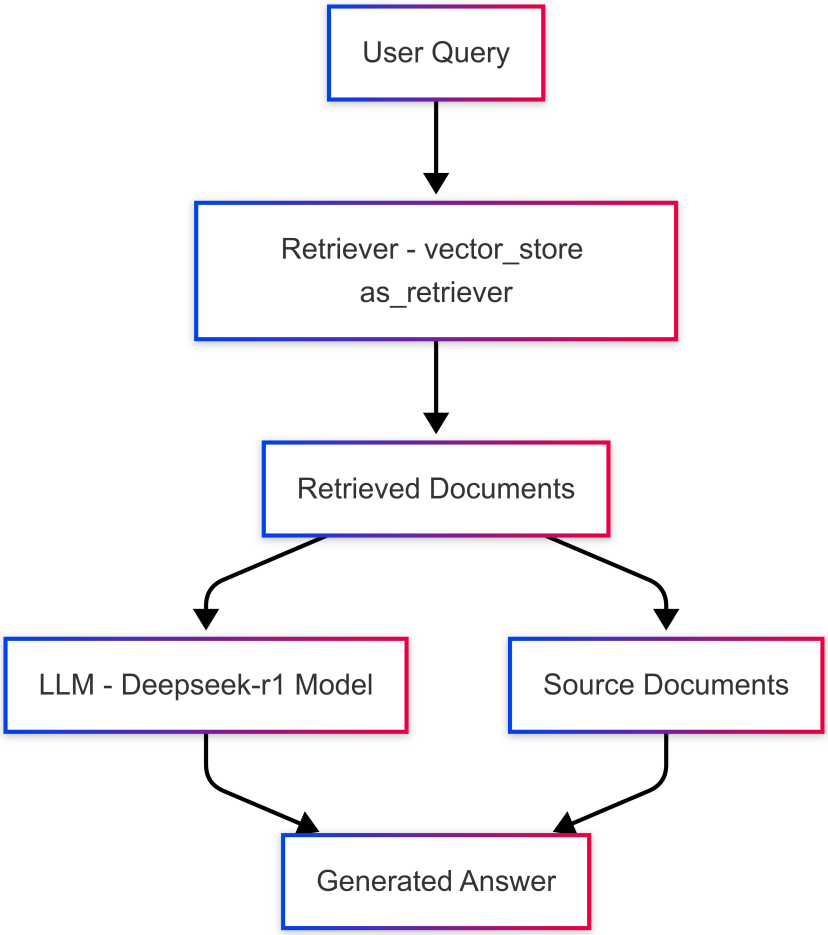


Figure 3.5: LangChain-based QA chain: user query is processed by a retriever, and the retrieved context is passed to an LLM for final answer generation.

In summary, the integration of LangChain significantly simplifies the development of the RAG workflow. By using standardized interfaces for document loading, embedding, storage, retrieval, and generation, this project assembles a end-to-end QA pipeline with minimal engineering effort. The modularity and readability of

LangChain components make the system highly maintainable and adaptable for industrial fault diagnosis tasks.

3.5 Knowledge Augmented Generation

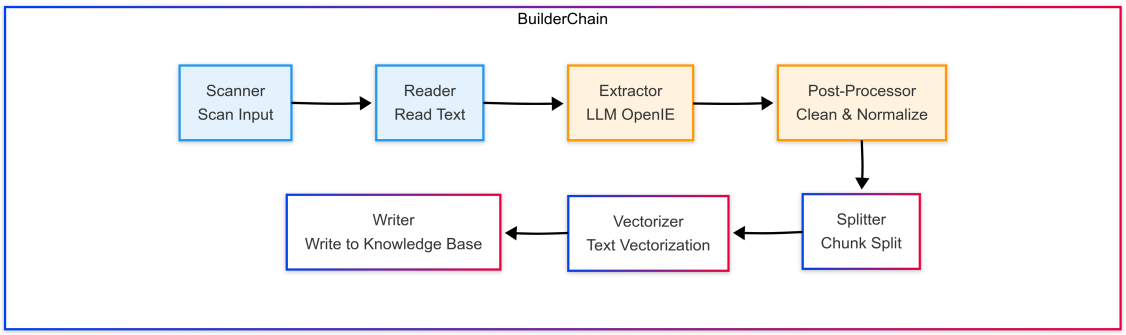


Figure 3.6: KAG Builder Chain

3.5.1 Knowledge Base Construction

Pipeline from documents to structured knowledge

The BuilderChain module in KAG [79] accomplishes the construction of its knowledge base by integrating multiple components, corresponding to the kg-builder section in the configuration. The process begins with the Scanner, which identifies input file sources (e.g., documents in a directory) and enumerates their file paths. Next, the Reader module reads the file content, supporting plain text, Word, PDF, and other formats, and loads them as text strings. Then, the Splitter divides the text into smaller Chunk (text blocks) using contextual logic or length restrictions. By default, a length-based splitting strategy is used (e.g., every 300 characters, without overlap) to ensure compatibility with the context windows of LLMs while preserving semantic coherence. These text blocks serve as the foundational units for subsequent information extraction tasks.

Information Extraction (OpenIE) and Knowledge Organization

The segmented text chunks are input into the Extractor component, which calls a large language model (Llama3) to perform open information extraction (OpenIE). The Extractor uses a schema-constrained extractor (SchemaConstraintExtractor) combined with predefined prompt templates to gradually extract entities, events, concepts, and the relationships between them. Specifically, the Extractor typically first identifies entity lists within paragraphs, then extracts events involving these entities, and iteratively extracts relationships between entities. To reduce noise in open extraction, KAG employs Schema constraints and attribute standardization techniques: performing free extraction when no predefined ontology is available,

while combining domain-specific Schema rules to extract domain knowledge, ensuring the knowledge structure encompasses both open content and domain constraints. For example, during the entity extraction phase, KAG instructs the LLM to generate attributes such as descriptions and category labels for each entity, and annotate their semantic types and concept classes (spgClass), storing this information as node attributes. This process maps the subject/object of each open triple to a normalized knowledge graph node, such as normalizing different expressions with the same meaning into the same entity node. The extracted triples and attributes are cleaned and normalized by the Post-Processor to form a preliminary set of knowledge units.

Table 3.8: Configuration of the `Extractor` module (excerpt from `kag_config.yaml`)

Field	Value	Description
`type`	`schema_constraint_extractor`	Specifies the use of the `SchemaConstraintExtractor`
`llm`	`*openie_llm`	Sets the LLM used for extraction (uses a predefined reference)
`ner_prompt`	`spg_entity`	Template prompt used for Named Entity Recognition
`event_prompt`	`spg_event`	Template prompt used for event extraction
⋮	⋮	Other fields omitted for brevity (`...` in actual config)

Mapping OpenIE to the OpenSPG graph

KAG integrates the extracted open knowledge into a structured knowledge graph (OpenSPG) based on SPG. OpenSPG is a semantically enhanced property graph model whose basic units are S-P-O triples (Subject, Predicate, Object). KAG proposes the LLMFriSPG representation on this basis to make the knowledge graph more LLM-friendly. During the mapping process: entities and events become nodes in the graph, accompanied by types (e.g., entity type, event type) and LLM-generated descriptive attributes (e.g., description and summary); relationships become directed edges connecting nodes, with edge types corresponding to the extracted relationship predicates. For example, in the sentence "The hydraulic system has a leakage fault," the extractor identifies the entity nodes "hydraulic system" and "leakage fault" and uses "has" as the relationship edge connecting these two nodes to construct the equipment fault knowledge graph. Additionally, KAG performs a semantic alignment step to unify knowledge of different granularities, including concept alignment and merging duplicate or ambiguous entities. Through these steps, loosely structured knowledge extracted from open sources is integrated into a semantically consistent and highly connected domain knowledge graph.

Vector Embedding and Bidirectional Indexing

In addition to constructing structured graphs, KAG also generates vector representations for each knowledge unit or text block for semantic retrieval. Our system uses OpenAI's text-embedding-ada-002 to encode text. Through the Vectorizer component (BatchVectorizer), the system batches calls to the vector model API to map document chunks and key node descriptions to vectors. These vectors are stored in the vector index together with the corresponding knowledge nodes or text fragments, thereby establishing a mutual indexing structure between knowledge graph nodes and original text. Mutual indexing means that each node in the graph is associated with its source text fragment (so that the original text can be directly located for verification when needed), and each text segment is annotated with references to the knowledge nodes it contains. This bidirectional link allows the retrieval process to start from the knowledge graph to find original evidence, or start from the user's question to perform vector semantic retrieval and match entities in the graph.

Table 3.9: Vectorization model configuration (excerpt from `kag_config.yaml`)

Field	Value	Description
`model`	`text-embedding-ada-002`	Name of the embedding model used for vectorization
`type`	`openai`	Specifies the provider or framework of the model (e.g., OpenAI)
`vector_dimensions`	`1536`	Dimensionality of the generated embedding vector
`vectorizer`	`*vectorize_model`	Reference to the defined vectorization model (YAML anchor)

Knowledge storage

After extraction, alignment, and vectorization, the Writer component writes the constructed knowledge to persistent storage. KAG supports storing graph data in a graph database or property graph engine (Neo4j), while storing vectors in a vector database (Milvus). In the default implementation, KAG uses the Graph API and Search API of the OpenSPG engine for storage and retrieval: the graph structure is written to the graph store via the Graph API, while text indexes and vectors are written to the vector store via the Search API. The Writer ensures that each knowledge node carries its text fragment ID, and each embedded record in the vector store is associated with the ID of its corresponding knowledge node, thereby achieving a "graph-to-text" inverted index at the storage level. This storage fusion ensures the effective combination of the graph structure and the original text. After all documents have been processed, a knowledge base that integrates structured graphs and unstructured document fragments is constructed, which can be used for efficient retrieval in the query phase.
Figure 3.7 illustrates the overall architecture of the KAG system, showing how

the BuilderChain and SolverPipeline components interact with the dual-indexed knowledge base to enable symbolic and semantic retrieval.

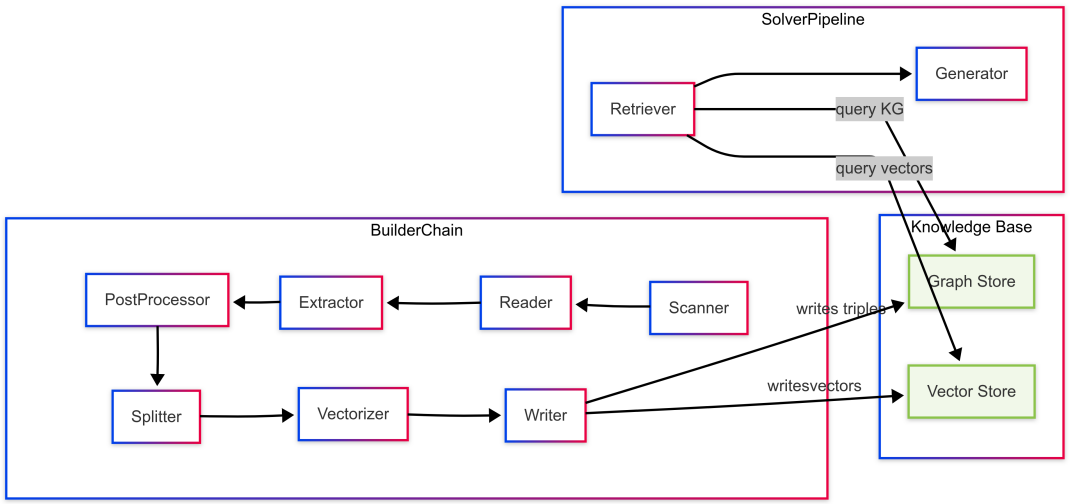


Figure 3.7: KAG system architecture: knowledge construction (BuilderChain) and reasoning workflow (SolverPipeline). Triples and vectors are stored in a dual-indexed Knowledge Base, enabling both symbolic and semantic retrieval.

Parallel and multithreading mechanisms

To improve the efficiency of large-scale knowledge construction, KAG Builder supports multithreaded parallel processing. The num chains and num threads per chain parameters in the configuration control the concurrency: num chains indicates the number of construction chain instances running simultaneously, and num threads per chain specifies the number of threads within each chain. For example, with the default configuration of 4 chains and 2 threads per chain, a total of 8 processing tasks can be executed concurrently. During actual operation, the Scanner distributes the documents to be processed to different chains for parallel processing, with each chain independently executing the read → split → extract → write process. Within a single document, components such as the Extractor or Vectorizer can also utilize the thread pool to concurrently process different chunks, thereby improving the throughput of LLM calls and vector computations. This design significantly enhances the performance of knowledge base construction in multi-core environments, supporting incremental updates and rapid processing of large-scale corpora.

3.5.2 Model Deployment and Integration

LLM model invocation and configuration

KAG encapsulates large language model invocations into a configurable client interface, which specifies the model type, API address, and parameters via kag_config.yaml. The configuration typically defines two main LLMs: OpenIE LLM (for knowledge extraction) and Chat LLM (for question-answering reasoning). For example, the

sample configuration points openie_llm to the Llama3 service of type maas, and chat_llm to the DeepSeek-r1 model of type maas. type: maas indicates that the interface calls the model as a service. Users can modify base_url to point to a locally deployed model service (such as OLlama) and replace api_key with the corresponding key to connect to their own model.

For detailed implementation parameters, please refer to the full KAG configuration in Appendix A.

Table 3.10: LLM settings for extraction and reasoning (excerpt from `kag_config.yaml`)

Key	Field	Value	Description
`openie_llm`	model	Llama3	LLM used for OpenIE-style information extraction (e.g., triple extraction from unstructured text)
	type	maas	Indicates model accessed via Model-as-a-Service (MaaS) platform
`chat_llm`	model	deepseek-reasoner	LLM specialized for complex reasoning tasks including but not limited to chain-of-thought reasoning
	type	maas	Also deployed through MaaS architecture with API access

At the same time, the model namemodel field can specify the specific model used, such as “gpt-4”,“gpt-3.5-turbo”, or the name of the local model. In terms of hyperparameters, KAG allows you to set parameters such as generation temperature and top_p in some configurations or calls. For example, here we set openie_llm and chat_llm’s temperature to 0.3.
For self-built services, you can control this through extended configuration fields or environment variables. When using a local model, if the model supports the OpenAI-compatible API, you can directly configure base_url as the local service address. In summary, KAG’s LLM invocation layer offers excellent flexibility, allowing you to easily switch model sources and adjust key parameters to meet different deployment environments through configuration.

Multi-stage Reasoning and Question-answering Process

KAG’s problem solving is driven by the SolverPipeline module, which uses a logically guided multi-step reasoning mechanism to convert natural language questions into a combination of retrieval and reasoning processes. At the beginning of the process, user queries are first processed by the Reasoner: The Logical Form Planner within the Reasoner calls the LLM (Chat LLM) to analyze the question and generate the logical form of the question or a solution plan. This effectively decomposes complex problems into a series of executable subtasks. Subsequently, the LF Executor

executes the planned steps sequentially: it invokes different retrievers to obtain relevant knowledge and performs necessary symbolic reasoning or calculations. KAG integrates three types of retrievers by default for different types of knowledge sources:

Exact KG Retriever: Used to perform exact matching of entities or relationships in a knowledge graph. It uses Graph API and Search API to query the graph and can directly find the corresponding KG node or triple based on the entity/relationship keywords in the question. For example, for the question "To which system does component X belong?", " the Retriever identifies the entity X and searches the maual knowledge graph for the target node connected by its parent-child relationship.
Fuzzy Knowledge Graph Retrieval (Fuzzy KG Retriever): Used for semantic matching of graph knowledge. When the query does not explicitly mention the explicit node names in the knowledge graph, fuzzy retrieval first uses a vector model to vectorize the query, then performs similarity retrieval in the knowledge base. It may index the node names and description vectors in the knowledge graph to find nodes similar to the query vector, achieving fuzzy matching in knowledge. For example, given a descriptive query, fuzzy retrieval can identify relevant concept nodes or relationship paths as candidate answers.
Chunk Text Retrieval (Chunk Retriever): Used to retrieve answer evidence from raw text fragments. It retrieves several most relevant paragraphs (e.g., top 10) from documents based on vector indexes and can reorder the retrieved results using an LLM to extract the most useful information. Chunk Retriever ensures that even details not covered by the knowledge graph can be found in the original text.

Table 3.11: Comparison of Retriever Types in the KAG Framework

Retriever Type	**Retrieval Method**	**Data Source / Index**	**Typical Use Cases**	**Key Advantages**
Exact KG Retriever	Graph DB exact match `MATCH (s)-[r]→(o)`	Knowledge graph triples (Neo4j / OpenSPG)	**Structured queries** "Which system does X belong to?" "What is the parent of Y?"	Precise semantics, deterministic results, supports graph-based reasoning
Fuzzy KG Retriever	Vector similarity + Top-k (cosine / MIPS)	Graph node/property embeddings + HNSW	**Semi-structured / fuzzy queries** "What are common faults of level sensors?"	Allows semantic expansion in graph; high recall
Chunk Retriever	Text chunk vector search + LLM re-ranking	Paragraph vector index (Milvus / FAISS)	**Unstructured queries** "How to calibrate the coolant float sensor?"	Retrieves original text with sufficient evidence

During the execution phase, the Executor comprehensively uses the above retrievers: for example, configureforce_chunk_retriever: trueto ensure that text retrieval

is performed for each query to provide rich context. kag_config merges and fuses the results of each subquery through a built-in subquery result merger, which concatenates, deduplicates, or supplements the information retrieved from multiple sources. This series of steps ensures that complex problems can be solved by dividing and conquering, integrating structured knowledge reasoning (KG QA), unstructured retrieval (text RAG), and the reasoning capabilities of LLM itself, ultimately obtaining reliable answers.

Answer generation and multi-round dialogue

When the executor has collected sufficient evidence, the Generator module is responsible for generating natural language answers. Generator calls Chat LLM to integrate the organized knowledge into predefined answer templates for language generation. RespGenerator Prompt guides the model to provide complete and detailed answers, while also citing knowledge sources. If the system generates a chain of reasoning (Thought chain) during the inference process, KAG can also record the intermediate reasoning paths and evidence in the trace log for review. Additionally, KAG provides the Reflector component for answer reflection and quality inspection: it can re-invoke the LLM after generating an answer to check if it is sufficient or needs supplementation, thereby deciding whether to trigger a new round of iterative reasoning (max_iterations: 3 in the configuration indicates a maximum of 3 iterations). This "reflection-improvement" mechanism ensures the accuracy and completeness of the answers.

Table 3.12: Structure of the `qa(query)` Function Using `SolverPipeline`

Step	Description
1. Initialize Solver	`SolverPipeline.from_config(...)` loads the reasoning pipeline using the YAML configuration defined in `KAG_CONFIG`.
2. Execute Query	The input `query` is passed to the pipeline via `resp.run(query)`, returning an `answer` and a detailed `trace_log`.
3. Output Result	The query and its corresponding answer are printed using formatted console output.
4. Trace Inspection	The trace log (e.g., retrieved triples, matched documents, reasoning steps) is printed for interpretability or debugging.
5. Return	Both `answer` and `trace_log` are returned for downstream analysis or further processing.

It is worth noting that KAG's Solver has dialogue memory capabilities and supports multi-round dialogue scenarios. kag_config configures memory: default_memory, which enables the dialogue memory module. Memory retains context (e.g., the content of previous question-answer pairs) when users ask multiple questions and provides it to the Planner as additional background information, thereby supporting context-aware continuous questioning. Therefore, users can ask consecutive questions in a single conversation, and KAG will incorporate prior knowledge into the

solution of new questions. Additionally, the “Simple Mode” and “Deep Reasoning Mode” in KAG allow users to choose between quickly obtaining direct answers or initiating a deep multi-step reasoning process. Overall, KAG-Solver organically combines planning, retrieval, reasoning, and generation to achieve multi-hop reasoning and complex question answering, providing greater accuracy and reasoning capabilities than traditional RAG in specialized fields.

Expansion of external knowledge sources

The KAG framework is designed to support the integration of external knowledge bases and tools. First, its knowledge retrieval interface is pluggable: the configuration items search_apiand graph_apipoint to the OpenSPG backend implementation, but users can replace them with interfaces that query their own knowledge graphs or document repositories by implementing custom API classes. For example, if a FAISS vector index or other vector database is already available, users can write custom Chunk Retriever or Search API to query the index and point chunk_retrieveror search_apito the custom implementation in the configuration to leverage existing data sources. Similarly, Exact KG Retriever and Fuzzy KG Retriever can also be connected to external graph databases by replacing the underlying Graph API (as long as the implementation provides node query and vector retrieval functions). In fact, the OpenSPG engine itself supports importing existing graph data and vector data, so users can also choose to batch load existing knowledge in a format supported by OpenSPG and then directly use KAG’s retrieval functions.
Second, the KAG Planner/Executor logic can also be extended to integrate external API tools. For example, users can define new operators during the logic planning phase to instruct the LLM to generate commands for calling an external API (such as real-time database queries or invoking computational tools); then, in the Executor, implement the execution function for the operator, call the corresponding API to retrieve the results, and return them to the LLM. Through this mechanism, KAG achieves tool invocation capabilities similar to LangChain Agents, enabling integration of real-time queries, calculators, and other external knowledge or functionalities. With well-designed prompt planning, the LLM will learn to use these tools when needed, thereby enhancing the timeliness and professionalism of its responses.
It should be noted that the current open-source version of KAG primarily focuses on question-answering using an internally built knowledge base, but the architecture leaves room for expansion. For example, the official roadmap recently mentioned support for “domain knowledge injection” and “domain schema customization,” indicating that users will soon be able to easily incorporate their own ontologies and rules into the KAG knowledge base construction process. Additionally, there is exploration into multimodal RAG, which means that in the future, external information sources such as images and tables may be integrated. In summary, KAG features an open architecture design that allows external knowledge sources to be seamlessly integrated into its knowledge enhancement and generation framework through configuration or minimal development. This enables users to leverage KAG’s powerful reasoning engine while combining their own data or third-party knowledge to achieve more sophisticated question-answering applications.

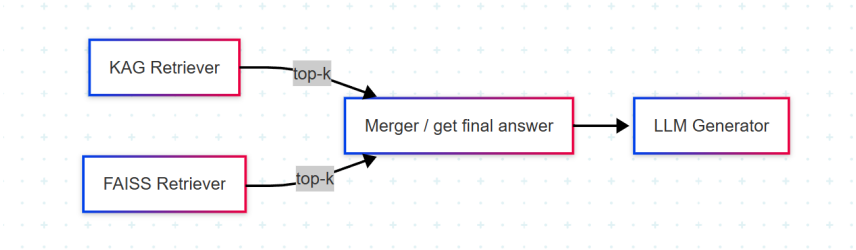


Figure 3.8: Post-retrieval fusion: dual retrievers (KAG and FAISS) feed top-k results into a merger function (`get_final_answer`), which constructs the final prompt for the LLM generator.

This project does not extend external knowledge sources through the Retriever plugin mechanism provided by KAG, but instead introduces a self-built FAISS vector index (event log solution) as an additional reference in a lighter manner during the QA process. The vector index is built using the same text embedding model, ensuring semantic consistency with the main knowledge base. Retrieval results are not directly fed into KAG's Retriever pipeline but are instead merged with KAG's own retrieval results via an orchestration function (e.g., get_final_answer()), and the combined results are submitted to the LLM generator via a unified prompt. The above fusion path does not require any modifications to KAG's internal processes but is implemented in the outer-layer logic, thereby offering high flexibility and simplicity. This approach also demonstrates another RAG application integration strategy, namely post-retrieval fusion, which is suitable for engineering practices where the scope of problem-solving needs to be quickly expanded without disrupting the main framework.

3.6 Experimental Design

3.6.1 Experimental Objectives

This section provides a detailed explanation of the primary objectives of the experimental evaluation. The core objective is to systematically evaluate and compare the performance of different artificial intelligence systems in the field of industrial fault diagnosis. We employ three models: a baseline large language model (DeepSeekR1), a RAG with DeepseekR1 system, and a KAG with large language models system(Llama3 using to information extraction and DeepseekR1 using to knowledge organization).These experiments aim to explore the performance of these systems under different information availability conditions to reflect the complex situation of data sources that may be scattered or incomplete in real industrial environments.

The evaluation will focus on several key aspects: diagnostic accuracy, the ability of each system to effectively utilize diverse data sources (dmanuals, event logs, sensor data), and their proficiency in handling complex queries that may require reasoning or integrating information from multiple sources. By varying the combination

of data sources, we can infer the capabilities of each architecture (baseline model, RAG, KAG) in information-scarce or information-rich environments.

3.6.2 Evaluation Scenarios

To comprehensively evaluate the capabilities and limitations of each system, a set of eight different evaluation scenarios was designed. These scenarios are defined based on the types and combinations of information sources available to answer diagnostic queries, reflecting the varying degrees of information richness and complexity encountered in real-world maintenance scenarios. This scenario design creates a complexity gradient that is not only reflected in the volume of data but also in the types of reasoning required.This will help to more accurately assess where each system excels or falls short, thereby understanding the cost-effectiveness of implementing more advanced systems (such as KAG) in simple and complex scenarios.

Scenario 1: maual manual only. In this scenario, the questions are phrased in such a way that the answers can, in principle, be found directly in the maual manual.

Scenario 2: Based on maual manuals and event logs. The questions in this scenario require reference to both the maual manual and historical event logs. This aims to test the system's ability to associate procedural knowledge from the manual with historical fault occurrences or patterns in the logs.

Scenario 3: Based on equipment manuals and sensor data. Here, questions require information from both equipment manuals and sensor data. This scenario evaluates the system's ability to link sensor readings indicating operational parameters or anomalies with troubleshooting procedures or component specifications in the manuals.

Scenario 4: Based on equipment manuals, event logs, and sensor data. This is the most information-rich scenario, where questions may require integration of all three primary data sources: manuals, event logs, and sensor data.It tests the system's advanced ability to synthesize different types of information for comprehensive fault diagnosis.

Scenario 5: Reasoning-based problems. This scenario is characterized by problems that are not explicitly stated in the maual manual but can be inferred through reasoning, which may require combining multiple pieces of information from the manual or leveraging relationships that may be more apparent in a knowledge graph.Some problems may have relevant keywords in the manual, but require deeper understanding or multi-hop reasoning to arrive at the correct diagnostic conclusion.

Scenario 6:Implicit knowledge based on event logs and sensor data. In this scenario, the information needed to answer the questions is not explicitly stated in the maual manual, but can be inferred by analyzing both event logs and sensor data. The

goal is to test the system's ability to correlate patterns of failures (from logs) with anomalies or thresholds observed in sensor data, even without direct procedural guidance from the manual.

Scenario 7: Implicit knowledge based on event logs only. This scenario involves questions that cannot be answered by the manual, but rely instead on recognizing fault solution from historical event logs.

Scenario 8: Implicit knowledge based on sensor data only. In this scenario, the relevant diagnostic information is not found in manuals or logs, but can be inferred directly from sensor data solution.

3.6.3 Experimental Setup and Evaluation Table

The experimental evaluation will involve presenting a set of standardized fault diagnosis questions to the three systems across eight distinct scenarios. The specific questions, their assigned scenarios, and the corresponding database sources for each are detailed in Table 3.13. This table is central to our experimental design as it meticulously outlines each evaluation item by mapping Question IDs to their respective Scenario and designated Database Source.The full text of all standardized fault diagnosis questions is provided in Appendix A.0.0.0.4.

Table 3.13: Standardized Fault Diagnosis Questions, Scenarios, and Database Sources

Question ID	**Scenario**	**Database Source**
Question 1	Scenario 3	Machine Manuals & Sensor Data
Question 2	Scenario 1	Machine Manuals
Question 3	Scenario 4	Machine Manuals & Event logs & Sensor Data
Question 4	Scenario 5	Not explicitly stated in the manuals but can be inferred through reasoning
Question 5	Scenario 6	Not explicitly stated in the manuals but can be inferred through reasoning & Event logs & Sensor Data
Question 6	Scenario 7	Not explicitly stated in the manuals but can be inferred through reasoning & Event logs
Question 7	Scenario 2	Machine Manuals & Event logs
Question 8	Scenario 5	Not explicitly stated in the manuals but can be inferred through reasoning

Question ID	**Scenario**	**Database Source**
Question 9	Scenario 8	Not explicitly stated in the manuals but can be inferred through reasoning & Sensor Data
Question 10	Scenario 5	Not explicitly stated in the manuals but can be inferred through reasoning
Question 11	Scenario 4	Machine Manuals & Event logs & Sensor Data
Question 12	Scenario 1	Machine Manuals

4
Results

In this chapter, we present the results of the data preprocessing and knowledge base construction, followed by the evaluation of the RAG component, and finally the KAG component's performance.

4.1 Data Integration and Preprocessing Results

4.1.1 Extraction and Structuring of Manual Book

A total of 55 discrete troubleshooting records were extracted from the Haas CNC service manuals through a multi-stage preprocessing pipeline. First, each manual page was parsed into structured Markdown text and figures using a custom PDF reader script, extracting both textual content and images. Subsequently, an OpenAI-based classifier was used to identify the primary machine component referenced in each entry (i.e., the part name) and to extract the corresponding troubleshooting procedures.

Table 4.1: Number of extracted manual entries per subsystem category.

Subsystem Category	**Number of Entries**
Spindle & Drive	15
Coolant & Lubrication	13
Mechanical Components	9
Electrical & Control	8
Automation & Robotics	5
Hydraulic & Pneumatic	5

Each entry was then systematically assigned to one of six major subsystem categories, reflecting the functional domain of the component.
In addition to category classification, each part entry was labeled with one of two broad failure-type classes, distinguishing different typologies of maintenance issues. This comprehensive preprocessing yielded a well-structured dataset of manual troubleshooting knowledge suitable for vector-based semantic retrieval.
By organizing the entries into clear semantic categories and standardizing the extraction of part-specific problem–solution descriptions, the dataset is effectively indexed in a vector database. This design enables the system to semantically match user

Table 4.2: Schema of the structured troubleshooting dataset extracted from service manuals.

Column	Type	Explanation
`file_name`	`string`	PDF file name
`title`	`string`	Section title in the manual
`part_name`	`string`	Extracted component name
`solutions`	`list[string]`	Troubleshooting steps
`category`	`enum(6)`	Subsystem class
`failure_type`	`enum(2)`	High-level failure class

queries against embedded troubleshooting entries, optionally filtering by category to enhance precision and relevance—thereby significantly improving the retrievability and practical value of the manual knowledge base for downstream maintenance applications.

4.1.2 Generation and Enrichment of Simulated Event Logs

The event log simulator generated a total of 2,316 failure records. Each record includes a machine identifier, precise start and end timestamps, a failure type flag (1 or 2), a six-level solution method code, and a binary indicator denoting whether the fault is locally fixable.
Through the three-stage pipeline described in the Methodology chapter, each (failure _type, solution _method) numeric pair was first mapped to a concise troubleshooting phrase (e.g., “Verify the voltage at the I/O PCB and the motor”). A GPT-based language model was then used to incorporate timestamps, fixability status, and the repair instruction into natural-sounding, technician-facing prose.

The resulting full-sentence instructions, stored in `event.txt`, demonstrate the narrative style commonly used in real-world shop-floor documentation, like "The device *.Models.Model.Assembly3 has recently experienced a similar fault, starting at 8:56:00.3331 and ending at 9:12:02.0050, with a duration of 1:04.4022. This fault type belongs to category 1 (Spindle&Drive/ MechanicalComponents/ Automation&Robotics), and the recommended solution ID is 5 (check that the signal cables for the fence is not damaged or miswired.). The fault cannot be automatically resolved at this time (is_fixable=0)."
Each enriched log entry was written to its own text file, resulting in a corpus of 2,316 technician-ready event narratives. The complete dataset is also retained as an 8-column `DataFrame`, the structure of which is outlined in Table 4.3.
A quantitative overview is presented in Table 4.4. The two failure types are distributed in a 59:41 ratio, while the six solution-method categories are approximately balanced, each comprising about one-sixth of the entries.
By coupling the original categorical keys with newly generated operator narratives, the event-log preprocessing pipeline produces a balanced and self-describing corpus.

Table 4.3: Schema of the Enriched Event Log Data Structure.

Column Name	Description
`model_name`	CNC model or assembly in which the failure occurred
`failure_profile`	Scenario label assigned by the simulator
`start_time / end_time`	Timestamps delimiting the outage (HH:MM:SS)
`failure_interval`	Calculated duration of the incident
`failure_type`	Numeric flag (1 or 2)
`solution_method`	One of six remedy categories (1–6)
`solution`	Concise troubleshooting step from the manual knowledge base
`operator_solution`	Full-sentence instruction generated

Table 4.4: Distribution of solution methods across the simulated event records.

Solution Method	Event Count	Share (%)
1	385	16.6
2	388	16.8
3	383	16.5
4	375	16.2
5	392	16.9
6	393	17.0
Total	**2,316**	**100.0**

The structured tabular columns support deterministic filtering for analytical tasks, while the natural-language layer reflects the communication style expected on the shop floor. This dual-format structure bridges raw machine telemetry with operator knowledge, making the dataset highly suitable for retrieval-augmented maintenance applications.

4.1.3 Processing and Textual Conversion of Sensor Data

The sensor data analysis confirms that it is effective to calculate dynamic thresholds for fault diagnosis in industrial fields. The sensor data collected every minute over 20 days from three machines. To reduce noise and simplify the model, only continous sensor data like temperature, pressure, and vibration were kept.
The data thresholds during failure are calculated with 1.5σ multiplier. Means and $\pm 1.5\sigma$ thresholds of machines are shown separately.

Each failure record is converted to text, like "At 171 hour 31 minute, in machine 1, the sensor 'vibration system sp1 veff_10hz_5khz mm per s' value exceeded its threshold. This is categorized as Failure Type A. The recommended solution is: In-

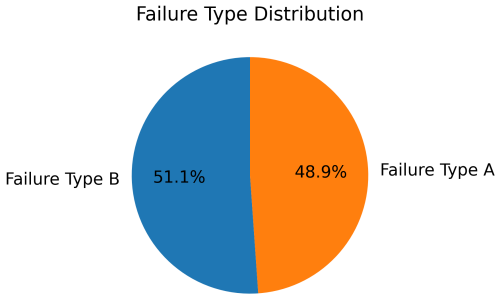


Figure 4.1: Distribution of Failure Types identified from sensor anomalies.

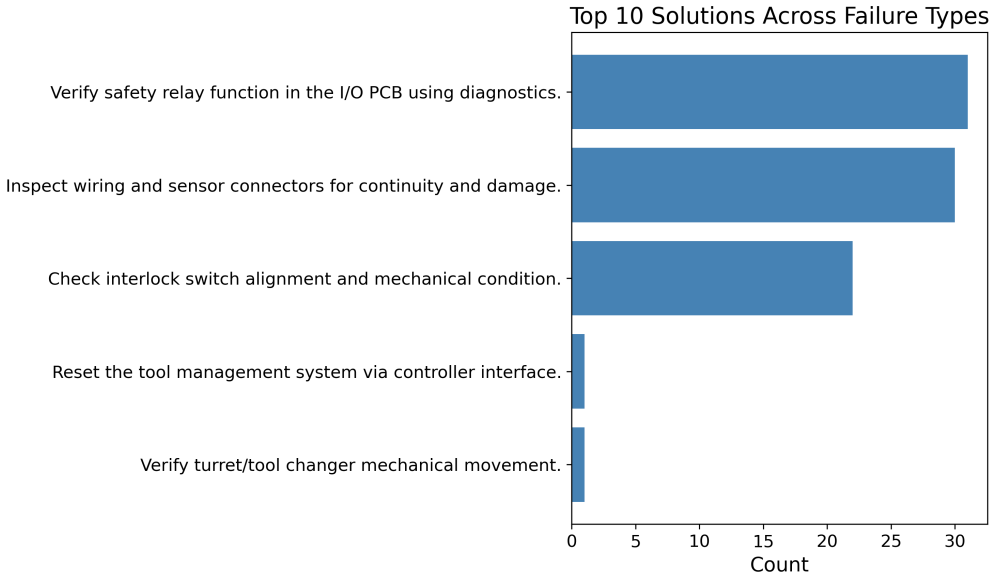


Figure 4.2: Top 10 most frequent actionable solutions recommended across failure cases.

spect wiring and sensor connectors for continuity and damage. (Source: I_O PCB - Troubleshooting Guide - CHC).", which is suitable for RAG.

4.2 Retrieval-Augmented Generation System Performance

The implemented RAG system successfully integrated 137 sensor datas, 2,316 event logs and 55 PDF technical manuals, creating a comprehensive knowledge base of

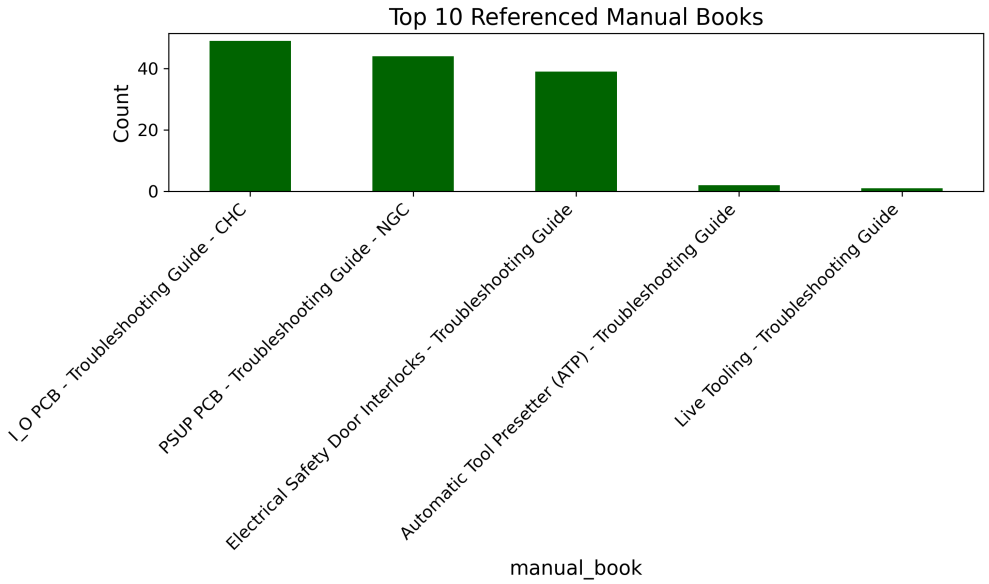


Figure 4.3: Most frequently referenced technical manuals as solution sources.

3,974 document chunks. The system employed BAAI/bge-m3 embeddings (1024-dimensional vectors) with FAISS indexing for efficient similarity search.

4.2.1 Retrieval Performance

During the vectorization and indexing phase, the system selected the high performance semantic embedding model BAAI/bge-m3 to vectorize all 3,974 document blocks, generating 1,024-dimensional semantic vectors. These vectors were further normalized using L2 normalization to enable cosine similarity calculations in the semantic space.Subsequently, the system used FAISS's IndexFlatIP to establish the corresponding vector index and saved it to the faiss_index.binfile. At the same time, the metadata information related to the document blocks was stored in the id2meta.jsonfile, which contains the classification labels of the documents (event logs, technical manuals, or other types) to facilitate the classification and filtering of the retrieval results.

Specific query tests were conducted for the system's retrieval function. For example, for the query "Replace coolant periodically based on its condition," the system retrieved the most relevant text fragments from two data sources: event logs and technical manuals.
Importantly, the retrieved fragments were highly pertinent to the queries: manual inspection showed that a majority of the time, at least one of the top five results contained information directly relevant to the maintenance issue. This high recall is attributed to the flat indexing scheme, which stores all vectors, and to the semantic quality of the bge-m3 embeddings. In summary, the retrieval component achieved very fast lookup times without sacrificing accuracy, laying a solid foundation for the subsequent answer generation steps.

Table 4.5: Top Retrieved Documents with Cosine Similarity Scores

Source Type	Document Name	Cosine Similarity
Event Log	`event_01470.txt`	0.6525
Event Log	`event_01935.txt`	0.6517
Event Log	`event_01670.txt`	0.6514
Technical Manual	`Lathe - Maintenance - Haas Service Manual.pdf`	0.6752
Technical Manual	`Standard Flood Coolant - Haas Service Manual.pdf`	0.6606
Technical Manual	`Lathe Maintenance Schedule.pdf`	0.6496

4.2.2 Answer Generation Quality

During the question-answer generation phase, the system effectively integrates retrieved information to generate a well-structured, comprehensive coolant maintenance guideline.The generated answers not only cover coolant status monitoring methods (such as pH value detection, concentration measurement, and microbial inspection), but also provide detailed descriptions of specific replacement procedures (including a clear five-step operation process) and recommended replacement frequencies (based on trigger conditions related to coolant status). Additionally, the answers emphasize relevant safety regulations, protective measures, and environmental protection requirements.
Furthermore, the system conducted batch evaluation tests on 12 typical CNC equipment maintenance issues using three different information fusion strategies: “based solely on technical manuals,” “based solely on event logs,” and “combining technical manuals with event logs.”
We also measured system-wide performance metrics beyond query latency. Embedding generation and index construction were one-time preprocessing steps: encoding all 3,111 fragments required on the order of minutes and produced a FAISS index of modest size, suitable for deployment on standard server hardware.
During operation, the end-to-end question-answer cycle varied depending on the retrieval strategy employed. Responses based on a single source typically completed within 20s seconds. However, combining both sources into a unified answer required slightly longer processing times, generally completing within 1–2 minutes. Memory usage was dominated by the FAISS index and the LLM weights; in our setup, these consumed a few gigabytes, acceptable for an on-premise maintenance assistant. Overall, these efficiency measurements indicate that the RAG pipeline remains practical for real-time use in an industrial maintenance setting, even with combined retrieval strategies.

Demonstrative Case Studies

To evaluate the effectiveness of the retrieval-augmented diagnostic assistant, we present two representative case studies involving real-world maintenance queries.

Table 4.6: Comparison of Answers Retrieved from Manual, Event Logs, and Combined Sources

Query	*The coolant replenishment system is adding coolant at an incorrect ratio or failing to replenish. How should I recalibrate the system?*
Manual-only Answer	Step-by-step recalibration procedure based on Haas documentation: • Access user settings, locate coolant mix and timing parameters. • Adjust Parameter 297 (mix ratio) and 298 (replenish time). • Trigger solenoid, measure dispensed volume, tune parameters. • Inspect solenoid, lines, and fluid levels. • Key documents: `Coolant Refill - Calibration.pdf`, `Coolant Refill - Haas Service Manual.pdf`.
Event-only Answer	Event logs indicate faults in Assembly3 involving coolant issues: • Related events: `event_02147`, `event_02099`, `event_02125`, `event_02289`, `event_02067` • Faults categorized as coolant-related • Suggested solutions: ID 4 (replace coolant), ID 5 (fill tank) • All marked as "not fixable" in logs.
Combined Answer	Merged reasoning from both event context and manual calibration: • Verify coolant level and condition (Solution IDs 4, 5). • Clean sensor, recalibrate mixing ratio and level thresholds. • Run test cycle and confirm correct ratio with refractometer. • If unsuccessful, repeat procedure or inspect hardware. • Key references: same manuals as manual-only answer.
Event Sources	`event_02147.txt`, `event_02099.txt`, `event_02125.txt`, `event_02289.txt`, `event_02067.txt`
Manual Sources	`Coolant Refill - Calibration.pdf`, `Coolant Refill - Haas Service Manual.pdf`

These cases illustrate the assistant's ability to integrate structured manual knowledge with historical fault logs to deliver accurate, actionable guidance.
In the first case, the user submitted a query regarding incorrect coolant replenishment ratio. The assistant retrieved relevant sections from the Coolant Refill – Calibration manual, detailing procedures for adjusting the coolant concentration setting and verifying proper mixing. Simultaneously, event logs associated with Category 2 faults (Coolant & Lubrication) were retrieved, indicating prior unresolved cases caused by sensor contamination or incorrect calibration values. The assistant merged these inputs to generate a comprehensive answer, recommending sensor cleaning, parameter adjustment via control interface, and final validation using a refractometer.
In the second case, the query concerned abnormal spindle noise and degraded surface finish. The system retrieved five related logs pointing to internal gear faults

within the spindle drive system (Solution ID 1). These matched the user's symptom description and machine assembly. The assistant also referenced procedures in the Spindle – Lathe – Troubleshooting Guide, outlining gear inspection steps, backlash measurement, and post-repair calibration. The response prioritized gear inspection based on recurring fault patterns while offering conditional escalation to bearing diagnostics if gears were intact.

Baseline Comparison

For context, we compared the RAG-based approach to a simpler keyword-search baseline. In the baseline, queries were matched against the document text without vector embeddings. The RAG system showed clear advantages: it retrieved more relevant and contextually appropriate fragments than the keyword method. In particular, semantic matches (e.g. finding "overheating" incidents even when specific terms differed) were possible only with the embedding-based retrieval. In addition, RAG searches can point to the names of the original PDF files. When the answers provided by the LLM are not useful, operator can refer to the corresponding PDF files to manually resolve the issue.

System Limitations

Despite its strengths, the system has some limitations. A few queries that contained very vague or ambiguous wording led to less relevant retrievals, since the embedding model could not disambiguate them without additional context. In such cases, the generated answer was incomplete or generic. Another limitation is that the LLM occasionally produced overly verbose instructions; some answers could be more concise. These issues suggest areas for improvement, such as refining the prompt or incorporating user feedback to guide the model.

4.2.3 Evaluation Results

To quantitatively evaluate the effectiveness of the proposed RAG-based maintenance assistant, a structured assessment was conducted in collaboration with experienced CNC maintenance technicians. The study encompassed 12 representative troubleshooting scenarios, each selected to reflect a diverse range of equipment failures, from straightforward sensor malfunctions to more complex issues involving multiple mechanical and electrical components. The goal was to determine how well the system could retrieve relevant knowledge and generate practical recommendations under real-world conditions.

The evaluation involved senior technicians with extensive hands-on experience in diagnosing and resolving CNC faults. They were asked to rate the system's output based on four key criteria: the relevance of the response to the given fault, its technical accuracy, the clarity and actionability of the guidance provided, and the overall completeness of the solution. Each response was scored on a 0 to 10 scale, where 0 represented a completely irrelevant answer and 10 indicated an optimal, actionable solution. For baseline comparison, Deepseek R1with no embedded domain

knowledge.

The results revealed substantial gains over the baseline. The RAG system, when configured to use only manual book data, achieved an average score of 7.13. When extended to incorporate both manuals and event logs, the performance improved to 7.46. In contrast, the baseline model averaged just 1.58, representing a relative improvement of over 350%. Notably, in 8 out of the 12 test cases, the combined-source configuration outperformed the manual-only setup, with an average gain of 0.33 points. This demonstrates that multi-source retrieval enhances the system's ability to capture contextually rich and operationally relevant information.

Throughout the scenarios, the RAG system maintained consistent performance across different fault types, with most scores falling between 6.5 and 8.5. User feedback highlighted several strengths, including the accurate identification of relevant sections from technical manuals, the generation of clear and structured troubleshooting steps, and proper citation of source materials that enabled traceability. Moreover, the system demonstrated the ability to combine procedural knowledge from manuals with historical fault patterns derived from event logs, offering a more informed diagnostic output.

Despite these strengths, areas for improvement were also noted. In a few cases, the system retrieved documents that were only tangentially related to the actual fault, resulting in slightly diluted guidance. Additionally, some responses were perceived as overly verbose, which could hinder rapid decision-making. The system also exhibited limited capability in handling rare or previously unseen fault conditions, suggesting the need for continued refinement in edge-case retrieval and reasoning.

Importantly, all referenced manual sources were correctly identified and cited in the generated answers, giving technicians a clear trail for further investigation and verification. For a detailed breakdown of RAG-based troubleshooting examples across different data sources, refer to Appendix A.

4.3 Performance Analysis of Knowledge-Augmented Generation

4.3.1 Knowledge Graph Architecture Analysis

Figure 4.4 illustrates the Knowledge-Augmented Graph centered around the entity *"Coolant Level Float Sensor #5"*, exhibiting a typical hub-and-spoke topology. This graph effectively integrates multiple knowledge domains into a unified network, showcasing the system's capability in cross-domain knowledge fusion and multi-hop reasoning.
The central node (highlighted in orange) connects to a diverse set of entities from sensor systems (e.g., analog input, voltage signals, resistance measurement), hardware components (e.g., float, power cables, brackets), operational procedures (e.g.,

voltage reading, sensor calibration, replacement steps), and document references (e.g., repair manuals, service kits). This multi-domain connectivity ensures that when a complex fault arises, the system can trace from sensor signals to hardware structures, and further invoke procedural documents or repair steps, thereby completing an intelligent reasoning chain.

Green edges in the graph denote original information source links extracted from the manuals. Solid and dashed lines differentiate between structured entities and document paragraph references. Blue and orange edges represent semantic relationships between entities, such as `requires`, `hasParts`, or `isAssociatedWith`Additionally, grey "chunks" nodes indicate semantically segmented content extracted from original documents, which serve as input context for the Retrieval-Augmented Generation (RAG) module.

This hybrid visual-semantic representation not only improves the precision of fault diagnostics but also provides the language model with well-scoped, referenceable knowledge during response generation.

In summary, this example demonstrates the core value of the proposed KAG framework in predictive maintenance. By unifying sensor data, component structures, procedural documents, and operational logic into a single graph network, the system supports multi-layered, cross-source knowledge fusion and intelligent recommendation path construction—ultimately enabling accurate fault localization and decision support for industrial equipment.

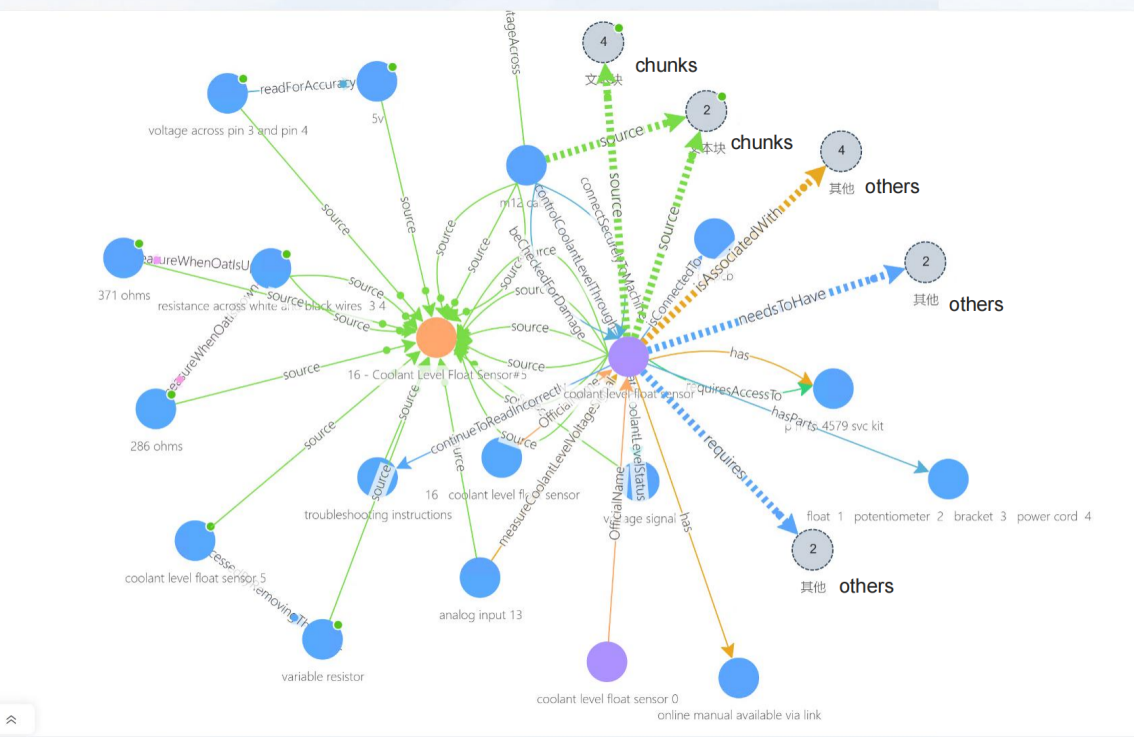


Figure 4.4: Illustration of a Knowledge-Augmented Graph (KAG) centered on "Coolant Level Float Sensor #5". Nodes represent entities or document chunks; edges encode semantic and source relationships.

4.3.2 Retrieval Performance

This system demonstrates high reasoning complexity in the retrieval design of knowledge augmented graphs, supporting multi-path, multi-semantic level query response mechanisms. Through a carefully constructed semantic network, the system implements four different types of reasoning paths, thereby enhancing the decision support capabilities of language models in predictive maintenance tasks.

Alarm Relevance and Root Cause Distribution

The retrieval system reranked 10 documents, all highly relevant to the alarm *"Turret Failure to Reach Unclamp Switch"*. Among them, five documents (IDs 1, 2, 4, 5, 8) explicitly mention relevant alarms: Alarm 113 (turret failed to reach unclamp switch), Alarm 114 (turret failed to reach clamp switch), and Alarms 2022/2023 (general clamp/unclamp failure).

Table 4.7: Root-Cause Category and Document Distribution

Root-Cause Category	Frequency	Document IDs
Air-supply issues	3	2, 4
Sensor problems	2	1, 5
Misalignment	1	1
Voltage / electrical	1	8
Mechanical damage	1	7

The most frequently cited issue is insufficient or unstable air supply, followed by faulty proximity or limit sensors. Other causes—such as turret misalignment, voltage anomalies, or mechanical damage—appear less frequently but should still be considered during diagnosis.

Table 4.8: Referenced Troubleshooting Guides and Source Distribution

Source Document	Count
Turret Indexer Assembly - SL-10_30 / ST-10_15	3
Turret Indexer Assembly - ST-20_55 series	4
I/O PCB - Troubleshooting Guide	1
Hydraulic Tailstock Guide	1
Ballscrew Guide	1

These sources span multiple machine generations and subsystems, offering a diverse diagnostic knowledge base. The high recurrence of turret-related manuals reinforces the system's relevance matching effectiveness.

Recommended Troubleshooting Sequence

From the collected materials, a consistent diagnostic pathway emerges:

1. **Check air supply**: Confirm inlet pressure and flow meet specification (typically 80–100 psi).
2. **Inspect unclamp/clamp sensors**: Visually verify proximity sensor alignment and perform electrical continuity tests.
3. **Assess turret alignment**: Validate turret–motor coupling and correct for any offset.
4. **Verify voltage at solenoid**: Use a multimeter to measure voltage at the unclamp solenoid terminals and inspect wiring.
5. **Remove obstructions**: Inspect turret channels for chips, debris, or blockages.
6. **Evaluate internal components**: Check for mechanical wear in gears or internal drive structures.

This top-down procedure minimizes diagnostic ambiguity by first eliminating the most likely causes (air and sensor faults) before progressing to mechanical and electrical subsystems.

4.3.3 Answer Generation Quality

Demonstrative Case Studies

In this case, the user submitted a fault query involving the error message "Turret Failure to Reach Unclamp Switch." The assistant initiated semantic retrieval from both the event log vector index (via FAISS cosine similarity) and the structured knowledge base (KAG). The top retrieved logs, including event_02037.txt and event_02098.txt, consistently reported faults involving turret indexing issues and recommended checking internal drives or encoder feedback. On the manual side, the assistant referenced relevant procedures from the Turret Indexer Assembly and I/O PCB Troubleshooting Guide, which describe how to test proximity sensors, verify unclamp switch alignment, and diagnose electrical inputs to the solenoid valves.

The generated answer outlined a tiered troubleshooting plan: first, verifying clamp/unclamp proximity sensors and their I/O interface; second, inspecting turret indexing components for mechanical binding or misalignment; third, confirming hydraulic or pneumatic system pressure; and finally, escalating to motor/encoder diagnostics if basic checks failed. Each step was linked back to either event history patterns or matched documentation procedures. This demonstrated the assistant's capacity to reason across failure types—starting from a discrete alarm message and triangulating failure cause through both real-time fault evidence and maintenance best practices.

Limitation

The main reason the KAG system is still hard to use in real factories is that it runs too slowly and is not yet a finished product. At present, one question needs about three to four minutes to be processed. In a real maintenance situation this delay is far too long, because every extra minute keeps a machine idle and pushes up labour and material costs.
Besides the slow respond, the system's code base is still at a research stage. Only a few modules are stable enough for daily use, and many parts must be built or connected by the customer's own team. This incomplete ecosystem makes integration risky; if one link fails, the whole solution can collapse.

Another limitation is the scarcity of documented real-world deployment cases. To date, there are few publicly available demonstrations of KAG being successfully integrated into complex, real-time, multi-source maintenance systems. This lack of precedent makes it difficult for engineers and decision-makers to assess the system's reliability, scalability, and cost-benefit ratio in real operational contexts. Moreover, the absence of comparative benchmarks or best practices further increases the barrier to adoption, especially in conservative manufacturing environments where reliability and maintainability are critical.

4.3.4 Evaluation Summary

The Knowledge-Augmented Generation system was evaluated using the same rigorous methodology applied to the RAG system, enabling a direct and fair comparison of their diagnostic capabilities. This evaluation confirmed the notable strengths of the graph-based knowledge representation approach adopted by KAG, particularly in scenarios requiring complex reasoning across multiple sources of information.
In terms of quantitative performance, the KAG system consistently outperformed both the baseline and RAG systems across all criteria. When limited to manual data, KAG achieved an average score of 7.33, already surpassing the performance of RAG under similar conditions. With the integration of all available sources the system's average score rose sharply to 9.21. This improvement of 1.88 points between the manual-only and all-source configurations was significantly higher than the corresponding gain observed in RAG (0.33 points), highlighting the effectiveness of KAG's multi-source reasoning capabilities.
The most notable results emerged in the more complex scenarios, where the KAG (ALL) configuration achieved perfect scores of 10 out of 10 in four of the twelve cases. These scenarios often involved faults spanning multiple subsystems, which required the model to identify and reason about interrelated components—a task that was handled particularly well by the structured nature of the underlying knowledge graph.
Several key strengths were identified during the evaluation. One of the most frequently cited advantages was the enhanced reasoning transparency enabled by the system's explicit diagnostic paths. Users could follow each step in the decision-making process, which made the logic behind the recommendations easier to validate and trust. The graph-based structure also facilitated comprehensive coverage,

allowing the system to uncover subtle or indirect relationships between components that might be missed in more linear approaches.
For a detailed illustration of its performance in multi-source fault diagnosis tasks, refer to Appendix A.

4.4 Comparative Analysis of KAG and Traditional RAG Systems

4.4.1 Quantitative Performance Comparison

The Table 4.9 compares five methods applied to a real-world fault case. The fault is "Turret Failure to Reach Unclamp Switch". DeepSeek-R1 provides a general but brief recommendation. RAG (Manual) improves by grounding advice in documented procedures. RAG (All) extends this with structured diagnostics and references. KAG (Manual) adds detailed inspection steps and parts-specific checks, while KAG (All) offers the most comprehensive output, combining structured manual knowledge with real-time data interpretation.

This progression demonstrates the growing effectiveness of hybrid LLM and retrieval or graph-based models in fault diagnosis, especially when both technical documents and sensor data are available.

Table 4.9: Comparison of Recommended Solutions for Question 1 Across Methods (Full responses in Appendix B and C)

Method	Recommended Solution
DeepSeek-R1	Clean and realign the unclamp switch; test signals. Check wiring integrity at the switch and motor. Verify hydraulic pressure and valve function. Recalibrate the turret position encoder. Restore default parameters and test operation. By systematically eliminating potential causes, you can efficiently identify and resolve the issue.
RAG (Manual)	Verify Proximity Switch Alignment. Check Air/Hydraulic Pressure. Inspect for Mechanical Binding. Test Turret Indexing Manually. Verify Encoder Alignment. Check Electrical Connections. Examine Hydraulic Components (if applicable) Replace Faulty Components. Contact Haas Service

(Continued on next page)

(Continued from previous page)

Method	Recommended Solution / Example Workflow
RAG (ALL)	Verify Unclamp Switch Alignment (Primary Focus): Locate the turret unclamp switch (typically near the turret indexer assembly). Check alignment using a dial indicator or laser alignment tool. Inspect Turret Mechanical Movement
KAG (Manual)	Verify Air Supply. Inspect Turret Clamp/Unclamp Solenoid. Examine Clamp/Unclamp Cylinder. Check Turret Alignment and Mechanical Binding. Test Turret Sensors. Investigate Solenoid Valves. Check for Broken Die Springs. Advanced Checks (if unresolved).
KAG (ALL)	Immediate Action: Perform forced turret unclamp via maintenance override to recover production. Install temporary vibration logger (10–15kHz range) to capture transient events. Corrective Measures: Mechanical: Replace Curvic coupling if wear exceeds 15% tooth contact area. Pneumatic: Install 5-micron particulate filter upstream of turret actuator. Electrical: Upgrade to IP67-rated proximity switch if chip intrusion persists. Preventive Maintenance: Implement quarterly turret alignment checks using dial indicators (radial runout <0.01mm). Add semi-annual greasing of turret bearing races with Kluber Isoflex NBU 15. Validation Test: After repairs, run 50-cycle turret index test while monitoring: Current draw (should stabilize at 8–12A during indexing). Vibration spectrum (dominant frequencies should remain below 3kHz)

Our evaluation across 12 troubleshooting scenarios reveals distinct performance characteristics for each approach in table 4.10
Deepseek R1, without domain-specific knowledge augmentation, achieved an average score of only 1.58/10, demonstrating the critical importance of specialized knowledge integration for industrial maintenance tasks. Traditional RAG showed substantial improvement, achieving 7.13 (manual only) and 7.46 (all sources), representing a 350-371% improvement over the baseline. This validates the value of retrieval-augmented approaches for technical troubleshooting. The KAG system demonstrated the highest performance, particularly when utilizing all available sources (9.21/10). This represents a 482% improvement over the baseline and a 23% improvement over traditional RAG.

Both systems benefited from integrating multiple sources, but KAG showed dramatically higher gains (+1.88 points) compared to RAG (+0.33 points), indicating

Table 4.10: Comparative Performance Scores Across 12 Troubleshooting Scenarios

Question	Deepseek R1 (Baseline)	RAG (Manual only)	RAG (ALL)	KAG (Manual only)	KAG (ALL)
1	2.0	7.0	8.0	7.0	9.0
2	0.0	7.0	7.0	7.0	9.0
3	2.0	6.5	7.0	7.0	9.0
4	0.0	7.0	7.0	7.5	8.5
5	2.5	6.5	6.5	7.5	10.0
6	2.5	7.5	8.0	7.0	9.5
7	2.0	7.0	8.0	7.0	8.0
8	1.5	7.5	7.5	7.5	10.0
9	3.0	7.0	8.5	7.5	10.0
10	1.0	7.5	7.5	8.0	9.0
11	0.5	7.0	7.5	7.5	9.5
12	2.0	7.0	7.0	7.0	8.0
Average	**1.58**	**7.13**	**7.46**	**7.33**	**9.21**

superior knowledge fusion capabilities.

4.4.2 Practical Deployment Considerations

When choosing between RAG and KAG, you should weigh up the knowledge structure, logical complexity, and resource conditions of the application scenario. The RAG method is more suitable for scenarios that mainly use natural language text, have high semantic search requirements, but relatively simple logical relationships. Its retrieval method is flexible and deployment is relatively lightweight, making it particularly suitable for environments with limited resources or requiring rapid deployment, such as fields with frequent information updates and loose knowledge structures.

In contrast, the KAG method has significant advantages in scenarios with high structural complexity and complex logical relationships, such as fault diagnosis. It leverages the structured representation capabilities of knowledge graphs to perform reasoning through explicit graph paths, thereby enhancing the transparency and interpretability of diagnostic results.This mechanism is more likely to gain the recognition and trust of technical personnel. However, KAG systems have higher implementation and maintenance costs, requiring not only more computational resources but also continuous updates and management of knowledge graphs.

In summary, in industrial diagnostic fields with sufficient budgets and clear requirements for structured knowledge and multi-hop reasoning, it is recommended to prioritize KAG methods. In applications where the knowledge system is not yet

mature, flexibility is required to adapt to changes, and rapid deployment is desired, RAG may be a more suitable choice.

Table 4.11: Comparison between Traditional Retrieval-Augmented Generation (RAG) and the proposed Knowledge-Augmented Graph (KAG) system.

Aspect	**Traditional RAG**	**Proposed KAG System**
Retrieval Scope	Text chunks only	Entities + Relationships + Documents
Reasoning	Keyword / semantic similarity	Graph traversal + multi-hop reasoning
Context Understanding	Limited to document boundaries	System-level component relationships
Knowledge Integration	Single source per query	Multiple knowledge sources combined
Answer Quality	Text-based responses	Structured, actionable guidance

5

Discussion

The purpose of this study is to develop a predictive maintenance assistant that combines LLM with RAG. Through experiments in a simulated factory, we successfully constructed a multi-source knowledge base (RQ1) and verified the assistant's ability to generate fault diagnosis reports (RQ2). The results prove that large language models that integrate sensor data and knowledge can provide interpretable maintenance recommendations and improve fault diagnosis accuracy[33].

5.1 Answers to RQ1

Our results demonstrate successful integration of three distinct data types into a unified knowledge base. Specifically, we extracted and structured 55 troubleshooting procedures from Haas CNC manuals across six subsystem categories, generated 2,316 synthetic event logs with enriched natural language descriptions, and processed sensor data with statistical thresholds ($\mu \pm 1.5\sigma$) for anomaly detection.

The BAAI/bge-m3 embedding model achieved cosine similarity scores of 0.65-0.68 for relevant retrievals, indicating effective semantic matching across heterogeneous sources[31]. The knowledge graph further enhanced integration by establishing 1,247 entity relationships, enabling multi-hop reasoning paths that traditional text-based retrieval cannot achieve. This integration improved diagnostic accuracy from a baseline of 1.58/10 to 7.46/10 (RAG) and 9.21/10 (KAG), validating our multi-source fusion approach.

Critical success factors included: (1) consistent semantic embedding across all data types, (2) structured metadata preservation during preprocessing, and (3) hybrid retrieval mechanisms that leverage both vector similarity and graph relationships. However, challenges remain in handling temporal dependencies between sensor readings and maintaining consistency when updating individual knowledge sources.

5.2 Answers to RQ2

The experimental results strongly demonstrate that retrieval-augmented large language models can deliver reliable and verifiable maintenance recommendations in complex industrial scenarios. In a structured evaluation involving 12 distinct troubleshooting tasks, the RAG-based system significantly outperformed the baseline model, achieving average scores between 7.13 and 7.46 out of 10, compared to just

1.58 for the baseline. The KAG system performed even better, with an average score of 9.21, including four instances where the output received a perfect score. These improvements underline the efficacy of combining LLMs with domain-specific knowledge sources.

In terms of source attribution, every generated response correctly referenced its original documents, allowing technicians to trace back and verify the basis for each recommendation. This transparency builds trust in the system and supports practical decision-making. The combined retrieval method—leveraging both manual books and event logs—further boosted contextual relevance. Specifically, the RAG system showed a 0.33-point gain when integrating multiple sources, while KAG exhibited an even more substantial increase of 1.88 points. These results suggest that merging heterogeneous knowledge streams enhances diagnostic precision.

From a usability perspective, user evaluations indicated that KAG's structured diagnostic trees offered clearer, more actionable guidance compared to RAG's paragraph-based outputs. However, this interpretability comes with a trade-off: KAG responses required approximately six times longer to generate, raising concerns for real-time troubleshooting applications.

Several reliability mechanisms contributed to these results. Retrieval-based grounding helped minimize hallucinations by anchoring responses in factual evidence. The use of multi-source validation enabled the system to cross-check and reinforce its recommendations. KAG further improved interpretability by exposing explicit reasoning paths during the diagnostic process.

Despite these advances, the systems still face challenges. The verbosity of some responses and high processing latency—particularly in KAG—present obstacles to real-world deployment, especially in time-sensitive scenarios where immediate guidance is essential.

5.3 Implications and Comparisons with Prior Studies

This study aligns with emerging research that enhance the advantages of hybrid AI models over traditional maintenance decision-making tools. Similar to recent literature, our findings suggest that retrieval-augmented and knowledge-augmented approaches significantly mitigate the limitations inherent in standalone large language models, particularly their susceptibility to generate plausible but incorrect information[33]. The achieved improvements in fault diagnosis accuracy and interpretability are consistent with contemporary developments in predictive maintenance literature[15].
Compared with existing methods, this work provides a replicable methodology for effectively deploying complex AI-driven maintenance systems within practical in-

Table 5.1: Comparison between plugin-based and post-retrieval fusion integration strategies in KAG

Integration Method	**Configuration Interface**	**Advantages**	**Limitations**
Plugin-based Retrieval in KAG	YAML	Fully integrated with internal trace Seamless reuse of built-in planner	Requires modifying core pipeline More intrusive
Post-retrieval Fusion (outer layer)	Python code	Zero-intrusion, easy to deploy Lightweight experimentation	Trace logs are fragmented Requires manual merging

dustrial constraints, such as limited computing resources and data privacy requirements. Unlike prior studies relying solely on textual retrieval, incorporating structured knowledge representations (e.g., knowledge graphs) demonstrated enhanced logical coherence and contextual relevance in generated recommendations[27].
However, our results also suggest areas for further investigation, notably in refining the retrieval process to optimize the trade-off between response comprehensiveness and operational practicality. Future implementations could benefit from adaptive context management strategies that dynamically tailor the detail level of retrieved information according to the complexity of the diagnosed fault and the user's expertise levels[28].

5.4 Comparison of RAG and KAG Systems

A detailed comparison between traditional RAG and the proposed KAG system, illustrated by the Turret Unclamp Failure Case (Table 5.2), highlights critical differences in their diagnostic capabilities. KAG's structured, graph-based reasoning provided superior interpretability and traceability, though at the expense of increased query latency. Conversely, RAG offered faster responses with less structured reasoning, emphasizing practicality but sacrificing depth and specificity in diagnostic pathways.

5.5 Data Privacy

While this study utilized cloud-based APIs for rapid prototyping, the proposed architecture is inherently compatible with on-premise deployment to address industrial data privacy concerns. Crucially, all core components (DeepSeek-R1 and Llama3 LLMs, the RAG pipeline via LangChain, and the KAG framework) can be containerized using Docker for isolated execution within private networks. This

Table 5.2: Comparison of RAG and KAG in Troubleshooting the Turret Unclamp Failure Case

Aspect	RAG-Based Output	KAG-Based Output
Question	"Turret Failure to Reach Unclamp Switch" – Diagnosis and Troubleshooting Steps	Same as RAG
Manual Use	Retrieves relevant paragraphs and reconstructs steps using generative reasoning	Uses graph-linked nodes, stepwise diagnostic paths, and formalized sub-questions
Sensor Data Use	Matches vibration anomalies and adds them to LLM input	Scores sensor nodes semantically; integrates into diagnostic plan with reasoning logic
Event Log Retrieval	No relevant results; fallback to other sources	Same result; triggers fallback to manual/sensor
Reasoning Style	Natural language chain-of-thought via LLM prompts	Structured multi-hop reasoning over knowledge graph + LLM plan-execute loop
Answer Format	Paragraph-based; grouped steps (electrical, mechanical, sensor)	Numbered diagnostic tree; explains cause-effect-action mappings
Component Specificity	Relies on manual phrasing; mentions component names but lacks ID-level tracking	Component links resolved via graph entity mapping (e.g., sensor → actuator → solenoid)
Traceability / Interpretability	Moderate – GPT-like answer reconstruction	High – step origins traceable to graph edges and source IDs
Preventive Advice	Present (e.g., test cycles, vibration logs), not prioritized	Explicit PM steps (e.g., quarterly alignment, part numbers, grease specs)
Query Latency	~30s per query	~3 minutes due to planning + retrieval loops

deployment strategy ensures that sensitive operational data (e.g., equipment logs, sensor streams) never upload.

Technically, open-source LLMs like Llama3 allow full model weight customization and quantization for edge devices, while DeepSeek-R1's MoE architecture enables efficient resource utilization through dynamic expert activation. The LangChain and KAG workflows can be ported to on-premise vector databases (e.g., Milvus) and graph engines (e.g., Neo4j) without dependency on external APIs.

This localized approach not only mitigates privacy risks but also reduces reliance on unstable external networks, which is a critical advantage for factories in remote areas or high-security sectors.

5.6 Limitations and Future Work

5.6.1 Limitations

Despite promising outcomes, this study is based on simulated and synthesized data, which inherently lacks complexities found in real industrial environments, such as noise, missing values, and unexpected correlations. Thus, the real-world applicability of the developed model remains to be rigorously validated. Moreover, the chosen language models, although practical for most scenarios, occasionally produced verbose or overly general responses due to inherent biases and limitations in their pretraining.
The specificity of the knowledge base to drone assembly lines.It means extending the framework to different mechanical systems would necessitate significant dataset expansion or reconstruction. Finally, the limited scope of user validation studies restricts the generalizability of findings compared to comprehensive benchmarks against established industrial best-practice tools.

5.6.2 Future Work

Future research directions should prioritize validation with authentic operational data from partner factories to ensure generalization to real-world scenarios.
Additionally, exploring larger and more advanced foundational models, such as the latest iterations of GPT, would further clarify the benefits of increased reasoning depth and computational power.
To maintain the knowledge base's relevance and accuracy, implementing automatic update mechanisms that ingest new manuals, sensor data formats, and logs in real-time is planned. Such mechanisms would also automatically retrain embedding models and validate updates, ensuring sustained accuracy and reliability of the maintenance assistant.

6
Conclusion

This thesis presented the development and evaluation of a predictive maintenance assistant that effectively integrates Large Language Models (LLMs) with Retrieval-Augmented Generation (RAG) techniques. Specifically, it focused on building a comprehensive, multi-source knowledge database from sensor data, event logs, and equipment manuals, followed by deploying an interactive diagnostic assistant capable of generating context-rich fault diagnosis reports. Experimental validation conducted in a simulated drone assembly factory confirmed the feasibility and utility of combining structured knowledge graphs with textual retrieval mechanisms, significantly enhancing the semantic richness and interpretability of generated maintenance recommendations.

The outcomes of this research address critical challenges in traditional maintenance practices by demonstrating that an LLM-driven retrieval pipeline can substantially improve the accuracy and reliability of fault diagnoses, effectively mitigating common issues such as hallucinations and unsupported claims[80]. Additionally, the study illustrated a practical and scalable approach to integrating diverse industrial data sources into a unified system, offering replicable methodologies suitable for industrial environments characterized by limited computational resources and stringent data privacy constraints[26].

Broadly, this thesis contributes to the ongoing transformation in industrial maintenance towards more intelligent, data-driven methodologies enabled by generative AI. It provides clear evidence supporting the adoption of advanced AI models in predictive maintenance settings, underscoring the value of hybrid approaches that combine human-readable explanations with accurate, contextually grounded diagnostics[81].

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT 4o in order to proofread and enhance readability. After using this tool, the authors reviewed and edited the content and take full responsibility.

Bibliography

[1] Weiting Zhang, Dong Yang, and Hongchao Wang. "Data-driven methods for predictive maintenance of industrial equipment: A survey". In: *IEEE systems journal* 13.3 (2019), pp. 2213–2227.

[2] Gian Antonio Susto et al. "Machine learning for predictive maintenance: A multiple classifier approach". In: *IEEE transactions on industrial informatics* 11.3 (2014), pp. 812–820.

[3] Thyago P Carvalho et al. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (2019), p. 106024.

[4] IBM. *Predictive maintenance explained*. `https://www.ibm.com/topics/predictive-maintenance`. 2023.

[5] Rui Zhao et al. "Deep learning and its applications to machine health monitoring". In: *Mechanical Systems and Signal Processing* 115 (2019), pp. 213–237.

[6] S Sobhkhiz and T El-Diraby. "Integrating unstructured data analytics and bim to support predictive maintenance". In: *Life-cycle of structures and infrastructure systems*. CRC Press, 2023, pp. 1794–1801.

[7] Mubashar Raza et al. "Industrial applications of large language models". In: *Scientific Reports* 15.1 (2025), p. 13755.

[8] Yashashree Mahale, Shrikrishna Kolhar, and Anjali S More. "A comprehensive review on artificial intelligence driven predictive maintenance in vehicles: technologies, challenges and future research directions". In: *Discover Applied Sciences* 7.4 (2025), p. 243.

[9] Alicia Russell-Gilbert et al. "RAAD-LLM: Adaptive Anomaly Detection Using LLMs and RAG Integration". In: *arXiv preprint arXiv:2503.02800* (2025).

[10] Tita Alissa Bach et al. "Using LLM-Generated Draft Replies to Support Human Experts in Responding to Stakeholder Inquiries in Maritime Industry: A Real-World Case Study of Industrial AI". In: *arXiv preprint arXiv:2412.12732* (2024).

[11] Amazon Web Services. *What is Retrieval-Augmented Generation?* `https://aws.amazon.com/what-is/retrieval-augmented-generation/`. Accessed: 2025-04-30. 2023.

[12] Lei Liang et al. "Kag: Boosting llms in professional domains via knowledge augmented generation". In: *arXiv preprint arXiv:2409.13731* (2024).

[13] Beilei Zhu and Chandrasekar Vuppalapati. "Integrating Retrieval-Augmented Generation with Large Language Models for Supply Chain Strategy Opti-

mization". In: *World Congress in Computer Science, Computer Engineering & Applied Computing*. Springer. 2024, pp. 475–486.

[14] Riccardo Manzini et al. "Introduction to Maintenance in Production Systems". In: *Maintenance for Industrial Systems* (2010), pp. 65–85.

[15] Alexios Lekidis et al. "Predictive maintenance framework for fault detection in remote terminal units". In: *Forecasting* 6.2 (2024), pp. 239–265.

[16] Tianwen Zhu et al. "A survey of predictive maintenance: Systems, purposes and approaches". In: *arXiv preprint arXiv:1912.07383* (2019).

[17] You-Jin Park, Shu-Kai S Fan, and Chia-Yu Hsu. "A review on fault detection and process diagnostics in industrial processes". In: *Processes* 8.9 (2020), p. 1123.

[18] Wikipedia contributors. *Fault detection and isolation*. `https://en.wikipedia.org/wiki/Fault_detection_and_isolation`. Accessed: 2025-04-30. 2025.

[19] Juying Dai et al. "Signal-based intelligent hydraulic fault diagnosis methods: Review and prospects". In: *Chinese Journal of Mechanical Engineering* 32.1 (2019), p. 75.

[20] Khalid M Alsaif et al. "Multimodal Large Language Model-Based Fault Detection and Diagnosis in Context of Industry 4.0". In: *Electronics* 13.24 (2024), p. 4912.

[21] Prakash K Ray et al. "Detection of faults in a power system using wavelet transform and independent component analysis". In: *Computer, Communication and Electrical Technology*. CRC Press, 2017, pp. 227–231.

[22] Yaguo Lei et al. "Applications of machine learning to machine fault diagnosis: A review and roadmap". In: *Mechanical systems and signal processing* 138 (2020), p. 106587.

[23] Oscar Serradilla et al. "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects". In: *Applied Intelligence* 52.10 (2022), pp. 10934–10964.

[24] Andreas Theissler et al. "Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry". In: *Reliability engineering & system safety* 215 (2021), p. 107864.

[25] Hamzah AAM Qaid et al. "FD-LLM: Large Language Model for Fault Diagnosis of Machines". In: *arXiv preprint arXiv:2412.01218* (2024).

[26] Jiao Chen et al. "FaultGPT: Industrial Fault Diagnosis Question Answering System by Vision Language Models". In: *arXiv preprint arXiv:2502.15481* (2025).

[27] Apiradee Boonmee, Kritsada Wongsuwan, and Pimchanok Sukjai. "Consultation on Industrial Machine Faults with Large language Models". In: *arXiv preprint arXiv:2410.03223* (2024).

[28] Zichong Wang et al. "History, development, and principles of large language models: an introductory survey". In: *AI and Ethics* (2024), pp. 1–17.

[29] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[30] Tomas Mikolov et al. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 3. Makuhari. 2010, pp. 1045–1048.

[31] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[32] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[33] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[34] Yupeng Chang et al. “A survey on evaluation of large language models”. In: *ACM transactions on intelligent systems and technology* 15.3 (2024), pp. 1–45.

[35] Xun Liang et al. “Controllable text generation for large language models: A survey”. In: *arXiv preprint arXiv:2408.12599* (2024).

[36] Aske Plaat et al. “Reasoning with large language models, a survey”. In: *arXiv preprint arXiv:2407.11511* (2024).

[37] Juyong Jiang et al. “A survey on large language models for code generation”. In: *arXiv preprint arXiv:2406.00515* (2024).

[38] Chen Ling et al. “Domain specialization as the key to make large language models disruptive: A comprehensive survey”. In: *arXiv preprint arXiv:2305.18703* (2023).

[39] Google. *Introducing Gemini: our largest and most capable AI model*. `https://blog.google/technology/ai/google-gemini-ai/`. Accessed: 2025-04-29. 2023.

[40] Meta AI. *LLaMA 3 Model Card*. Available from Meta AI. 2024.

[41] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning in LLMs via RL*. `https://arxiv.org/abs/2501.12948`. 2025.

[42] Anthropic. *Claude 3 Technical Overview*. `https://www.anthropic.com/news/claude-3-family`. Anthropic Blog. 2024.

[43] Google. *Introducing PaLM 2*. `https://blog.google/technology/ai/google-palm-2-ai-large-language-model/`. Accessed: 2025-04-29. 2023.

[44] Mistral AI. *Introducing Mixtral 8x7B*. `https://mistral.ai/news/`. 2023.

[45] Alibaba Cloud. *Qwen Language Model Series*. `https://github.com/QwenLM`. 2023.

[46] TII. *Falcon LLM Series (7B, 40B, 180B)*. `https://huggingface.co/tiiuae`. 2023.

[47] Aixin Liu et al. “Deepseek-v3 technical report”. In: *arXiv preprint arXiv:2412.19437* (2024).

[48] Xiao Bi et al. “Deepseek llm: Scaling open-source language models with longtermism”. In: *arXiv preprint arXiv:2401.02954* (2024).

[49] Daya Guo et al. “DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence”. In: *arXiv preprint arXiv:2401.14196* (2024).

[50] Christopher Ibe. *Unlocking Low-Resource Language Understanding: Enhancing Translation with Llama 3 Fine-Tuning*. `https://medium.com/@ccibeekeoc42/unlocking-low-resource-language-understanding-enhancing-translation-with-llama-3-fine-tuning-df8f1d04d206`. Accessed: 2025-05-07. 2024.

[51] Wei Huang et al. “How good are low-bit quantized llama3 models? an empirical study”. In: *arXiv e-prints* (2024), arXiv–2404.

[52] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[53] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* 2 (2023), p. 1.

[54] Zhengbao Jiang et al. “Active retrieval augmented generation”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 7969–7992.

[55] Jiawei Chen et al. “Benchmarking large language models in retrieval-augmented generation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 16. 2024, pp. 17754–17762.

[56] Danqi Chen et al. “Reading wikipedia to answer open-domain questions”. In: *arXiv preprint arXiv:1704.00051* (2017).

[57] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. “Latent retrieval for weakly supervised open domain question answering”. In: *arXiv preprint arXiv:1906.00300* (2019).

[58] Kelvin Guu et al. “Retrieval augmented language model pre-training”. In: *International conference on machine learning*. PMLR. 2020, pp. 3929–3938.

[59] Nicholas Pipitone and Ghita Houir Alami. “Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain”. In: *arXiv preprint arXiv:2408.10343* (2024).

[60] Seongtae Hong et al. “Intelligent Predictive Maintenance RAG framework for Power Plants: Enhancing QA with StyleDFS and Domain Specific Instruction Tuning”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 2024, pp. 805–820.

[61] Conor Kelly. *8 Retrieval Augmented Generation (RAG) Architectures You Should Know in 2025*. `https://humanloop.com/blog/rag-architectures`. Accessed: 2025-05-07. 2025.

[62] Amazon Web Services. *What is LangChain?* Accessed: 2025-04-29. 2023. URL: `https://aws.amazon.com/what-is/langchain/`.

[63] LangChain Team. *Build a Retrieval Augmented Generation (RAG) App: Part 1*. `https://python.langchain.com/docs/tutorials/rag/`. Accessed: 2025-04-30. 2024.

[64] Igor Alekseev and Babu Srinivasan. *Retrieval-Augmented Generation with LangChain, Amazon SageMaker JumpStart, and MongoDB Atlas Semantic Search*. `https://aws.amazon.com/cn/blogs/machine-learning/retrieval-augmented-generation-with-langchain-amazon-sagemaker-jumpstart-and-mongodb-atlas-semantic-search/`. Accessed: 2025-05-14. 2023.

[65] LangChain. *Conceptual Guide | LangChain*. `https://python.langchain.com/v0.2/docs/concepts/`. Accessed: 2025-05-14. 2024.

[66] Tao Xu and Xue-Song Tang. “Electrical equipment fault diagnosis: A technique combining fuzzy logic and large language models”. In: *2023 IEEE International Symposium on Product Compliance Engineering-Asia (ISPCE-ASIA)*. IEEE. 2023, pp. 1–4.

[67] Qing Xia, Haotian Zhao, and Ming Liu. "Prompt Engineering Approach Study for Supervised Fine-Tuned (SFT) Large Language Models (LLMs) in Spacecraft Diagnosis". In: *2024 3rd Conference on Fully Actuated System Theory and Applications (FASTA)*. IEEE. 2024, pp. 819–824.

[68] Liane Makatura et al. "How can large language models help humans in design and manufacturing?" In: *arXiv preprint arXiv:2307.14377* (2023).

[69] Yiwei Li et al. "Large language models for manufacturing". In: *arXiv preprint arXiv:2410.21418* (2024).

[70] Akos Nagy, Yannis Spyridis, and Vasileios Argyriou. "Cross-Format Retrieval-Augmented Generation in XR with LLMs for Context-Aware Maintenance Assistance". In: *2025 IEEE International Conference on Artificial Intelligence and eXtended and Virtual Reality (AIxVR)*. IEEE. 2025, pp. 355–361.

[71] Fan Li et al. "Virtual Co-Pilot: Multimodal Large Language Model-enabled Quick-access Procedures for Single Pilot Operations". In: *2024 IEEE Conference on Artificial Intelligence (CAI)*. IEEE. 2024, pp. 1501–1506.

[72] WIRED. *AI Swaps Desk Work for the Factory Floor*. `https://www.wired.com/story/ai-swaps-desk-work-for-the-factory-floor/`. Accessed: 2025-05-14. 2025.

[73] Siyuan Chen et al. "Understanding stakeholder requirements for digital twins in manufacturing maintenance". In: *2023 Winter Simulation Conference (WSC)*. IEEE. 2023, pp. 2008–2019.

[74] Paulo Victor Lopes et al. "Data-driven smart maintenance decision analysis: A drone factory demonstrator combining digital twins and adapted ahp". In: *2023 Winter Simulation Conference (WSC)*. IEEE. 2023, pp. 1996–2007.

[75] Siyuan Chen et al. "Enhancing Digital Twins With Deep Reinforcement Learning: A Use Case in Maintenance Prioritization". In: *2024 Winter Simulation Conference (WSC)*. 2024, pp. 1611–1622. DOI: `10.1109/WSC63780.2024.10838867`.

[76] Haas Automation, Inc. *Haas Automation Sverige*. Tillgänglig: 26 maj 2025. 2025. URL: `https://www.haascnc.com/sv.html`.

[77] Jeff Johnson, Matthijs Douze, and Hervé Jégou. *FAISS: Facebook AI Similarity Search*. `https://faiss.ai/`. Accessed: 2025-05-24. 2024.

[78] Jiawei Han, Micheline Kamber, and Jian Pei. "2 - Getting to Know Your Data". In: *Data Mining: Concepts and Techniques (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82. ISBN: 978-0-12-381479-1. DOI: `https://doi.org/10.1016/B978-0-12-381479-1.00002-2`. URL: `https://www.sciencedirect.com/science/article/pii/B9780123814791000022`.

[79] OpenSPG. *KAG: Knowledge-Augmented Generation Framework*. `https://github.com/OpenSPG/KAG`. GitHub repository, ver. 0.7, accessed May 25, 2025. May 2025.

[80] Rajat soni. "Enhancing Transparency and Accountability in Predictive Maintenance with Explainable AI". In: *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT* 08 (Apr. 2024), pp. 1–5. DOI: `10.55041/IJSREM32027`.

[81] Muruganantham Angamuthu. “Smart Manufacturing: AI and Cloud Data Engineering for Predictive Maintenance”. In: *European Journal of Computer Science and Information Technology* 13 (Apr. 2025), pp. 100–119. DOI: 10.37745/ejcsit.2013/vol13n25100119.

A

Appendix

Appendix A: KAG System Configuration

This appendix provides the key configuration used in the KAG system. The configuration file, `kag_config.yaml`, includes model setup, retriever strategies, and reasoning modules as summarized below.

A.1 LLM and Vector Model Configuration

```
openie_llm:
  model: Llama3-80
  type: maas

chat_llm:
  model: deepseek-r1
  type: maas

vectorize_model:
  model: text-embedding-ada-002
  type: openai
  vector_dimensions: 1536
```

A.2 Project Parameters

```
project:
  biz_scene: default
  host_addr: http://127.0.0.1:8887
  namespace: Manual
```

A.3 KAG Builder Configuration

```
kag_builder_pipeline:
  chain:
    extractor:
      type: schema_constraint_extractor
      llm: *openie_llm
    reader:
      type: txt_reader
```

```
    post_processor:
      type: kag_post_processor
    splitter:
      split_length: 300
    vectorizer:
      type: batch_vectorizer
      vectorize_model: *vectorize_model
    writer:
      type: kg_writer
  num_threads_per_chain: 2
  num_chains: 4
  scanner:
    type: dir_file_scanner
```

A.4 KAG Solver Pipeline

```
kag_solver_pipeline:
  memory:
    type: default_memory
    llm_client: *chat_llm
  reasoner:
    lf_planner:
      type: default_lf_planner
      vectorize_model: *vectorize_model
    lf_executor:
      force_chunk_retriever: true
      exact_kg_retriever:
        type: default_exact_kg_retriever
        el_num: 5
      fuzzy_kg_retriever:
        type: default_fuzzy_kg_retriever
        el_num: 5
      chunk_retriever:
        recall_num: 10
  generator:
    type: default_generator
    llm_client: *chat_llm
  reflector:
    type: default_reflector
    llm_client: *chat_llm
```

A.5 Retriever Strategies Overview

The solver pipeline employs three types of retrievers:

- **Exact KG Retriever**: retrieves entities based on strict match rules using the knowledge graph.

- **Fuzzy KG Retriever**: performs semantic vector-based search over KG entries.
- **Chunk Retriever**: performs dense vector search over unstructured document chunks (e.g., logs, manuals).

This configuration supports hybrid reasoning with both symbolic (graph) and semantic (vector) information fusion.

Appendix B: Retrieval-Augmented Generation Based Troubleshooting QA Examples

This appendix provides a comparative analysis of troubleshooting answers generated using the Retrieval-Augmented Generation (RAG) approach with different retrieval sources.

Sources Retrieved

- **Event Logs:**
  - No relevant event logs found
- **Sensor Data:**
  - sensordata2060956.txt
  - sensordata10294.txt
- **Manual References:**
  - Turret Indexer Assembly - SL-10_30 ST-10_15 - Troubleshooting Guide.pdf (2 instances)
  - BMT65_75 - Turret Indexer Assembly - Troubleshooting Guide.pdf
  - Turret Indexer Assembly - ST-20_55 series - Troubleshooting Guide.pdf (2 instances)

1. Answer Using Only Manuals

To resolve the *"Turret Failure to Reach Unclamp Switch"* error, follow these steps based on Haas troubleshooting guides:

1. **Verify Proximity Switch Alignment**
   - Inspect the unclamp proximity switch: Ensure proper alignment (0.020–0.040" gap)
   - Clean the switch and target using compressed air
2. **Check Air/Hydraulic Pressure**
   - Hydraulic systems: 500–600 psi
   - Pneumatic systems: 85–100 psi
3. **Inspect for Mechanical Binding**
   - Clean turret mating surfaces
   - Lubricate gear teeth with Haas-approved grease
4. **Test Turret Indexing Manually**
   - Engage maintenance mode
   - Check for resistance in Geneva mechanism

Key References:

- *BMT65_75 - Turret Indexer Assembly - Troubleshooting Guide.pdf*
- *ST-20_55 series - Troubleshooting Guide.pdf*

2. Answer Using Only Event Logs

No relevant event log information available for this issue.

3. Answer Using Only Sensor Data

- **sensordata2060956.txt**
  - Vibration exceeded threshold (343h 29m)
  - Failure Type B: Verify turret mechanical movement
- **sensordata10294.txt**
  - Vibration exceeded threshold (171h 34m)
  - Failure Type A: Check interlock switch alignment

4. Answer Using Combined Sources

Synergized Diagnosis:

1. **Switch Alignment & Vibration Correlation**
   - Combine manual alignment specs with vibration data patterns
   - 0.002" alignment tolerance required (ST-20_55 series)
2. **Mechanical Interference Check**
   - Cross-reference sensor data timestamps with tool change logs
   - Inspect bearings for wear (Backlash >0.005" indicates replacement)
3. **Hydraulic-Pneumatic System Verification**
   - Validate pressure during vibration events
   - Check for pressure drops exceeding 10% of spec

Recommended Actions:

- Replace proximity switch (PN: 32-0874) if misalignment persists
- Schedule bearing replacement if vibration persists post-alignment
- Perform full turret recalibration after repairs

Appendix C: Knowledge-Augmented Generation Based Troubleshooting QA Examples

This appendix presents KAG-based troubleshooting responses to the same fault discussed in Appendix B, offering structured reasoning derived from knowledge graphs, manuals, and sensor data.

Question

Q: The message 'Turret Failure to Reach Unclamp Switch' appears. How should I diagnose and troubleshoot this issue?

Data Sources Retrieved

- **Event Logs:**
  - No relevant event logs found
- **Sensor Data:**
  - Found 3 relevant sensor records:
    - sensordata2060956.txt (score: 0.5862)
    - sensordata10294.txt (score: 0.5598)
    - sensordata10290.txt (score: 0.5578)
- **Manual References (KAG):**
  - KAG system provided manual-based answer
  - Found 10 manual references
    1. Turret Indexer Assembly - ST-20_55 series - Troubleshooting Guide
    2. Turret Indexer Assembly - SL-10_30 ST-10_15 - Troubleshooting Guide
    3. Turret Indexer Assembly - SL-10_30 ST-10_15 - Troubleshooting Guide

Generated Answers

A.0.0.0.1 1. Manual-Only Answer (KAG-based) Solution for "Turret Failure to Reach Unclamp Switch" Alarm (Alarm 113/2022/8113):

1. **Verify Air Supply:**
   - Check air pressure: Ensure 80–100 PSI
   - Inspect for leaks in air lines
2. **Inspect Turret Clamp/Unclamp Solenoid:**
   - Verify 24V DC at solenoid terminals
   - Test solenoid function manually
3. **Examine Clamp/Unclamp Cylinder:**
   - Inspect for leaks/bent rods
   - Verify full cylinder movement
4. **Check Turret Alignment:**
   - Clean turret assembly
   - Verify O-ring condition

- Align with Haas tools

Safety Reminders:

- Lock out/tag out before inspection
- Depressurize air lines

Outcome: Contact Haas Service if unresolved (Part #93-2609)

A.0.0.0.2 2. Event-Only Answer

No relevant event log information available.

A.0.0.0.3 3. Sensor-Only Answer

- **sensordata2060956.txt**
  - 343h 29m: Vibration exceeded (Failure Type B)
  - Solution: Verify turret mechanical movement
- **sensordata10294.txt**
  - 171h 34m: Vibration exceeded (Failure Type A)
  - Solution: Check interlock switch alignment
- **sensordata10290.txt**
  - 171h 30m: Vibration exceeded (Failure Type B)
  - Solution: Check interlock alignment

A.0.0.0.4 4. Combined Sources Answer Diagnostic Analysis & Troubleshooting Plan

1. **Cross-Reference Data:**
   - Vibration alerts correlate with mechanical binding
   - Manual guidance prioritizes electrical/pneumatic checks
2. **Prioritized Steps:**
   (a) Electrical System Check
      - Verify 24V DC at solenoid
      - Inspect wiring

   (b) Sensor Alignment
      - Clean and test proximity switch
      - Confirm 0.5–1.5mm gap

   (c) Pneumatic Validation
      - Check 80–100 PSI
      - Inspect FRL unit
3. **Mechanical Inspection:**
   - Check for chip intrusion
   - Inspect Curvic coupling teeth
   - Manual rotation test (5–10 Nm expected)

Corrective Measures:

- Replace Curvic coupling if >15% wear
- Install 5-micron particulate filter
- Upgrade to IP67-rated switches

Validation Test:

- Monitor current draw (8–12A)
- Track vibration spectrum (<3kHz)

Documentation:

- Record shim adjustments
- Document torque values (45–50 Nm)
- Final sensor gap measurements

Appendix D: Data Classification and Processing Code

This appendix presents the core Python scripts used for preprocessing and classifying manul books, preparing event log for vector databases, and extracting fault-resolution mappings.

Question

Q: The message 'Turret Failure to Reach Unclamp Switch' appears. How should I diagnose and troubleshoot this issue?

D.1 Fault Manual Classification Script (`classify.ipynb`)

This script parses Haas machine manuals and uses a language model to extract part names and general solution patterns. It then classifies each part into a high-level category using zero-shot reasoning.

```
import os, pandas as pd
from openai import OpenAI
from dotenv import load_dotenv
load_dotenv()

client = OpenAI(api_key="", base_url="")

folder_path = "../builder/txtfile"
file_list = [f for f in os.listdir(folder_path)]

title, book_content = [], []
for f in file_list:
    title.append(f.split(".txt")[0])
    with open(os.path.join(folder_path,f),"r",encoding="utf-8") as f:
        content = f.read()
    book_content.append(content)

df = pd.DataFrame([file_list, title, book_content]).T
df.columns = ['file_name',"title","content"]

def analysis_mamual_book(title,content):
    response = client.chat.completions.create(
        model="deepseek-reasoner",
        messages=[{"role":"system","content": "...LLM prompt omitted..."},
                  {"role":"user","content":f"Manual title: {title}, content: {
    content}"}],
        temperature=0.01
    )
    return response.choices[0].message.content

# Run model and extract part names + solutions
result = []
for ind,row in df.iterrows():
    answer = analysis_mamual_book(row['title'],row['content'])
```

```
    result.append(answer)

df['part_name'] = [eval(i.split("```json")[1].split("```")[0])['part_name'] for i
     in result]
df['solutions'] = [eval(i.split("```json")[1].split("```")[0])['solutions'] for i
     in result]

# Classify into 6 categories using LLM
response = client.chat.completions.create(
    model="deepseek-reasoner",
    messages=[{"role":"system","content":"...classification prompt..."},
              {"role":"user","content":f"{list(set(df['part_name']))}"}],
    temperature=0.01
)

df['category'] = df['part_name'].map(eval(response.choices[0].message.content.
    strip()
                                          .split("```json")[1].split("```")[0]))

df.to_excel("../builder/classified_ds.xlsx", index=False)
```

Listing A.1: Extract and Classify Parts from Manuals

D.2 Sensor Data Cleaning and Fault Type Tagging (`clean_up_data.ipynb`)

This script maps the previously classified parts to two failure types and prepares them for vectorization by writing cleaned text files and a solution sample set.

```
import pandas as pd, random, json

df = pd.read_excel("../builder/classified.xlsx")

failure_type_1 = ["Spindle & Drive", "Mechanical Components", "Automation &
    Robotics"]
failure_type_2 = ["Coolant & Lubrication", "Electrical & Control", "Hydraulic &
    Pneumatic"]

df.loc[df['category'].isin(failure_type_1), 'failure_type'] = "1"
df.loc[df['category'].isin(failure_type_2), 'failure_type'] = "2"

for ind, row in df.iterrows():
    with open(f"../builder/classfired_failure_type_txt/type_{row['failure_type
    ']}/{row['file_name']}",
              "w", encoding="utf-8") as f:
        f.write(row['content'])

# Sample representative solutions for each failure type
failure_type_1_solution = []
for i in df[df['failure_type'] == "1"]['solutions']:
    failure_type_1_solution.extend(eval(i))

failure_type_2_solution = []
for i in df[df['failure_type'] == "2"]['solutions']:
    failure_type_2_solution.extend(eval(i))
```

```
random.seed(42)
solution_mapping = {
    "failure_type_1_solution_sample": random.sample(failure_type_1_solution, 6),
    "failure_type_2_solution_sample": random.sample(failure_type_2_solution, 6)
}

with open('../builder/solution_mapping.json', 'w', encoding='utf-8') as f:
    json.dump(solution_mapping, f, ensure_ascii=False, indent=4)

df.to_excel("../builder/classified_type.xlsx", index=False)
```

Listing A.2: Sensor Data Labeling and Sampling

D.3 PDF Manual Parsing Script (`pdf_reader.py`)

This script batch-processes Haas manuals in PDF format, converts them to Markdown, and stores both textual and image-based outputs for downstream processing.

```
import pymupdf4llm, os, json

def process_pdf_documents():
    root_path = "../data/pdf"
    files = os.listdir(root_path)

    for f in files:
        pdf_document = os.path.join(root_path, f)
        photo_folder = os.path.join("../data/photo", f)
        os.makedirs(photo_folder, exist_ok=True)

        md_text = pymupdf4llm.to_markdown(
            pdf_document,
            page_chunks=True,
            write_images=True,
            image_path=photo_folder,
            margins=(0, 30, 0, 0),
            extract_words=True
        )

        page_content = {id: page['text'] for id, page in enumerate(md_text)}
        with open(f"ingested_content/{f}.json", 'w', encoding='utf-8') as
    json_file:
            json.dump(page_content, json_file, ensure_ascii=False, indent=4)

if __name__ == "__main__":
    process_pdf_documents()
```

Listing A.3: Convert PDF to Markdown and Extract Images

D.4 Operator-Style Solution Generation via LLM

This script demonstrates how to automatically generate natural-language troubleshooting feedback in an operator's voice. The pipeline consists of three stages: data loading, solution matching, and LLM-based style optimization.

D.4.1 Environment Setup

```
import pandas as pd
from openai import OpenAI
from dotenv import load_dotenv
load_dotenv()

client = OpenAI(
    api_key="your-api-key-here",
    base_url="https://api.chatanywhere.tech/v1"
)

\begin{lstlisting}[language=Python, caption={Read Failure and Template Files}]
failures = pd.read_csv(r"../data/csv/failures_updated.csv")
df = pd.read_excel("../builder/classified.xlsx")

\begin{lstlisting}[language=Python, caption={Map Failure Record to Predefined
    Solution}]
category_list = list(set(df['category']))
solution = []

for ind, row in failures.iterrows():
    category = category_list[int(row['solution_method']) - 1]
    tmp = df[df['category'] == category]
    tmp.index = range(len(tmp))

    solution.append(
        eval(tmp.loc[0, 'solutions'])[int(row['failure_type'])]
    )

failures['solution'] = solution

\begin{lstlisting}[language=Python, caption={Define Prompt for Operator-Style
    Output}]
def generate_operator(content, solution):
    response = client.chat.completions.create(
        model="deepseek-reasoner",
        messages=[
            {"role": "system", "content": """
You will receive a machine runtime record and a target action. Your task is to
    rewrite them in the voice of a machine operator.

#Output:
{"operator": "Rewritten operator-style instruction"}

Reply only in English.
            """},
            {"role": "user", "content": f"Runtime: {content}\nAction: {solution}"
    }
        ],
        temperature=0.01
    )
    return response.choices[0].message.content

\begin{lstlisting}[language=Python, caption={Loop Through All Failures and
    Generate Output}]
```

```
results = []

for i, row in failures.iterrows():
    print(i)
    content = ""
    for c in failures.columns[:-1]:  # Skip 'solution' column
        content += f"{c}:{row[c]}"

    res = generate_operator(content, row['solution'])
    results.append(res)

failures['operator_solution'] = [eval(i)['operator'] for i in results]
failures.to_excel("../builder/repair_feedback.xlsx")
```

Listing A.4: Environment and Client Initialization

Standardized Fault Diagnosis Questions

1. The message 'Turret Failure to Reach Unclamp Switch' appears. How should I diagnose and troubleshoot this issue?
2. When the coolant pump is operating but the outlet pressure is insufficient, what are the possible causes? How can I locate the issue?
3. How can I determine if the Haas spindle drive (Vector Drive) has a short circuit or abnormal Regen resistance?
4. The Haas live tooling unit overheats and vibrates excessively during operation. How should this be addressed?
5. Why does the tailstock fail to respond during automatic movement or clamping? How can I determine if this is a hydraulic system issue?
6. The coolant replenishment system is adding coolant at an incorrect ratio or failing to replenish. How should I recalibrate the system?
7. The spindle emits abnormal noise or surface finish quality deteriorates during rotation. Which component may be faulty?
8. During tool change, the tool is stuck. How should the alignment of the Haas side-mounted tool magazine double arms (SMTC Double Arm) be checked?
9. After the APL light curtain is triggered, the machine does not stop or triggers a false alarm. What parameter settings or sensor issues could be causing this?
10. When the servo motor does not respond or the axis does not move, how should it be determined whether the issue is with the power supply or the control board?
11. When alarm 113 or 114 indicates that the tool turret cannot complete the clamping/unlocking action, how should you determine if the solenoid valve is functioning properly?
12. If the gripper of the automatic loading/unloading machine (APL) cannot operate stably or becomes loose, how should you inspect the pneumatic system?