POLITECNICO DI TORINO

Collegio di Ingegneria Gestionale e della Produzione
Corso di Laurea Magistrale in Management Engineering

Retrieval-Augmented Generation (RAG) System for Manufacturing

Supervisor

Prof.ssa Giulia Bruno

Candidate

Javohir Bakhronov

A.A 2025/2026

Abstract

The use of Large Language Models (LLMs) in industrial settings has opened new possibilities for intelligent support within manufacturing systems. Nonetheless, standalone LLMs often struggle to reliably access specialized technical documentation needed in production environments. To address this, this thesis applies the development of a Retrieval-Augmented Generation (RAG) system focused to manufacturing that facilitates structured access to technical documentation and production-related information.

Improving over existing RAG systems, this research presents an enhanced version that incorporates enriched metadata to improve document grounding and retrieval accuracy. A dedicated evaluation metrics program was also designed to measure and analyze retriever performance, allowing for systematic comparisons of different system versions. The updated RAG system was deployed at Mind4LAB, Politecnico di Torino, and integrated into a collaborative manufacturing environment with a Yaskawa collaborative robot.

Three RAG system versions are Baseline, Version 1, and Version 2 that were tested across defined industrial use cases. The obtained results enabled structured comparison of system behavior and retrieval performance across versions. Based on these findings, a conceptual analysis was conducted to examine the impact of metadata enrichment and system improvements on RAG performance in manufacturing applications.

Index

List of Figures

List of Tables

Chapter I: Introduction

1.1 Context of the work

Manufacturing is entering a new phase where success increasingly relies on how quickly a team can transform technical knowledge into clear, practical actions. Today's production lines are filled with scattered information which are robot manuals, maintenance procedures, quality-control guidelines, and internal engineering notes. When an operator or engineer encounters an alarm, starts a recovery process, or needs to adjust parameters, the real challenge isn't missing data. It's the difficulty of quickly finding the right information and turning it into simple, step-by-step instructions.

To begin with, Large Language Models (LLMs) seem like a natural way to access knowledge: they understand questions and produce fluent answers. But fluency doesn't always equal reliability. In industrial settings, answers need to be based on authoritative documents, traceable, and aligned with procedures. LLMs alone can't guarantee this because they generate responses from internal representations learned during training, which might be incomplete, outdated, or not match a plant's documentation and constraints. This challenge is well known in knowledge-heavy NLP: sometimes, the model offers a confident answer even when the supporting evidence isn't in its training data.

Retrieval-Augmented Generation (RAG) shows a promising solution: it links generation to external evidence by first retrieving relevant passages and then producing an answer based on those sources. This approach mixes the disciplines of information retrieval, indexing, ranking, and relevance evaluation with the power of LLMs for synthesis and instruction. In open-domain contexts, this retrieval-plus-generation method has shown strong improvements in factual accuracy and performance on knowledge-intensive tasks. The effectiveness of modern retrieval techniques, including dense passage retrieval, highlights the importance of the retrieval component in overall system performance. Essentially, RAG shifts focus from "what the model remembers" to "what the system can reliably find and justify."

This shift is highly relevant to industry. Manufacturing knowledge is often domain-specific, procedural, and safety-critical. A RAG-based assistant can serve as a guided interface over a curated collection of manuals and internal documents, retrieving pertinent information and

generating responses rooted in those sources. This can help troubleshoot faster and support more consistent decision-making. Recent research has shown potential of RAG-like systems for manufacturing quality control, demonstrating that grounding industrial help in technical content is feasible. Still, two main gaps remain: first, many RAG systems are tested mainly on general benchmarks rather than in real industrial settings; second, evaluation is often an afterthought rather than a core part of system development. Experts emphasize that robust RAG evaluation needs systematic metrics and clear insights into both how retrieval and generation perform.

This work focuses precisely on that point: designing a manufacturing-focused RAG system, building an integrated evaluation framework, and testing it in real-world industrial applications. Specifically, it's applied in a collaborative robotics environment at Mind4LAB, Politecnico di Torino, working with a Yaskawa collaborative robot.

1.2 Research questions

The following research questions guide this thesis:

RQ1: How can a RAG system be effectively implemented in a manufacturing environment, and why is such integration necessary?

RQ2: How can improvements to the current RAG architecture enhance query response quality and retrieval performance?

RQ3: How can a dedicated evaluation program be designed to estimate RAG system-level metrics and systematically assess retriever metrics?

RQ4: How do different RAG system versions (Baseline, Version 1, and Version 2) perform when implemented in real industrial use cases at Mind4LAB with a Yaskawa collaborative robot?

RQ5: What conceptual insights can be derived from comparing the three RAG versions in terms of architectural evolution, system behavior, and industrial applicability?

1.3 Aim of the thesis and technological objectives

This thesis aims to improve, deploy, and systematically evaluate an industrial-oriented RAG system that provides evidence-based knowledge assistance for manufacturing workflows and collaborative robotics.

Concretely, the work pursues four technological objectives:

1. System upgrade for obtaining better query answering

Improve the existing RAG system by metadata-enriched organization of industrial documents and targeted pipeline refinements, with the goal of increasing the relevance and grounding of answers to manufacturing queries.

2. Integrated evaluation program for RAG system assessment
   Design and implement an evaluation program integrated into the RAG system to compute system-level metrics and expose retriever behavior in a repeatable, version-comparable way. This objective is motivated by the broader need for systematic, introspective evaluation of RAG pipelines.

3. Real industrial deployment in a collaborative robotics setting
   Implement and deploy three system versions (Baseline, Version 1, Version 2) in real industrial use cases at Mind4LAB (Politecnico di Torino), integrating them into a collaborative manufacturing environment involving a Yaskawa collaborative robot.

4. Cross-version comparison and conceptual analysis
   Execute consistent use cases across the three versions, compare observed results, and produce a conceptual analysis that explains how architectural changes (including metadata integration and evaluation metrics) influence retrieval behavior and end-to-end system performance.

1.4 Scope and industrial validation setting

This thesis focuses on providing manufacturing-oriented, document-based support, particularly for reliably retrieving procedural and technical knowledge. Industrial validation takes place at Mind4LAB, Politecnico di Torino, in a collaborative manufacturing environment using a Yaskawa collaborative robot. This setup is designed to evaluate whether the system can function effectively under real-world conditions which facing domain-specific terminology, detailed procedures, and time-sensitive operator requirements, where accuracy and proper grounding are essential.

1.5 Structure of the document

The structure of the thesis is:

In Chapter 2, Theoretical Background explains the fundamentals of Information Retrieval and modern retrieval approaches relevant to RAG, as well as the principles of retrieval-conditioned generation.

In Chapter 3, State of the Art examines RAG architectures, evaluation methods, and industrial applications, with a focus on systematic assessment and diagnostic frameworks.
In Chapter 4, System Design and Implementation describes the development from Baseline to Version 1 and the upgraded Version 2, including metadata integration choices.
In Chapter 5, Evaluation Program and Methodology details the integrated evaluation process, the system-level metrics calculated, and the experimental protocol for cross-version comparison and explanation of the created database
In Chapter 6, Industrial Deployment at Mind4LAB covers integration with the collaborative setup and the Yaskawa collaborative robot, including the tested use cases.
In Chapter 7, Results and Conceptual Analysis presents the results and offers a conceptual review of system evolution across versions.
In Chapter 8, Conclusions and Future Work summarizes contributions and suggests future directions for industrial-grade RAG systems.

Chapter II: Theoretical background

2.1 Manufacturing knowledge

Manufacturing is undergoing a fundamental transformation driven by global competition, rapid technological advancements, and increasing uncertainties in supply chains. These challenges motivate production systems to become more adaptable, operate in shorter cycles, meet stricter quality standards, and build greater resilience. Industry 4.0 plays a crucial role in this change by helping factories integrate cyber-physical systems, enhance connectivity, and rely more on data-driven processes. Evidence shows that adoption varies widely and is heavily influenced by existing infrastructure, skills, and governance, especially for small and medium-sized enterprises, where digitization tends to happen gradually. Meanwhile, the focus is expanding beyond efficiency alone; Industry 5.0 emphasizes sustainability, resilience, and placing humans at the core. It highlights technologies that support operators and strengthen the social and technical aspects of manufacturing. Research on factory design now considers these priorities, requiring new approaches to design and modeling. Overall, while Industry 4.0 and automation are progressing, their adoption differs across sectors and maturity levels, resulting in a landscape where connected production and intelligent automation are advancing but have not yet been uniformly implemented across the manufacturing.

At the market level, the push for AI adoption is driven by high expectations for growth. A detailed analysis by Statista shows that the global AI market is expected to exceed $1 trillion by 2031, expanding steadily at 26,6% each year. This trend suggests that there's serious, ongoing investment rather than just a passing fad. Additionally, the same report reveals that by 2024, 72% of businesses worldwide are already integrating AI into their operations, demonstrating that AI is evolving from isolated tests to a fundamental part of how organizations plan and compete.

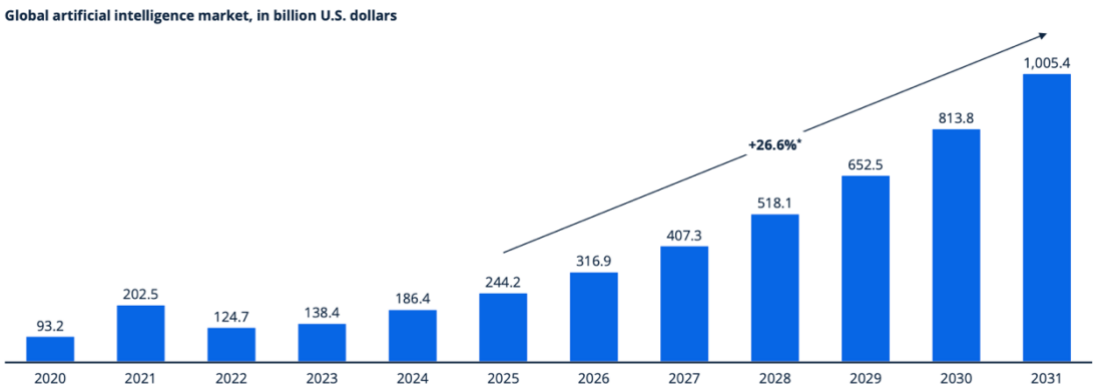

Figure 1:The global AI market is projected to exceed one trillion U.S. dollars by 2031[28]

However, research on Industrial AI shows that adopting the technology doesn't immediately boost productivity. Companies often experience an initial “adjustment phase,” where costs for data engineering, workflow redesign, governance, and training rise before the benefits become clear. This pattern resembles a productivity J-curve. The main point is that AI delivers true value when it is fully integrated into production and decision-making processes, rather than simply being installed.' This view aligns with Industry 4.0 research, which considers advanced digital technologies as capabilities that build on complementary assets like infrastructure, skills, and management practices, resulting in varying adoption rates across countries and firms.

From a manufacturing standpoint, the adoption of digital technologies is often limited by how ready companies are to digitize, especially small and medium-sized enterprises. These businesses usually advance gradually rather than jumping straight into full automation. This step-by-step approach impacts AI adoption because successful implementation depends on having sufficient data, seamless integration, and stable digital workflows. Ultimately, Industry 5.0's broader policy perspective highlights that scaling AI should prioritize human-centricity, resilience, and sustainability. This means that adoption is now judged not only by efficiency gains but also by how well it supports workers and ensures robust operations.

AI adoption within companies is accelerating, but it varies greatly in depth and maturity. Many firms report using AI, yet only some have integrated it into their main workflows where it can provide lasting performance improvements. Evidence shows that while many businesses are adopting AI and increasing their investments, this widespread interest does not mean they all have the same level of capability or readiness. Studies from industrial sectors explain why companies often face an early “integration penalty”: costs for data engineering, system integration, governance, and workflow redesign rise before productivity

benefits begin to show, following a J-curve pattern. At the same time, the innovation scene highlights “Applied AI" as one of the top innovation trends. “Generative AI” also draws strong interest, indicating that companies are investing in AI tools that directly impact production and exploring new generative capabilities, though at different speeds depending on their data resources and organizational strength.

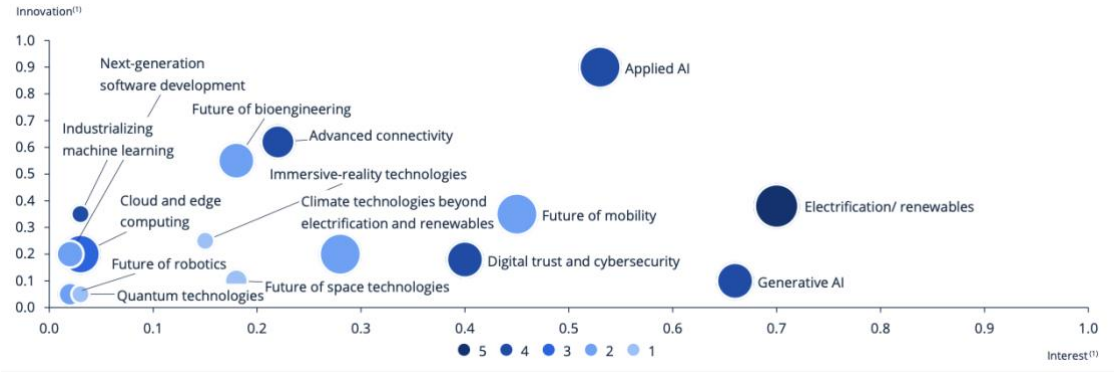

Figure 2:The global AI market is projected to exceed one trillion U.S. dollars by 2031[28]

Industry 5.0 reframes industrial transformation around human-centricity, resilience, and sustainability, shifting the emphasis from automation-only efficiency to socio-technical systems in which advanced technologies actively support workers and make production networks more resilient to disruption. [23] This evolution requires not only new machines but also new methods for delivering trusted operational knowledge at the point of work, especially in environments with collaborative robots, frequent changeovers, and safety-critical procedures. [22], [23] In this context, a RAG system is particularly well aligned as an enabling technology: it couples retrieval from authoritative plant and vendor documentation with LLM-based synthesis, producing answers that are grounded, traceable, and context-aware rather than purely fluent text. [1] Manufacturing-oriented RAG studies demonstrate that this approach can support diagnosis and corrective actions by transforming large technical corpora into actionable guidance, thereby reducing manual search and interpretation effort. [7], [13] Moreover, because much industrial know-how is fragmented across unstructured communications and siloed documents, RAG-style knowledge pipelines can help convert dispersed organizational knowledge into reusable decision support, directly reinforcing the industry 5.0 goal of augmenting human operators with reliable, accessible intelligence. [19], [23]

2.2 Retriever – Augmented (RAG) systems

The roots of Retrieval-Augmented Generation lie in a long-standing idea in information science: when a question requires specific knowledge, the system should retrieve evidence before generating an answer. Classical Information Retrieval established the formal machinery for doing this at scale indexing, ranking, relevance, and evaluation, so that a user query could be matched to the most useful documents rather than relying on memorized or manually curated responses. This IR foundation is essential because it defines the core problem that RAG subsequently solves selecting the appropriate evidence from a large corpus under time and noise constraints. [3], [15]

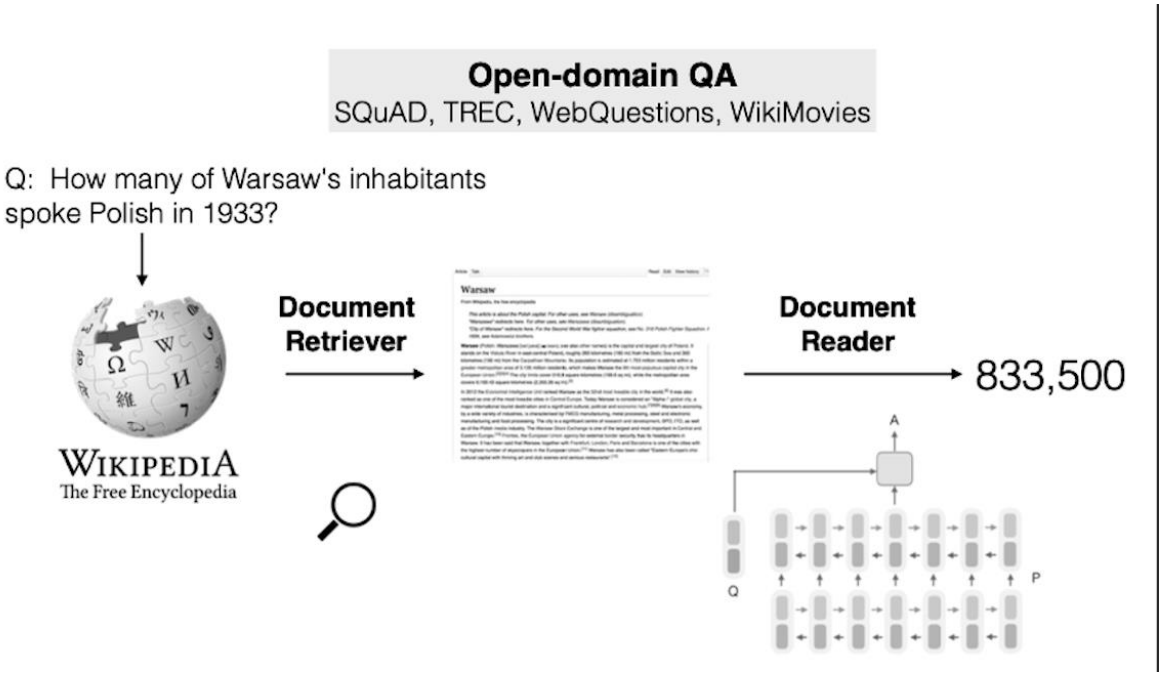

Figure 3: An overview of the question answering system, DrQA.

The modern motivation for RAG emerged when neural language models became strong generators but still lacked controlled access to up-to-date or domain-specific knowledge. Early open-domain QA systems showed that retrieval-based answering outperformed methods relying solely on large corpora such as Wikipedia, in which the system must first find relevant passages before producing an answer grounded in them. Work such as Reading Wikipedia to Answer Open-Domain Questions helped formalize this retrieval-first approach in neural QA systems, highlighting retrieval as a key skill for tackling knowledge-rich questions. Meanwhile, datasets like SQuAD sped up progress in machine reading and answer extraction, reinforcing the idea that high-quality QA depends on accessing the right context, not just generation.

A decisive step toward today's RAG systems was the development of strong neural retrievers that retrieve relevant passages using dense embeddings rather than relying solely on keyword overlap. Dense Passage Retrieval (DPR) introduced an effective dual-encoder framework for open-domain QA, enabling semantic retrieval at scale and making retrieval a practical front end for neural generation and reading models. [9] This retrieval capability created the missing system component needed to reliably connect a query to the right evidence.

The term and architecture "Retrieval-Augmented Generation" were then explicitly formulated by Lewis et al., who proposed RAG as a unified framework in which a generator conditions its output on documents retrieved from an external corpus at inference time. Their work showed that retrieval can serve as an external memory, improving factual accuracy and coverage in knowledge-intensive NLP tasks by allowing the system to "look up" relevant information rather than relying solely on model parameters. [1] This paper is widely regarded as the canonical origin of RAG as a named system architecture, defining the retriever–generator coupling that later industrial and academic systems have built upon. [1]

2.2.1 RAG system paradigms

Retrieval-Augmented Generation (RAG) is now best understood as a system design space rather than a single architecture. The unifying principle is simple: generation should be conditioned on retrieved evidence from an external corpus, treating retrieval as an explicit, updatable memory at inference time rather than relying solely on parameters learned during pretraining. [1], [29] What distinguishes modern RAG systems is how they operationalize this principle, how they index knowledge, select evidence, suppress noise, and constrain the generator. This has led the literature to converge on a small set of paradigms that correspond to increasingly strong engineering control: Naive RAG, Advanced RAG, Modular/Agentic RAG, and an emerging class of Structured (Graph-based) RAG systems. [29], [4], [30]

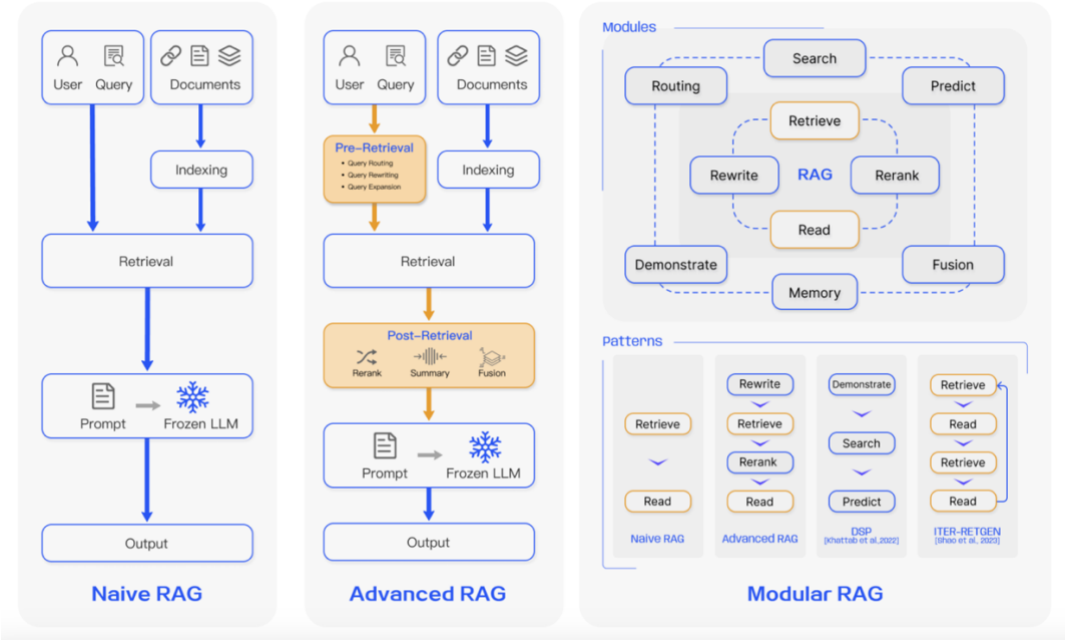

Figure 4: RAG models

2.2.2 Naive RAG: the minimal retrieve-then-generate pipeline

Naive RAG represents the simplest version of the canonical pipeline: documents are divided into chunks and indexed; a retriever selects the top-k most relevant chunks for a query; then a language model creates an answer based on those chunks as context. This approach is useful because it provides a basic roundedness benchmark with minimal complexity. However, it also reveals a fundamental limitation of RAG: the reliability of the generated answer depends entirely on the quality of the evidence provided. If retrieval results in noisy, redundant, or only weakly relevant chunks, the model may produce plausible but misleading answers or average out conflicting evidence. Studies show that naive pipelines can be fragile in ambiguous situations, multi-step questions, and in corpora filled with repetitive boilerplate or near-duplicate procedures, which are common in industrial documents. For this reason, industrial guidance treats naive RAG as a starting point for testing feasibility rather than an endpoint ready for deployment.

2.2.3 Advanced RAG: engineering evidence quality before generation

Advanced RAG maintains its core structure but introduces optimization layers to enhance the quality of evidence provided to the LLM. The guiding principle is control: minimizing the chances of retrieving irrelevant context and maximizing the likelihood that the most important passage appears clearly in the final prompt. Common approaches include rewriting and expanding queries to improve recall, combining sparse lexical matching with

dense semantic retrieval to capture both keywords and intent, reranking with more powerful cross-encoders to correct approximate neighbor errors, and filtering or compressing context to reduce distractions and redundancy. These techniques are not superficial; they directly address the main operational challenge in real RAG systems: retrieval noise that can interfere with generation. Industry perspectives on RAG highlight these layers because real-world data like PDF manuals, procedural instructions, and technical standards tends to diminish the effectiveness of simple top-k retrieval methods.

2.2.4 Modular and agentic RAG: componentized pipelines for heterogeneous industrial knowledge

As RAG systems are integrated into diverse enterprise datasets, a simple, linear pipeline no longer suffices. Modular RAG approaches treat retrieval and generation as flexible, interchangeable components guided by routing and control logic. Instead of relying on a single retriever and index, the system may manage multiple storage types such as vector indexes, keyword indexes, structured databases, and conversational memories, employing different retrieval strategies based on the query and combining or removing duplicate evidence before generating a response. In more dynamic scenarios, the system iterates: it retrieves information, drafts an answer, checks confidence or coverage, and retrieves again if necessary. This approach reflects the real-world diversity of industrial questions. Some need exact matches of identifiers or error codes, others require conceptual retrieval, and some demand multi-document synthesis. Industry best practices explicitly favor modular designs because they enhance controllability, support governance, and allow targeted updates without risking the stability of the entire pipeline.

2.2.5 Structured and graph-based RAG: retrieval over relations and evidence chains

A newer approach expands RAG beyond simply ranking independent chunks by similarity. It leverages the inherent structure within information, using graph-based methods to model entities, relationships, fault codes, procedures, and dependencies. This allows retrieval not just of isolated passages but of connected evidence chains. “GraphRAG” exemplifies this shift, aiming to enhance retrieval and reasoning when relevant data is spread across multiple documents and linked through relationships. This approach is particularly useful in industrial settings, where technical documents naturally contain relational links machines comprising

subsystems, procedures referencing prerequisites, alarms associated with corrective actions, and maintenance steps dependent on configuration constraints. By incorporating this structural information, graph-augmented retrieval offers a more principled way to access and connect relevant data, reducing reliance on local similarity and supporting more accurate multi-hop evidence aggregation.

2.2.6 Relevance of paradigms to industrial deployment

The significance of these paradigms becomes much clearer when applied in industrial environments. Manufacturing RAG systems for quality control show how RAG can enhance diagnosis and corrective actions by anchoring responses in domain-specific knowledge instead of relying on generic generation. Similarly, automotive industrial RAG systems highlight the need for robust document ingestion and retrieval strategies when dealing with large collections of technical PDFs, which are often hampered by formatting issues and procedural redundancy. In this setting, industrial guidance points out that achieving deployment-grade reliability usually means moving beyond simple RAG approaches to more advanced, modular paradigms, with structured retrieval emerging as an especially promising solution for navigating complex technical knowledge spaces.

2.2.7 RAG Evaluation Methodologies and Diagnostic Frameworks

RAG evaluation has become a key area of research and development because retrieval-augmented systems tend to fail in ways that are qualitatively different from standalone language models. A RAG pipeline is a complex system, comprising indexing, retrieval, context assembly, and generation whose reliability depends not just on producing answers that sound correct, but also on whether the system correctly retrieved relevant evidence, used it appropriately, and remained faithful to it. Recent surveys highlight that evaluation needs to be broken down: retrieval quality should be assessed separately from generation quality, and generation should be tested for proper grounding, effective use of context, and resistance to hallucinations when evidence is weak or incomplete. This approach encourages the development of methodologies that examine how variations in chunking, retrieval strategies, reranking, or prompting influence behavior, making comparisons between different system versions more scientifically valid rather than anecdotal. Beyond simple scoring, the field is moving toward diagnostic frameworks that interpret failures at the individual response level. Platforms like “INSPCTORRAGET” facilitate this by linking each response to its retrieved context and allowing systematic analysis of failure modes, transforming evaluation into a

process of troubleshooting and improvement. In industrial environments, where documentation varies and ensuring procedural correctness is vital, this evolution is essential: deployment-ready RAG systems require ongoing, structured assessment to detect regressions, understand weaknesses, and justify trust in their outputs under real-world conditions.

Frameworks	Version	License	Integration: Compatible Frameworks	Integration: Models	Metrics: Questions-Contacts	Metrics: Questions-Responses (given questions)	Metrics: Contexts-Responses (given questions)	Metrics: Contexts-Ground Truth (given questions and/or responses)	Metrics: Responses-Ground Truth (given questions)	Metrics: Responses (given questions and/or contacts)
Giskard	2.10.0	Apache-2.0	GitHub, MLflow, Weights & Biases, Baseten, HuggingFace, PyTorch models, AITD, Pytest	Catboost, Hugging Face models, Langchain models, PyTorch models, Sklearn models, Tensorflow models	ragas_answer_relevancy	x	ragas_faithfulness	ragas_context_precision ragas_context_recall	Correctness (CoT) LLM-as-a-judge Ground Truth Similarity	x
promptfoo	0.50.1	MIT license	GitHub Actions, GitLab CI, Provider: models, Mischa:Chat	OpenAI, Anthropic, Azure OpenAI models, Llamacpp, Ollama, Google Vertex models, Google AI Studio, Generic webhook, Custom API Provider, Custom scripts, HuggingFace models, Cohere, Replicate, Groq, Mistral AI models, OpenLLM, OpenRouter, Perplexity, text-	[Model-graded metrics] context-relevance	[Model-graded metrics] lm-rubric, answer relevance, Similarity	[Model-graded metrics] context-faithfulness	[Model-graded metrics] context-recall	[Deterministic metrics] contains, equals, LLM/string distance, perplexity, regex, rouge-n, starts-with [Model-graded metrics] lm-rubric, model-graded-closedqa, factuality,	Model-graded metrics: lm-rubric, model-graded-closedqa, classifier grading, Sentiment, Tone and emotion, Helpfulness, Grounding, factuality, evidence-type
Ragas	0.1.6	Apache-2.0	Llamaindex, Langchain, LangFuse, Arize-Phoenix, LangFuse, Athina, Zeno, TonicValidate, Haystack	OpenAI models, Azure OpenAI models, Amazon Bedrock models, Google Vertex AI models	Context Relevancy	Answer Relevance	faithfulness	Context Precision Context Recall Context entities recall	Answer semantic similarity Answer correctness	Aspect Critique (harmfulness, maliciousness)
DeepEval	0.21.15	Apache-2.0	Llamaindex, Hugging Face	Llamaindex, Hugging Face models	ContextualRelevancyMetric	SummarizationMetric AnswerRelevancyMetric (RAGAS)	Faithfulness (RAGAS)	ContextualRecallMetric (RAGAS) ContextualPrecisionMetric (RAGAS)	GEval	HallucinationMetric BiasMetric ToxicityMetric KnowledgeRetentionMetric
agenta	0.12.6	MIT license	Langchain, Llamaindex, and "any others"	OpenAI models, Cohere, or local models, and "any others"	x	x	x	x	Exact match Regex match Webhook evaluator (Correctness) Similarity match (Jaccard) AI Critic (LLM-based)	x
Trulens	0.27.0	MIT license	Langchain, Llamaindex, NeMo, Guardrails	OpenAI models, AzureOpenAI models, Amazon Bedrock models, LiteLLM models, Langchain models	[Generation-based Stock Feedback Functions] content_relevance, qs_relevance	[Generation-based Stock Feedback Functions] relevance	[Generation-based Stock Feedback Functions] Groundedness	x	[Stock Feedback Functions] HF language_match [Generation-based Stock Feedback Functions] comprehensiveness	[Stock Feedback Functions] HF toxic, OpenAI moderation, harassment [Generation-based Stock Feedback Functions]
Tonic Validate	4.0.4	MIT license	Llamaindex	OpenAI models, AzureOpenAI models	Retrieval precision	x	Augmentation precision Augmentation accuracy Answer consistency (binary) Retrieval recall	x	Answer Similarity	Contains Text
LangChain	0.1.13	MIT license	Various	Various	String Evaluator Scoring Evaluator	String Evaluator Scoring Evaluator	String Evaluator Scoring Evaluator	String Evaluator Scoring Evaluator	String Evaluator -Criteria Evaluation (correctness) -Embedding Distance -Exact Match, Regex Match -Scoring Evaluator	String Evaluator -Criteria Evaluation (conciseness or custom) -Custom String Evaluator -HF evaluate library (perplexity etc.)
LangSmith	0.1.40	Closed beta	Various	Various	-LangChain evaluators	-LangChain evaluators	-LangChain evaluators	-LangChain evaluators	-LangChain evaluators	-LangChain evaluators
Llamaindex	0.10	MIT license	UpTrain, DeepEval, Ragas, Tonic Validate, and various others	Various	RelevancyEvaluator RetrievalEvaluator AnswerRelevancyEvaluator GuidelineEvaluator	x	FaithfulnessEvaluator RelevancyEvaluator	x	CorrectnessEvaluator SemanticSimilarityEvaluator	x
Haystack	1.25	Apache-2.0	Ble, Basic Agent Memory Tool, Chainlit, Traceloop, and others	OpenAI models, Anthropic models, Cohere models, Hugging Face models, Amazon Bedrock models	x	x	x	Recall Mean Reciprocal Rank (MRR) Mean Average Precision (mAP)	Exact Match (EM) F1 Semantic Answer Similarity (SAS)	x
Haystack	2.0	Apache-2.0	DeepEval, Context AI, Ragas, UpTrain, Gradient, fastRAG, FastEmbed, Jina AI embedding models, INSTRUCTOR embedding models, Voyage AI embedding models, Llamacpp models, Mistral models, mixed/bread models, Ollama models	OpenAI models, Azure OpenAI models, Google AI models, Google Vertex AI models, Anthropic models, Cohere models, Hugging Face models, Amazon Bedrock/Sagemaker models, VLM Invocation Layer, FastEmbed, Jina AI embedding models, INSTRUCTOR embedding models, Voyage AI embedding models, Llamacpp models, Mistral models, mixed/bread models, Ollama models	[RagasEvaluator] CONTEXT_RELEVANCY [DeepEvalEvaluator] ANSWER_RELEVANCY [UpTrainEvaluator] CONTEXTUAL_RELEVANCE	[RagasEvaluator] ANSWER_RELEVANCY [DeepEvalEvaluator] FAITHFULNESS [UpTrainEvaluator] UTILIZATION RESPONSE_COMPLETENESS RESPONSE_CONSISTENC	[RagasEvaluator] FAITHFULNESS CONTEXTUAL_PRECISION [DeepEvalEvaluator] FAITHFULNESS [UpTrainEvaluator] RESPONSE_COMPLETENESS	[RagasEvaluator] CONTEXT_PRECISION CONTEXT_RECALL ANSWER_CORRECTNESS [SimilarityTrain] SIMILARITY CONTEXTUAL_PRECISION CONTEXT_RECALL	[RagasEvaluator] ASPECT_CRITIQUE [UpTrainEvaluator] ANSWER_CORRECTNESS RESPONSE_CONCISENESS CRITIQUE_LANGUAGE CRITIQUE_TONE_GUIDELINE_ADHERENCE	x
Deepset Cloud	0.0.41	Closed source	Haystack integrated frameworks	Haystack models	Reference analysis (top-k optimization)	x	Groundedness	x	x	x
UpTrain	0.6.12	Apache-2.0	OpenAI Evals, Llamaindex, Validate, HuggingFace, Django, LangFuse, Helicone, Zeno, and others	OpenAI models, Azure OpenAI models, Claude models, Ollama models, Together AI models, Anyscale models	Context Relevance	x	Response Relevance	Context Utilization Factual Accuracy Context Conciseness Context Reranking	Response Matching	Response Completeness Response Conciseness Response Validity Response Consistency Language Quality

Figure 5: Existing RAG Evaluation Frameworks [30]

Frameworks	Version	Metrics: Ground Truth	Grounding Questions	Metrics: Summarization	Customized Metrics	Performance Tests	Faithfulness Tests	A/B Tests	Metrics Cost	Callback Token Counts	LLM Vulnerability Scan	ML Vulnerability Scan
Giskard	2.10.0	x	x	x	✓	x	x	x	x	x	Hallucination and Misinformation (Plausibility & coherency) Harmful Information Generation Prompt Injection (character level, jailbreaking) Robustness (LLM output coherency) Output Formatting	Performance Bias Unrobustness Overconfidence Underconfidence Unethical
promptfoo	0.50.1	x	x	x	x	x	x	x	Latency Cost	x	Model-graded metrics: classifier grading - Toxicity	x
Ragas	0.1.6	x	x	x	x	x	✓	x	x	x	Aspect Critique	x
DeepEval	0.21.15	x	x	x	x	x	x	x	Latency Cost	x	HallucinationMetric BiasMetric ToxicityMetric KnowledgeRetentionMetric	x
agenta	0.12.6	x	✓	x	x	✓	✓	x	x	x	x	x
Trulens	0.27.0	Combinators Ground Truth Agreement (agreement, measure,	x	x	x	x	x	x	Latency Cost Token Counts	x	[Classification-based Stock Feedback Functions] HF toxic, OpenAI moderation_harassment, OpenAI moderation_hate, OpenAI moderation_selfharm, OpenAI moderation_sexual, OpenAI moderation_sexualminors, OpenAI moderation_violence, OpenAI moderation_violencegraphic [Generation-based Stock Feedback Functions]	x
Tonic Validate	4.0.4	x	x	x	x	x	x	x	Latency	x	x	x
LangChain	0.1.13	String Evaluator Scoring Evaluator	x	x	✓	x	x	x	Callback Token counts	x	String Evaluator Criteria Evaluation (constitutional principles)	x
LangSmith	0.1.40	-LangChain evaluators	x	x	x	x	✓	✓	Latency Cost Token Counts	x	Chat Bot Bias chmarking using Simulation	x
Llamaindex	0.10	x	x	x	x	x	x	✓	Cost	PairwiseEvaluator	GuidelineEvaluator	x
Haystack	1.25	x	x	x	x	x	x	x	x	x	x	x
Haystack	2.0	x	x	x	x	x	x	x	x	x	x	x
Deepset Cloud	0.0.41	x	x	x	x	x	x	x	Latency	x	x	x
UpTrain	0.6.12	x	x	Sub-Query Completeness Multi-query Accuracy Prompt Injection	x	x	x	x	x	Prompt Injection Jailbreak Detection	x	x

Figure 6:Existing RAG Evaluation Frameworks.2 [30]

Chapter III: State of the art

3.1 Literature Review

The literature review was conducted as a structured, multi-stage study to characterize RAG as an engineering discipline across architectural evolution, evaluation methodology, and industrial deployment. The process began with a targeted Scopus search using the following query to retrieve publications explicitly connected to retrieval-augmented generation in industrial or manufacturing contexts:

("retrieval-augmented generation" OR "retrieval augmented generation" OR RAG) AND (manufacturing OR industrial OR factory OR "quality control" OR robotics OR "collaborative robot" OR "technical documentation").

This search returned 90 papers in Scopus. To reduce database bias and improve coverage, the same search logic (with minor syntax adjustments) was applied to Web of Science (50 papers) and IEEE Xplore (40 papers), yielding 120 records before deduplication. After removing duplicates (61 excluded), 90 papers remained for the first screening stage. In Screening Step 1, titles, abstracts, and keywords were assessed to retain only papers addressing at least one of the following dimensions: (i) RAG architectures and design paradigms, (ii) RAG evaluation methodologies and diagnostics, or (iii) industrial and manufacturing applications; this stage excluded 65 papers, leaving 70 for full-text evaluation. In Screening Step 2, full texts were assessed for technical depth and relevance, retaining only papers that reported actionable architectural details, evaluation protocols, or validated industrial use cases; 38 papers were excluded at this stage. Ultimately, 31 papers were selected for detailed analysis and information extraction. The full methodology is summarized in Figure (Summarizing of Article), which provides a transparent view of the filtering pipeline from database retrieval through screening and final selection, ensuring that the final corpus is both representative of the field and tightly aligned with the thesis focus on RAG-based knowledge assistance in manufacturing.

#	Reference	Category	Application & Scope	Architecture	LLM	Dataset
[1]	**Lewis et al., 2020**	RAG (foundation)	Knowledge-intensive QA / NLP	Retriever–Generator (retrieve-then-generate; multiple docs)	Seq2Seq generator	Open-domain QA benchmarks
[2]	**Chen et al., 2017**	Retrieval QA (pre-RAG)	Open-domain QA over Wikipedia	Retriever and Reader pipeline	—	Wikipedia-based QA
[3]	**Manning et al., 2009**	IR (foundation)	Information retrieval theory	Indexing, ranking theory	—	—
[4]	**Gan et al., 2025**	RAG evaluation (survey)	RAG assessment and metrics taxonomy	Taxonomy of metrics and failure modes	LLMs (various)	Surveyed benchmarks
[5]	**Gao et al., 2025**	Long-context evaluation	Long-context RAG / LLM stress-test	Unified needle-in-a-haystack framework	LLMs (various)	U-NIAH benchmark
[6]	**Fadnis et al., 2025**	RAG evaluation (diagnostic)	RAG introspection and debugging	Instance-level analysis platform (INSPECTORRAGET)	LLM judge, task LLM	Evaluation suites (various)
[7]	**Heredia Alvaro & Barreda, 2025**	Industrial RAG	Manufacturing quality control assistance	Industrial RAG pipeline for QC	LLM (domain QA)	Manufacturing QC corpus
[8]	**Rajpurkar et al., 2016**	Dataset	Reading comprehension benchmark	Benchmark dataset (SQuAD)	—	SQuAD
[9]	**Karpukhin et al., 2020**	Retriever model	Open-domain QA retrieval	Dual-encoder dense retrieval (DPR)	—	ODQA benchmarks
[10]	**Gao et al., 2025**	IR and RAG (survey)	RAG foundations and link to IR	Conceptual unification of IR and RAG	—	Surveyed datasets
[11]	**Gao et al., 2025**	Foundation models (survey)	LLM landscape and efficiency	Model families and efficiency survey	LLMs (various)	—
[12]	**Mishra et al., 2025**	RAG enhancement	Enterprise knowledge retrieval	Metadata-enriched indexing and retrieval	LLM (metadata generation)	Enterprise documents
[13]	**Gan et al., 2025**	Industrial RAG	Automotive technical documentation QA	PDF ingestion, retrieval, QA pipeline	LLM (domain QA)	Automotive PDFs
[14]	**Gan et al., 2025**	RAG enhancement	Domain QA / enterprise	Metadata-aware retrieval pipeline	LLM-assisted	Domain corpus

#	Reference	Category	Application & Scope	Architecture	LLM	Dataset
[15]	**Jarvelin & Kekalainen, 2002**	IR evaluation	Ranking evaluation theory	DCG / nDCG framework	—	—
[16]	**Hermann et al., 2015**	Dataset / RC	Reading comprehension task	Benchmark task (CNN/DailyMail)	—	CNN/DailyMail
[17]	**Antal & Buza, 2025**	RAG evaluation (benchmark)	Compare open LLMs in RAG using RAGAS	RAGAS-based evaluation framework	Open-source LLMs	Thesis abstracts dataset
[18]	**Simon et al., 2024**	RAG evaluation (method)	Configuration dependency validation	Systematic evaluation protocol (case study)	LLM judge, RAG pipeline	Case-study dataset
[19]	**Zhang et al., 2025**	Industrial / enterprise RAG	Supply chain knowledge base management	Multi-agent automation, RAG knowledge base	LLM agents	Enterprise communications data
[20]	**McElheran et al., 2025**	Industrial AI (empirical)	Firm-level AI productivity and adoption	Empirical analysis (J-curve framing)	—	Firm / survey data
[21]	**Doyle & Cosgrove, 2019**	Manufacturing digitization	SME manufacturing digitization roadmap	Digitization roadmap / case approach	—	—
[22]	**Leng et al., 2024**	Manufacturing systems (review)	Industry 4.0–5.0 design and modelling methods	Review of methods	—	—
[23]	**Breque et al., 2021**	Industry 5.0 (policy)	Human-centric, resilient industry	Policy framework	—	—
[24]	**Luo & Zahra, 2023**	Industry 4.0 (review)	Adoption patterns in international business	Literature synthesis	—	—
[25]	**Alshahrani, 2023**	Industry 4.0 (SLR)	SCM benefits and challenges in emerging markets	Systematic literature review	—	—
[26]	**Han et al., 2025**	RAG paradigm (structured)	Multi-hop / relational retrieval	Graph-augmented retrieval and generation (GraphRAG)	LLMs (various)	Graph / relational corpora

#	Reference	Category	Application & Scope	Architecture	LLM	Dataset
[27]	**Carlier et al., 2025**	Market report	Global Industry 4.0 trends	Market analytics	—	Market datasets
[28]	**Ferdelman et al., 2025**	Market report	Global AI adoption trends	Market analytics	—	Market datasets
[29]	**Gao et al., 2024**	RAG (survey)	RAG paradigms: naive, advanced, modular	Naive / Advanced / Modular taxonomy	LLMs (various)	Surveyed benchmarks
[30]	**Applied AI Initiative, 2024**	Industrial RAG (white paper)	Industrial RAG deployment guidance	Reference architectures and governance	LLMs (various)	Industrial examples
[31]	**Wenqi Jianget al., 2025**	Retrieval-Augmented Generation	Large Language Models (LLMs)	RAGO for systematic RAG serving optimization	LLMs	

Table 1: List of References and their classification.

3.2 Theoretical Foundations of Retrieval-Augmented Generation

Before RAG existed as a named paradigm, it was implicit in the architecture of open-domain question-answering systems that recognized the impossibility of encoding all world knowledge within the parameters of a single model. Understanding this lineage is not merely historical: the design choices made in early retrieval-reader pipelines, which index granularity, retrieval scoring, and evidence aggregation, remain the load-bearing elements of every modern RAG system, including the one evaluated in this thesis.

3.2.1 Information Retrieval as the Enabling Discipline

The intellectual foundations of retrieval-augmented generation are laid in classical information retrieval (IR) theory. Manning et al. [3] provide the canonical reference: their systematic treatment of document indexing, inverted indices, vector space models, and probabilistic ranking frameworks establishes the vocabulary and the formal machinery that underpin all modern retrieval systems. Two results from IR theory are particularly consequential for RAG. First, the ranking principle that relevance is a graded property and that the utility of a retrieved document depends on its position in the ranked list motivates the Discounted Cumulative Gain (DCG) and normalized DCG (nDCG) metrics formalized by Jarvelin and Kekalainen [15], metrics that remain central to retrieval evaluation more

than two decades later. Second, the distinction between exact-match retrieval (term frequency-based, exemplified by BM25) and semantic retrieval (embedding-based) is not a modern invention but a tension that IR researchers have grappled with since the introduction of latent semantic indexing in the 1990s. The hybrid retrieval strategy adopted in this thesis combining dense vector search with BM25 lexical matching represents the contemporary resolution of a problem that Manning et al. [3] and the IR community identified long before the emergence of transformers.

Karpukhin et al. [9] mark the transition from classical IR to neural retrieval with Dense Passage Retrieval (DPR), which replaces the BM25 sparse representation with a dual-encoder architecture that maps queries and passages into a shared dense embedding space. DPR demonstrates that a retriever trained end-to-end on question-passage pairs substantially outperforms BM25 on open-domain QA benchmarks, establishing that retrieval quality is the principal determinant of downstream answer quality a finding that every RAG system designer must internalize. The ODQA benchmarks on which DPR is evaluated were themselves introduced to the community through foundational datasets: Rajpurkar et al. [8] released SQuAD, the first large-scale crowdsourced reading comprehension dataset, and Hermann et al. [16] produced the CNN/DailyMail dataset, which established abstractive reading comprehension as a tractable benchmark task. Together, [8] and [16] created the evaluation infrastructure that made systematic comparison of retrieval and generation components possible, and without which the RAG research agenda described in subsequent sections could not have been constituted.

3.2.2 The RAG Paradigm: From Retriever-Reader to Retrieve-then-Generate

Chen et al. [2] introduce the retriever-reader architecture, which is later named DrQA and serves as the direct predecessor of modern RAG. This architecture involves a TF-IDF retriever that selects relevant Wikipedia documents, and a neural reader that extracts or generates answers from the retrieved context. DrQA establishes the architectural template that Lewis et al. [1] later generalize. The contribution of Lewis et al. [1] is to replace the fixed Wikipedia index with a learnable retriever, substitute the extractive reader with a sequence-to-sequence generator, and, importantly, train the entire retrieve-then-generate pipeline end-to-end using marginalization over the retrieved documents. This approach transforms RAG from a modular pipeline into a jointly optimized probabilistic model,

demonstrating on multiple knowledge-intensive NLP tasks that a retriever-augmented generator outperforms both parametric-only language models and non-parametric retrieval baselines. The key insight of Lewis et al. [1] that the generator should be conditioned on multiple retrieved documents simultaneously, with retrieval uncertainty averaged rather than resolved by a hard top-1 decision is the origin of the context-window assembly logic that governs all subsequent RAG systems.

Gao et al. [10] offer a comprehensive survey that traces the conceptual progression from classical IR through DPR and DrQA to the modern RAG paradigm, arguing that RAG is best understood not as a new technology but as the culmination of research initiated by IR researchers decades earlier. This framing is important for the present thesis: the architectural decisions evaluated in Chapters 4 through 7, which include semantic chunking, hybrid retrieval, cross-encoder reranking, and metadata enrichment, are not arbitrary engineering choices but thoughtful implementations of principles with strong theoretical foundations.

Theoretical synthesis

The evolution from Manning et al. [3] through DPR [9] to Lewis et al. [1] establishes three principles that govern the design of the present system: (i) retrieval quality is the primary determinant of generation quality; (ii) hybrid retrieval that combines dense semantic matching with sparse lexical matching is more robust than either alone on domain-specific corpora; and (iii) the granularity at which documents are chunked is the most consequential single decision in the indexing pipeline.

3.3 Architectural Evolution of RAG Systems

Since the publication of Lewis et al. [1], the RAG literature has produced a progressively richer taxonomy of architectures. Gao et al. [29] provide the most comprehensive synthesis, organizing the field into three generations: Naive RAG, which faithfully implements the retrieve-then-generate loop with static chunking and a single retrieval call; Advanced RAG, which introduces pre-retrieval query transformation, hybrid retrieval, and post-retrieval reranking; and Modular RAG, which treats retrieval, reranking, memory, routing, and generation as independently composable modules. Each generation addresses specific failure modes of its predecessor, and each maps directly onto architectural choices evaluated in this thesis.

3.3.1 From Naive to Advanced RAG

Naive RAG, as described by Gao et al. [29], experiences three common failure modes: retrieval noise (irrelevant chunks increase the size of the context window), retrieval gaps (important information is missing from the index due to poor chunking), and generation drift (the model produces plausible but unsupported statements when the retrieved context is unclear). These failure modes are exactly what we see in Version 0 of the current system, where fixed-window sentence chunking results in fragmented chunks that split Yaskawa INFORM command descriptions across chunk boundaries, and where purely semantic retrieval fails to find passages, because relevance is indicated by exact command-name matches rather than semantic similarity.

Advanced RAG addresses these failure modes through a set of architectural improvements that are consistently supported by the literature. The hybrid retrieval approach that combines dense vector search with BM25 lexical matching is repeatedly recognized as the most significant single upgrade. The underlying idea, formalized in the IR literature [3] and empirically confirmed across multiple reviewed RAG studies, is that technical corpora include a mix of semantically rich prose and structurally precise notation (such as command names, parameter identifiers, and error codes) that necessitate both retrieval methods. Semantic search finds conceptually similar passages; BM25 locates exact string matches. Neither method alone suffices for a corpus like Yaskawa robot programming documentation, which blends explanatory prose with exact INFORM command syntax.

Post-retrieval reranking with a cross-encoder model is the second validated improvement. While a bi-encoder retriever (such as BAAI/bge-large) efficiently scores query-document similarity by encoding query and document separately, a cross-encoder re-ranker jointly attends to the query and each candidate passage, providing a more precise relevance score at the expense of increased computational load. This asymmetry is addressed in Advanced RAG by applying the costly cross-encoder only to the small candidate set returned by the more affordable bi-encoder retriever an architecture that Gao et al. [29] describe as a key feature of the Advanced RAG approach and that is directly implemented in Version 1 of the current system.

3.3.2 Graph-Augmented Retrieval

Han et al. [26] introduce GraphRAG, which extends the retrieval step from flat vector search to graph traversal over a knowledge graph constructed from the document corpus.

GraphRAG is motivated by the observation that many industrial queries require multi-hop reasoning for example, a question about which joint speed limit applies when a specific safety mode is active requires connecting information scattered across multiple sections of a robot programming manual. By constructing a knowledge graph of entity relationships and traversing it at query time, GraphRAG substantially improves performance on queries that require compositional reasoning across multiple evidence fragments. The performance gains reported by Han et al. [26] on relational corpora are substantial, but the construction of the knowledge graph adds significant engineering overhead and requires domain-specific entity recognition. The present thesis adopts a pragmatic alternative: automatic chunk-level metadata enrichment (Version 2) that approximates structural context without a full knowledge graph, providing retrieval disambiguation at a fraction of the construction cost.

3.3.3 Metadata Enrichment

Two works in the corpus specifically address the use of LLM-generated metadata to improve retrieval precision without building a full knowledge graph. Mishra et al. [12] show that adding automatically extracted metadata such as the document title, section heading, content type, and key entities to each indexed chunk produces measurable improvements in Context Precision and Context Recall in enterprise document retrieval tasks. The process is straightforward: when a retriever scores a query against a chunk, it compares the query embedding with the combination of the chunk text and its metadata, effectively enhancing the chunk's representation with structural signals that raw text alone might lack. Gan et al. [14] independently verify this finding in a domain QA setting, demonstrating that metadata-aware retrieval outperforms raw-text retrieval across multiple LLM backends, with the most significant improvements seen for queries that involve terminological variants of the indexed content.

These results directly motivate Version 2 of the current system, which introduces a ChunkMetadataEnricher module that automatically extracts title, keywords, chunk type, and position-in-document fields for every indexed chunk. The empirical question whether this enrichment improves F1, TCA, and intervention rate compared to Version 1 is a key evaluative hypothesis of this thesis. The literature suggests an affirmative answer; Chapters 5 through 7 provide the evidence.

Architectural synthesis: the V0, V1, V2 design rationale

The three-version incremental architecture in this thesis directly follows the Naive, Advanced,Modular progression outlined by Gao et al. [29]. Version 0 incorporates Naive RAG. Version 1 includes the core Advanced RAG improvements validated in the literature: semantic chunking, hybrid retrieval, cross-encoder reranking, and an upgraded embedding model. Version 2 adds the metadata enrichment layer identified by Mishra et al. [12] and Gan et al. [14], which is considered the most valuable Modular RAG addition for domain-specific corpora. This design is intentionally incremental so that the contribution of each component can be empirically measured.

3.4 RAG Evaluation Methodology

The evaluation of RAG systems is more technically challenging than assessing standalone retrievers or generators because errors can occur at any stage of the process and spread in complex ways. A retriever that provides high-precision passages might still result in poor overall performance if the generator fails to synthesize them properly; similarly, a generator that produces fluent, confident responses may hallucinate content that isn't present in the retrieved context. The existing literature has made significant progress in creating evaluation frameworks that identify failures at the component level rather than only reporting end-to-end scores, and the evaluation method in this thesis builds directly on that progress.

3.4.1 Metrics Taxonomy and the Multi-Dimensional Evaluation Imperative

Gan et al. [4] offer the most detailed taxonomy of RAG evaluation metrics to date, identifying five independent dimensions: retrieval quality (Precision, Recall, MRR, nDCG), generation quality (Exact Match, F1, BLEU, ROUGE), faithfulness to retrieved context, latency and efficiency, and human satisfaction. A key finding from their survey is that relying on a single metric can be systematically misleading: a system that optimizes retrieval precision at the expense of recall may appear stronger on precision-heavy benchmarks, while a system that enhances fluency at the cost of grounding can score well on BLEU while silently hallucinating. This thesis adopts a multi-metric approach using Precision, Recall, F1, Textual Content Accuracy (TCA), and hallucination rate to precisely avoid the single-metric bias identified by Gan et al. [4]. The non-monotonic hallucination result observed here (Version 1 shows a higher hallucination rate than Version 0 despite better F1 and TCA scores) clearly demonstrates why joint multi-metric evaluation is necessary: without the hallucination metric, Version 1 would seem unambiguously superior to Version 0 across all measured dimensions.

The importance of ranking-aware metrics is demonstrated by Jarvelin and Kekalainen [15] through the DCG and nDCG framework, which penalizes systems that return relevant documents at low ranks more heavily than those that return them at high ranks. nDCG is used in several of the reviewed evaluation frameworks and offers a theoretically motivated complement to raw Recall@k, which considers all rank positions equally. The SQuAD dataset [8] contributed to evaluation methodology not only as a benchmark but also as a design template: its formulation of reading comprehension as an extractive span-selection task established Exact Match and F1 over character spans as standard metrics for short-answer QA, metrics that are still in use in the current evaluation.

3.5 Industrial Deployment of RAG Systems

The theoretical advances described in Sections 3.2 and 3.3 acquire operational significance only when they translate into working systems deployed in real manufacturing environments. The literature reviewed here includes four works that document such deployments at varying scales and in different industrial contexts, providing both empirical evidence for the value of RAG in manufacturing and practical guidance on the engineering challenges that arise outside the laboratory.

3.5.1 Quality Control and Technical Documentation

Heredia Alvaro and Barreda [7] present a RAG system specifically designed for manufacturing quality control (QC), where operators must rapidly locate procedural specifications, tolerance standards, and corrective action procedures embedded in unstructured documentation. Their system demonstrates measurable reductions in time-to-answer and improvements in procedural accuracy compared to unassisted document search, with human satisfaction scores indicating that operators perceive the system as a reliable source of grounded guidance rather than a hallucination-prone chatbot. The evaluation uses both retrieval metrics (Context Precision, Context Recall) and operational metrics (time-to-correct-answer, procedural error rate), establishing a dual-measurement protocol that the present thesis adapts for the robot programming domain.

The automotive documentation domain is addressed by Gan et al. [13], who describe a PDF ingestion and retrieval pipeline for automotive technical manuals, with particular attention to the preprocessing challenges posed by complex PDF layouts multi-column tables, embedded diagrams, cross-referenced procedures. Their finding that OCR quality and layout-aware chunking are the primary determinants of downstream retrieval performance

reinforces the present thesis's choice of the SemanticSplitterNodeParser over fixed-window chunking: for documentation as densely structured as the Yaskawa YRC1000 INFORM manuals, semantically coherent chunking is not an optimization but a prerequisite for usable retrieval.

3.6 Industry 4.0 and Industry 5.0: The Organizational Imperative for Intelligent Assistance

RAG systems do not develop in isolation. Their adoption in manufacturing is influenced by structural pressures that the Industry 4.0 and Industry 5.0 literature have more precisely characterized over the past decade. Understanding these pressures is crucial both for situating the current thesis within the larger evolution of industrial work and for predicting the conditions under which a RAG assistant for collaborative robot programming would provide the greatest organizational benefit.

3.6.1 The Digitization Imperative and the Knowledge Access Problem

Doyle and Cosgrove [21] document the digitization of manufacturing in small and medium enterprises (SMEs), highlighting a persistent gap between the amount of structured and unstructured knowledge produced by modern manufacturing systems and workers' ability, especially those without specialized training to find and use that knowledge in real time. Their case-based analysis shows that SMEs that successfully close this gap using digital knowledge management tools see measurable improvements in process consistency and training time, whereas those that fail experience recurring procedural errors and high onboarding costs for operators. This finding directly supports the central operational metric of this thesis: the operator-intervention rate, which measures how often a non-expert user cannot complete a programming task without human help essentially, how frequently the knowledge access problem shows up as a task failure.

Luo and Zahra [24] and Alshahrani [23] offer complementary perspectives from international business and supply chain management, documenting how different firms adopt Industry 4.0 technologies across various sizes and markets. A common finding in both reviews is that technology adoption follows an S-curve, with early adopters gaining significant competitive advantages, while most firms are delayed due to integration challenges and skills shortages. RAG systems, as described in [30], are positioned to smooth

part of this adoption curve by lowering the skills required for workers, a claim that this thesis tests in a robot programming context.

3.6.2 Industry 5.0: Human-Centric Technology and the Role of Cognitive Assistance

While Industry 4.0 focuses mainly on automation and efficiency, the Industry 5.0 policy framework described by Breque et al. [23] reorients manufacturing technology toward human wellbeing, resilience, and enhancing human capabilities rather than replacing them. The collaborative robot programming scenario in this thesis is a typical Industry 5.0 example: the aim is not to remove the programmer from the process but to provide a non-expert user with a cognitive assistant that makes documentation accessible, syntax clear, and procedural steps explicit. Leng et al. [22] place this shift in the context of the broader evolution of manufacturing system design approaches, noting that Industry 5.0 introduces new requirements for human-centered interfaces and adaptive knowledge systems that extend beyond the instrumentation and connectivity goals of Industry 4.0.

The market context for these developments is quantified by Carlier et al. [27] and Ferdelman et al. [28]. Carlier et al. [27] document accelerating investment in Industry 4.0 infrastructure globally, with the adoption of AI-assisted manufacturing tools growing at a compound annual rate that validates the strategic importance of the technology. Ferdelman et al. [28] analyse AI adoption trends at the firm level, identifying knowledge management and documentation intelligence as among the highest-value AI use cases in manufacturing, with documented productivity gains concentrated in firms that successfully integrate AI assistance into operator workflows rather than deploying it as a standalone analytics tool.

McElheran et al. [20] provide the most rigorous empirical treatment of the productivity effects of industrial AI adoption, using firm-level survey data to document a J-curve pattern: productivity initially declines as firms absorb the integration costs of new AI tools and then rises sharply once the tools are embedded in standard operating procedures. This result has direct implications for the evaluation design of the present thesis: the intervention rate metric is measured at a single point in the learning curve, and the absolute rates observed (66.3% for V0, 39.0% for V1, 29.4% for V2) should be interpreted as initial-deployment baselines that are expected to improve as operators become familiar with the system's capabilities and limitations.

Chapter IV: Retriever – Augmented System (RAG)

4.1 Introduction

This chapter documents the design, configuration, and iterative development of a private RAG (Retrieval-Augmented Generation) system built on PrivateGPT v0.6.2. The system was developed and deployed in a laboratory environment to enable private, context-aware querying of Yaskawa industrial documentation without exposing any data to external services.

Development advanced through three distinct versions. Version 0 represents the stock, unmodified PrivateGPT installation with its default setup, serving as the baseline and highlighting the need for customization. Version_1 introduces comprehensive pipeline modifications to tackle the limitations of the default setup when used with specialized technical documentation. Version_2 builds on Version_1 by adding an automatic metadata enrichment layer at ingestion time, further enhancing retrieval quality without changing any other pipeline component.

Each version is described in terms of its architecture, key design decisions, and its relationship to the preceding version. Diagrams are provided for each version, showing the end-to-end workflow, component architecture, and, where relevant, detailed internal flows. A consolidated three-way comparison is presented at the end of the chapter.

4.2 Version 0 (Baseline) - Default PrivateGPT Configuration

Version 0 refers to the unmodified, out-of-the-box installation of PrivateGPT v0.6.2, running with its default local {*settings.yaml}* configuration. It serves as the project's starting point and the baseline against which subsequent versions are compared. Understanding its default behavior and limitations is essential to justify the modifications introduced in Versions 1 and 2.

4.2.1 System Overview

In its default configuration, PrivateGPT provides a fully local, privacy-preserving RAG pipeline accessible via a *{Gradio}* web interface and a *{REST API}*. The system is designed to ingest documents and answer natural-language queries solely via on-device computation, ensuring that no data leaves the user's machine. All components, including document parsing, embedding generation, vector storage, retrieval, and LLM inference, run locally without any external API calls.

The default pipeline uses a fixed-sentence-window parser for document segmentation, a compact embedding model for vector generation, a Qdrant vector store for indexing, purely

semantic vector retrieval, and a locally quantized *{Mistral-7B}* language model for response generation. Reranking is disabled by default. This configuration prioritizes simplicity and ease of installation over retrieval quality and is adequate for general-purpose document Q&A but struggles with highly technical, densely formatted documentation.

4.2.2 Document Parsing

The default document parser is *{SentenceWindowNodeParser}*, which segments documents into overlapping windows of a fixed number of sentences (default window_size =3). This approach is straightforward to configure but semantically unaware: chunk boundaries are determined by sentence count rather than by content coherence. As a result, related information is often split across adjacent chunks, and each chunk may begin or end in the middle of a conceptual unit such as a procedure, parameter description, or command specification.

For general prose documents, this limitation is manageable because sentence windows of modest size typically capture sufficient context. However, for highly structured technical documentation, such as Yaskawa robot programming manuals, which feature command tables, code listings, multi-column layouts, and densely cross-referenced procedures, fixed window_size segmentation produces fragmented, low-coherence chunks that significantly degrade retrieval quality.

4.2.3 Embedding Model

The default embedding model is *{BAAI/bge-small-en-v1.5},* a compact *{HuggingFace}* model with approximately 137 million parameters. It is loaded and executed locally using the *{Hugging Face Transformers library}*. The model is computationally efficient and has a small memory footprint, making it suitable for low-resource environments. However, its relatively limited capacity means that embeddings of technically complex text, particularly passages with specialized vocabulary, command syntax, or multi-domain content are less discriminative than those produced by larger models.

4.2.4 Retrieval

Version 0 uses a purely semantic retrieval strategy: at query time, the user's question is embedded using the same model as the documents, and the top-k most similar chunks are retrieved from Qdrant using cosine similarity (default similarity_top_k = 2). No lexical or keyword-based retrieval is performed. This means that queries that use terminology different from the exact wording in the source documents may fail to retrieve relevant passages, even when the semantic intent is equivalent. Reranking is disabled by default {*rerank.enabled* =

false in settings.yaml}, so the two retrieved chunks are passed directly to the LLM without any additional filtering or rescoring.

4.2.5 Language Model

The default LLM is *{Mistral-7B-Instruct-v0.2}* in its *{Q4_K_M}* quantized GGUF format, served via *{LlamaCPP (llama-cpp-python)}*. The model runs fully locally on CPU or GPU, depending on available hardware. With approximately 7 billion parameters and 4-bit quantization, it offers a reasonable balance between response quality and inference speed for general-purpose tasks. However, the specific task of answering complex technical queries about industrial robot programming requires precise interpretation of command syntax, parameter constraints, and operational procedures.

Layer	Component	Implementation / Notes
Document Parser	SentenceWindowNodeParser	Fixed window, window_size=3 Default PrivateGPT configuration
Metadata Enrichment	— Not present —	No enrichment in default config
Embedding Model	BAAI/bge-small-en-v1.5	137M parameters, default HuggingFace local embedding model
Vector Store	Qdrant	Cosine similarity, local instance (same across all versions)
Retrieval Strategy	Vector-only *{VectorIndexRetriever}*	similarity_top_k=2 No lexical fallback, no reranking
Reranker	Disabled (default)	rerank.enabled = false in settings.yaml
LLM	Mistral-7B-Instruct-v0.2 (GGUF)	Local via LlamaCPP, Q4_K_M quant. 7B parameters, CPU/GPU inference
Chunk Metadata	3 fields	doc_id, file_name, page_label
Observability	None (default)	No logging, no latency metrics in default configuration
UI	Gradio UI	Provided by default PrivateGPT (unchanged across all versions)

Table 2: Component-level architecture of Version 0.

4.3 Version 1 — Private_GPT1

Private_GPT1 is the first customized iteration of the PrivateGPT system, designed to address the limitations identified in Version 0 when applied to Yaskawa technical documentation. It modifies every major component of the default pipeline, including document parsing, embedding generation, retrieval, reranking, and LLM inference, while retaining the overall PrivateGPT architectural framework and its REST API interface.

4.3.1 Semantic Chunking

The SentenceWindowNodeParser was replaced with the *{SemanticSplitterNodeParser}*, which determines chunk boundaries by analyzing semantic topic changes in the text rather than using a fixed sentence count. The algorithm computes the semantic dissimilarity between consecutive sentences using the embedding model and inserts a chunk boundary only when the dissimilarity exceeds a configurable threshold.

The parser was calibrated for the Yaskawa documentation domain through systematic parameter testing. The final configuration uses a very high topic-change sensitivity threshold (breakpoint_percentile_threshold = 98) and an overlap buffer (buffer_size = 3). The high percentile threshold dramatically reduces spurious splits at ambiguous boundaries, ensuring that chunks correspond to complete conceptual units such as an entire command description or procedure. The buffer preserves a small overlap at boundaries to prevent information loss. After re-ingestion with this configuration, the median chunk length increased substantially, and chunk coherence improved measurably.

4.3.2 Embedding Model Upgrade

The default *{bge-small-en-v1.5 model}* (137M parameters) was replaced with *{BAAI/bge-large-en-v1.5}*, which has 335 million parameters and was specifically pre-trained and fine-tuned to maximize semantic similarity. Because the Yaskawa documentation is entirely in English, the monolingual focus of the *{bge-large model}* is not a constraint, and its larger capacity directly benefits the quality of embeddings for technically complex passages. The primary trade-off is increased memory consumption and longer per-chunk embedding time, both of which are manageable with the available laboratory hardware.

4.3.3 Hybrid Retrieval

A hybrid retrieval strategy was implemented that combines dense vector retrieval with sparse *{BM25 keyword-based retrieval}*. The motivation for this hybrid approach is that technical manuals frequently contain precise terms, command names, error codes, and parameter identifiers that a purely semantic retrieval model may not reliably match when the query uses exactly the same terminology as the source. BM25 retrieval provides lexical coverage that complements semantic similarity.

Due to a version incompatibility between *{LlamaIndex 0.11.x}* and PrivateGPT 0.6.2, the off-the-shelf *{HybridRetriever component}* could not be used directly. Instead, a fully custom hybrid retrieval implementation was developed, using two independent retrieval

channels: a *{VectorIndexRetriever}* over chunks stored in Qdrant (cosine similarity) and a *{BM25 index}* built directly from the stored text nodes using a Unicode word-character tokenizer (splitting on the \w+ regex pattern with lowercasing). *{The BM25 index}* is cached at first launch to minimize overhead on subsequent queries. Chunks shorter than 20 characters are excluded from the BM25 index to reduce noise. Results from both channels are merged using OR union and deduplicated by node identifier.

A fanout strategy (fanout_factor = 3) is applied to both channels before merging: each retriever over-retrieves a candidate set three times the size of the final top-k, thereby widening the search space and improving recall. After deduplication, a cross-encoder *{reranker (cross-encoder/ms-marco-MiniLM-L-2-v2)}* assigns a unified relevance score to all candidates, enabling fair comparison between vector-retrieved and BM25-retrieved chunks regardless of their original score scales. Only the top-k highest-scoring chunks are retained for the LLM context. Each retained chunk is tagged with its provenance (vector, BM25, or hybrid) for auditability.

4.3.4 Language Model

Although the system was designed for fully local operation, early trials with the 7B Mistral model yielded unsatisfactory response quality for complex technical queries. Given the hardware limitations of the laboratory workstation, LLM computation was temporarily outsourced to Google Gemini 2.5 Flash via its external API endpoint. This decision was justified by the absence of sensitive data in the documentation corpus (only publicly available Yaskawa manuals were used) and by the expectation that a hardware upgrade would restore full local operation in future iterations. The embedding model and vector store continued to run locally.

Layer	Component	Implementation / Notes
Document Parser	SemanticSplitterNodeParser	buffer_size=3 Semantic topic-change detection
Metadata Enrichment	— Not present —	No enrichment in Version 1
Embedding Model	BAAI/bge-large-en-v1.5	335M parameters, higher semantic fidelity for technical English text
Vector Store	Qdrant	Cosine similarity, local instance (same across all versions)
Retrieval Strategy	Hybrid: Vector and BM25 (custom)	Fanout=3 per channel, OR union Deduplication by node_id

Layer	Component	Implementation / Notes
Reranker	cross-encoder/ms-marco-MiniLM-L-2-v2	Applied post-merge, top-k pruning Provenance tags per chunk
LLM	Google Gemini 2.5 Flash	External API endpoint (hardware constraints on local 7B)
Chunk Metadata	3 fields	doc_id, file_name, page_label (same as Version 0)
Observability	Latency instrumentation	JSON snapshot, per-stage timestamps session/query identifiers
UI	Gradio UI	Provided by default PrivateGPT (unchanged across all versions)

Table 3: Component-level architecture of Version_1.

4.4 Version 2 — Private_GPT2

Private_GPT2 is an incremental enhancement of Private_GPT1 that adds automatic chunk-level metadata enrichment during document ingestion. The entire pipeline inherited from Version 1 is preserved without modification; the sole addition is a metadata enrichment stage inserted between the chunking and embedding generation steps. This design ensures that the improvement introduced by Version 2 is isolated, measurable, and independently comparable to Version 1.

4.4.1 Motivation for Metadata Enrichment

In Private_GPT1, each stored chunk carries only three metadata fields: *doc_id, file_name, and page_label*. These fields describe the provenance of the chunk but provide no semantic information about its content. Retrieval decisions are therefore based entirely on vector cosine similarity and BM25 keyword matching against the raw chunk text. While this is effective for many queries, it provides no signal about the structural type of a chunk (a procedure, a parameter table, a code listing), no explicit representation of its key topics, and no title-level descriptor that could disambiguate semantically similar but topically distinct passages.

Private_GPT2 addresses this limitation by automatically extracting four additional metadata fields per chunk at ingestion time which *title, keywords, chunk_type, and position_in_document*, so using a configurable, strategy-based enrichment pipeline. These richer metadata fields provide additional retrieval signals and enable future filtering, boosting, or weighting strategies at query time.

4.4.2 Enrichment Architecture

The enrichment pipeline is implemented in the new module chunk_metadata_enrichment.py and orchestrated by the *{ChunkMetadataEnricher class}*. The enricher applies a configurable set of strategies to each chunk node, each strategy implementing the abstract MetadataEnrichmentStrategy interface. Four strategies are available:

- *RuleBasedTitleExtractor*: extracts a representative title from a chunk of chunk using heuristic rules, such as identifying the first non-trivial line or matching heading-like patterns.
- *RuleBasedKeywordExtractor*: identifies the most salient keywords from the chunk using frequency analysis and positional weighting, without requiring an external model.
- *ChunkTypeClassifier:* classifies each chunk into a structural category (paragraph, table, code listing, header, etc.) based on text formatting patterns and token distributions.
- *LLMBasedSummarizer:* generates a natural language summary of the chunk using the configured LLM. This strategy is disabled by default to avoid significant ingestion overhead from repeated API calls.

In addition to the strategy-derived fields, the coordinator class assigns a position_in_document index to each chunk, recording its sequential order within the source document.

4.4.3 Pipeline Integration

The enrichment stage is integrated into the ingestion pipeline via a new *{EnhancedIngestionHelper} class* that wraps and extends the original *{IngestionHelper}*. In all four ingest component classes—*SimpleIngestComponent*, *BatchIngestComponent*, *ParallelizedIngestComponent*, and *PipelineIngestComponent*—the _save_docs() method was modified to apply enrichment after transforming documents into *{LlamaIndex nodes}* and before insertion into Qdrant. The original IngestionHelper is retained for backward compatibility; the entire enrichment pipeline can be disabled at runtime by setting *{metadata_enrichment.enabled = false in settings.yaml}*, producing behavior identical to Version 1.

A new Pydantic configuration model, *MetadataEnrichmentSettings*, was added to settings.py, exposing individual flags to enable or disable each enrichment strategy independently. A corresponding configuration block was added to *settings.yaml* with sensible defaults *(title extraction, keyword extraction, and type classification enabled; summarization disabled)*.

Layer	Component	Implementation / Notes
Document Parser	SemanticSplitterNodeParser	buffer_size=3 Semantic topic-change detection
Metadata Enrichment	ChunkMetadataEnricher (NEW)	title, keywords, chunk_type, position_in_document per chunk
Embedding Model	BAAI/bge-large-en-v1.5	335M parameters, higher semantic fidelity for technical English text
Vector Store	Qdrant	Cosine similarity, local instance (same across all versions)
Retrieval Strategy	Hybrid: Vector + BM25 (custom)	Fanout 3 per channel, OR union Deduplication by node_id
Reranker	cross-encoder/ms-marco-MiniLM-L-2-v2	Applied post-merge, top k pruning Provenance tags per chunk
LLM	Google Gemini 2.5 Flash	External API endpoint (hardware constraints on local 7B)
Chunk Metadata	7 fields	doc_id, file_name, page_label,title, keywords, chunk_type, position
Observability	Latency instrumentation	JSON snapshot and per-stage timestamps session/query identifiers
UI	Gradio UI	Provided by default PrivateGPT (unchanged across all versions)

Table 4: Component-level architecture of Version_2

4.4.4 Algorithms for Level Metadata Enrichment

Version 2 introduces an enrichment stage between chunking and embedding. Each chunk is enhanced with a title, keywords, chunk type, and document position. The module *chunk_metadata_enrichment.py* defines four strategies that are managed by *ChunkMetadataEnricher*.

Metadata Schema - The ChunkMetadata Pydantic model specifies the fields included in each chunk:

```
class ChunkMetadata(BaseModel):
    title: Optional[str] = None
    keywords: List[str] = []
    summary: Optional[str] = None
    chunk_type: Optional[str] = None
    section_header: Optional[str] = None
```

```
    position_in_document: Optional[int] = None
```

Strategy Interface - All strategies inherit from *MetadataEnrichmentStrategy* and implement a single *enrich() method*:

```
class MetadataEnrichmentStrategy:
    def enrich(self, text: str, existing_metadata: Dict) -> Dict:
        raise NotImplementedError
```

Rule-Based Title Extraction - *RuleBasedTitleExtractor* scans the first three lines of a chunk using three patterns in priority order, then falls back to the first short line if none match:

```
class RuleBasedTitleExtractor(MetadataEnrichmentStrategy):
    def __init__(self):
        self.patterns = [
            (r'^#{1,6}\s+(.+)$',       1.0),   # Markdown header
            (r'^\d+\.[\.\d]*\s+(.+)$', 1.0),   # Numbered section
            (r'^([A-Z][A-Z\s]{10,100})$', 0.8),  # ALL-CAPS phrase
        ]

    def enrich(self, text: str, existing_metadata: Dict) -> Dict:
        for line in text.strip().split('\n')[:3]:
            for pattern, confidence in self.patterns:
                match = re.match(pattern, line.strip())
                if match:
                    return {'title': match.group(1).strip(),
                            'title_confidence': confidence,
                            'title_method': 'rule_based'}
        # Fallback: first line if short and not a sentence
        first = text.strip().split('\n')[0].strip()
        if first and len(first) < 100 and not first.endswith('.'):
            return {'title': first, 'title_confidence': 0.5,
                    'title_method': 'first_line'}
        return {}
```

Rule-Based Keyword Extraction - RuleBasedKeywordExtractor tokenises the chunk, removes stop words, ranks by frequency, and overlays capitalised phrase detection to surface named entities:

```
class RuleBasedKeywordExtractor(MetadataEnrichmentStrategy):
    def __init__(self, max_keywords: int = 5):
        self.max_keywords = max_keywords
        self.stop_words = {'the','a','an','and','or','in','on','at',
                           'to','for','of','with','is','was', ...}

    def enrich(self, text: str, existing_metadata: Dict) -> Dict:
        words = re.findall(r'\b[a-zA-Z]{3,}\b', text.lower())
        freq  = {w: words.count(w) for w in words
```

```
            if w not in self.stop_words}
        # Capitalised phrases as named-entity candidates
        caps  = re.findall(r'\b[A-Z][a-z]+(?:\s+[A-Z][a-z]+)*\b', text)
        # Merge: frequency-ranked tokens + capitalised phrases
        candidates = [w for w,_ in sorted(freq.items(),
                    key=lambda x: x[1], reverse=True)]
        candidates += [c.lower() for c in caps]
        # Deduplicate and truncate
        seen, keywords = set(), []
        for c in candidates:
            if c not in seen and c not in self.stop_words:
                seen.add(c); keywords.append(c)
            if len(keywords) >= self.max_keywords: break
        return {'keywords': keywords, 'keyword_method': 'rule_based'}
```

Chunk-Type Classification - *ChunkTypeClassifier* assigns a structural label using a priority cascade, which can be *code, list, table, header, or paragraph*.

```
class ChunkTypeClassifier(MetadataEnrichmentStrategy):
    def enrich(self, text: str, existing_metadata: Dict) -> Dict:
        t = text.strip()
        # Code block
        if t.startswith('```') or t.startswith('    '):
            return {'chunk_type': 'code'}
        # List (majority of lines match list patterns)
        lines = t.split('\n')
        list_lines = sum(1 for l in lines if
                    re.match(r'^\s*[-*+\d]', l))
        if list_lines > len(lines) * 0.5:
            return {'chunk_type': 'list'}
        # Table
        if t.count('|') > 3:
            return {'chunk_type': 'table'}
        # Header
        if re.match(r'^#{1,6}\s+', t):
            return {'chunk_type': 'header'}
        return {'chunk_type': 'paragraph'}
```

ChunkMetadataEnricher Coordinator - The coordinator applies all active strategies sequentially to each node. Per-strategy and per-node errors are caught independently so a single failure never stops the batch:

```
class ChunkMetadataEnricher:
    def enrich_node(self, node: BaseNode,
                    position: Optional[int] = None) -> BaseNode:
        text     = node.get_content()
        metadata = node.metadata.copy()
        for strategy in self.strategies:
            try:
                metadata.update(strategy.enrich(text, metadata))
            except Exception as e:
```

```
            logger.warning(f'{strategy.__class__.__name__} failed: {e}')
        if position is not None:
            metadata['position_in_document'] = position
        node.metadata = metadata
        node.excluded_embed_metadata_keys = ['doc_id',
                                             'position_in_document']
        node.excluded_llm_metadata_keys   = ['file_name', 'doc_id',
            'page_label', 'keyword_method', 'title_method',
            'summary_method', 'title_confidence', 'position_in_document']
        return node

    def enrich_nodes(self, nodes: List[BaseNode]) -> List[BaseNode]:
        enriched = []
        for i, node in enumerate(nodes):
            try:    enriched.append(self.enrich_node(node, position=i))
            except: enriched.append(node)  # keep original on failure
        return enriched
```

The *excluded_embed_metadata_keys* list keeps positional and system fields out of the embedding vector. The *excluded_llm_metadata_keys* list keeps diagnostic fields (_method, _confidence) out of the LLM context window.

Factory Function and settings.yaml - The factory *create_metadata_enricher()* constructs the enricher from a config dictionary. Summarisation is off by default:

```
def create_metadata_enricher(llm=None, config=None):
    config = config or {}
    return ChunkMetadataEnricher(
        llm=llm,
        enable_title_extraction   = config.get('enable_title_extraction',   True),
        enable_keyword_extraction = config.get('enable_keyword_extraction', True),
        enable_summarization      = config.get('enable_summarization',      False),
        enable_type_classification= config.get('enable_type_classification',True),
    )
```

The corresponding *settings.yaml* block that controls all flags at runtime:

```
metadata_enrichment:
  enabled: true
  enable_title_extraction: true
  enable_keyword_extraction: true
  enable_summarization: false   # disabled: high per-chunk LLM cost
  enable_type_classification: true
```

Chapter V: Evaluation Metrics for Retrieval-Augmented Generation Systems

Before defining individual metrics, this section presents the structure of the evaluation database from which all quantitative results in this chapter are derived. The database is organized as a flat spreadsheet with one row per evaluated use case and 47 columns partitioned into eight functional groups (A–H). Each group captures a distinct dimension of system performance, from task definition through to end-to-end execution correctness.

Group	Group Name	Key Metrics / Columns	Purpose
A	Task & Query Definition	Task ID, Task Name, Difficulty, Query, qid, Ground Truth	Defines the evaluation scope: what was asked, its difficulty, and the annotated correct answer.
B	System Outputs & Performance Logs	Answer, Time (ms), Average Time, Latency	Records what the system produced and how fast, enabling latency and throughput analysis.
C	Novice User Efficiency	Total Interaction Turns, Theoretical Minimum Turns, Effective Turns (ET), Wrong Queries (WQ), Learning Rate, Intervention Count	Quantifies operator-side effort: how many attempts were needed and how many were wasted.
D	System Quality (Instruction-Level)	Task Success, Ineffective / Interfering Instruction, Refusal, Instruction Success Rate (ISR)	Evaluates whether each response satisfied the instruction intent, flagging failure modes.
E	System-Level Metrics (End-to-End)	Precision, Recall, F1 Score (end-to-end token overlap)	Measures overall token-level alignment between the generated answer and the ground truth.
F	Retriever Metrics	Precision, Recall, Recall@1k/3k/5k, Hit@1k/3k/5k, AP, MAP, MRR, nDCG@1k/3k/5k	Evaluates chunk-level retrieval quality across multiple rank cutoffs and ranking-sensitive metrics.
G	Generator / Synthesis Metrics	Context Utilisation, Noise Sensitivity (Relevant / Irrelevant), Hallucination, Self-Knowledge, Faithfulness	Characterises how faithfully the LLM grounds its response in the retrieved context.
H	Execution Correctness Metrics	Number of Steps, Task Completion Accuracy (TCA), Goal Achievement (GA), Instruction-to-Execution Fidelity (IEF)	Captures physical-task-level outcomes: were the correct steps produced and successfully executed on the robot?

Table 5: Evaluation Database - Column Group Overview

5.1 Introduction

Rigorous quantitative evaluation is crucial for trustworthy systems research. In Retrieval-Augmented Generation (RAG), where a retriever component supplies a language model with relevant document snippets before generating responses, the evaluation challenge is twofold: assessing both the accuracy of retrieval and the quality of the generated output. This dual focus distinguishes RAG evaluation from traditional language model benchmarking, which

usually considers outputs alone, and from classic information retrieval assessment, which focuses solely on the ranking of retrieved documents.

Our evaluation framework builds on established principles from information retrieval, as outlined by Jurafsky and Martin, incorporates position-sensitive ranking metrics from Jarvelin and Kekalainen, and is guided by recent advancements in RAG-specific evaluation methods. It is implemented as a standalone, dependency-free Python module and applied to three versions of the PrivateGPT system described earlier, enabling consistent, reproducible measurement of improvements at each step.

Metrics are divided into five groups that cover the entire evaluation lifecycle of a deployed RAG system. The first group includes traditional retrieval measures which Precision, Recall, F1, Recall@k, Hit@k, Mean Reciprocal Rank (MRR), Average Precision (AP), Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (nDCG@k) which evaluate how well the retriever surfaces and ranks relevant document chunks. The second group contains novice-user efficiency metrics, such as Total Interaction Turns, Theoretical Minimum Turns, Effective Turns (ET), Wrong Queries (WQ), Learning Rate, and Intervention Count, which quantify the cognitive load and iteration overhead of an inexperienced operator interacting with the system. The third group assesses instruction-level quality through Task Success, Ineffective or Interfering Instruction, Refusal, and Instruction Success Rate (ISR), measuring the system's reliability in executing operator commands without ambiguity or refusal. The fourth group captures generator and synthesis quality via Context Utilization, Noise Sensitivity (Relevant and Irrelevant), Hallucination Rate, Self-Knowledge, and Faithfulness, characterizing the extent to which the language model grounds its responses in retrieved evidence rather than parametric memory. The fifth group provides end-to-end execution correctness metrics — Task Completion Accuracy (TCA), Goal Achievement (GA), and Instruction-to-Execution Fidelity (IEF) — which evaluate system performance based on observable physical outcomes in the industrial robotic deployment.

5.2 Precision

Precision is the most basic metric of selectivity in information retrieval. It measures the conditional probability that a retrieved document chunk is relevant to the query, given that it was retrieved. Equivalently, it is the fraction of retrieved items that are relevant. Following the notation of Jurafsky and Martin, precision is defined as:

$$Precision = |R| / |T| \quad (1)$$

where R ⊆ T denotes the set of relevant retrieved document chunks and T denotes the total set of retrieved document chunks. In the present implementation, T is the ranked list of chunk node identifiers returned by the hybrid retriever for a given query, and R is the subset of those identifiers that appear in the human-annotated ground-truth set gold_ids for that query.

Precision measures the system's ability to avoid false positives, which is the retrieval of chunks that are semantically or topically unrelated to the query. A high precision value indicates that the ranking function effectively discriminates relevant document passages from irrelevant ones, ensuring that the context window presented to the language model is information-dense. In contrast, a low precision score indicates that the retrieved context is filled with off-topic material, increasing the likelihood that the language model will generate inaccurate or hallucinated responses by attending to misleading context.

5.3 Recall

Recall measures the proportion of relevant documents retrieved by a retrieval system. It estimates the conditional probability that a relevant document chunk is retrieved by the system, given that it exists in the collection. The formal definition is:

$$Recall = |R| / |U| \quad (2)$$

where U is the set of all document chunks in the collection that are relevant to the query (Jurafsky and Martin). Recall, therefore, measures the system's ability to minimize false negatives, which are failures to retrieve relevant chunks that exist in the indexed corpus. High recall is especially important in technical documentation areas like industrial robot programming, where missing relevant information can have direct operational consequences. For example, retrieving an incomplete procedure, such as a set of movement commands with one step omitted, could lead to incorrect or unsafe robot behavior. Unlike precision, recall does not penalize the retrieval of irrelevant documents: a system that retrieves all relevant chunks along with many irrelevant ones still achieves perfect recall.

5.4 F1 Score

The F1 score provides a single scalar summary of the balance between precision and recall. It is defined as the harmonic mean of the two, giving equal weight to each:

$$F1 = (2 \times Precision \times Recall) / (Precision + Recall) \quad (3)$$

The harmonic mean is preferred over the arithmetic mean because it penalizes extreme imbalances between precision and recall. A system with very high precision but near-zero

recall, or vice versa, will receive a low F1 score regardless of the dominant metric's high value. This property makes F1 a robust summary metric for retrieval quality without sacrificing either objective.

In the context of RAG evaluation, the F1 score is particularly useful for comparing system versions, as both objectives, avoiding irrelevant chunks (precision) and covering all relevant chunks (recall), must be satisfied to produce complete and trustworthy responses.

5.5 Recall@k

Recall@k is a threshold-based variant of recall that restricts the evaluation to the top-k retrieved document chunks. It is defined as the proportion of relevant instances retrieved within the top-k results, over the total number of relevant cases:

$$Recall@k = |RD \cap Top\text{-}k(d)| / |RD| \quad (4)$$

where RD is the set of relevant documents and Top-k(d) is the set of the top-k retrieved documents. In the current implementation, Recall@k is computed for $k \in \{1, 3, 5\}$, providing a multi-threshold view of coverage quality.

Recall@k directly addresses a practical constraint of RAG systems: the language model's context window limits the number of chunks that can be included in the prompt. Even if a system retrieves 20 chunks in principle, only the top 3 or top 5 may be passed to the LLM. Recall@k therefore measures whether the system surfaces the needed knowledge within the operationally relevant retrieval window, rather than at an arbitrarily deep rank.

5.6 Hit@k

Hit@k is a binary metric indicating whether the retrieval system returns at least one relevant document chunk among the top-k results. Unlike Recall@k, which quantifies the fraction of relevant chunks retrieved, Hit@k returns 1 if any relevant chunk appears in the top-k and 0 otherwise:

$$Hit@k = 1 \text{ if any}(d \in RD \text{ for } d \in Top\text{-}k(d)), \text{ else } 0 \quad (5)$$

Hit@k serves as a tolerant pass/fail test for retrieval success. A system that achieves Hit@k = 0 for a given query has experienced a total retriever failure for that query; no relevant knowledge was surfaced within the operationally relevant retrieval window, while Hit@k = 1 simply confirms that at least a starting point for the answer was retrieved, regardless of how many other relevant chunks were missed.

This metric is especially useful for procedural technical queries, where even a single relevant chunk can help the language model produce a somewhat correct response. The average Hit@k across all queries indicates the proportion of queries for which the retriever avoids complete failure.

5.7 Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank is a standard evaluation metric for ranked retrieval systems that measures how quickly the retriever surfaces the first relevant result. It is defined as the mean of the reciprocal ranks of the first relevant document retrieved across a set of queries:

$$MRR = (1/|Q|) \times \Sigma_i (1 / rank_i). \quad (6)$$

where |Q| is the number of queries and $rank_i$ is the rank position of the first relevant document chunk for the i-th query (Jurafsky and Martin,). If no relevant document is retrieved for a given query, the reciprocal rank for that query is 0.

The reciprocal rank function assigns higher scores to systems that place the first relevant result at a high rank: a relevant chunk at rank 1 contributes 1.0, at rank 2 contributes 0.5, at rank 3 contributes 0.333, and so on. MRR is particularly relevant in the RAG context because the language model's response quality depends directly on the order in which chunks appear in the retrieved context. A high MRR indicates that the retriever consistently places the most relevant chunk near the top of the results, which directly improves response quality.

5.8 Average Precision (AP)

Average Precision (AP) summarizes the quality of a ranked retrieval result by computing the average of precision values at each rank position where a relevant document chunk is encountered. It provides a scalar representation of the area under the precision–recall curve for a single query:

$$AP = (1/|R_r|) \times \Sigma_k P(k) \times rel(k) \quad (7)$$

where P(k) is the precision at rank cutoff k, rel(k) is an indicator function equal to 1 if the document at rank k is relevant and 0 otherwise, and $|R_r|$ is the total number of relevant documents retrieved (Jurafsky and Martin). AP rewards systems that rank relevant documents early: a system that retrieves all relevant documents but places them at the bottom of the ranked list will receive a lower AP than one that places them at the top.

5.9 Mean Average Precision (MAP)

Mean Average Precision (MAP) extends AP to a multi-query evaluation by computing the arithmetic mean of AP scores across all queries in the evaluation set:

$$MAP = (1/|Q|) \times \Sigma_i AP(i) \quad (8)$$

where |Q| is the total number of evaluation queries (Jurafsky and Martin). MAP is the primary overall retrieval metric in the current evaluation framework and is used as the official metric in TREC benchmark assessments. A high MAP means the retrieval system reliably ranks relevant chunks near the top across all query types, which is ideal for a RAG system handling various technical queries.

5.10 Normalized Discounted Cumulative Gain (nDCG@k)

Normalized Discounted Cumulative Gain (nDCG@k) is a position-aware retrieval metric that considers the rank of relevant documents by applying a logarithmic discount to each retrieved chunk's relevance contribution. Unlike AP, which treats all relevant documents equally regardless of their position in the ranking, nDCG@k explicitly penalizes relevant documents that are lower in the ranked list.

The metric is computed in two steps. First, the Discounted Cumulative Gain at rank k (DCG@k) is computed as:

$$DCG@k = \Sigma_{i=1}^{k} rel_i / log_2(i + 1) \quad (9)$$

where rel_i represents the graded relevance of the result at position i. In the binary relevance case used in this evaluation, rel_i equals 1 if the chunk at position i is in the gold annotation set and 0 otherwise. A relevant chunk at rank 1 contributes 1.0, at rank 2 contributes 0.631, at rank 3 contributes 0.5, and so forth.

DCG@k is then normalized by the Ideal DCG at rank k (IDCG@k), the maximum possible DCG@k achievable by placing all relevant chunks at the top:

$$nDCG@k = DCG@k / IDCG@k$$

The normalized value ranges from 0 (no relevant chunks retrieved) to 1 (perfect ranking). nDCG@k is computed for $k \in \{1, 3, 5\}$ in the present evaluation. Its inclusion is specifically motivated by the manufacturing context: if a procedure requires steps to be executed in sequence, a retrieval result that places later steps before earlier ones may lead to a correct but improperly ordered response, a failure mode that nDCG@k captures with a level of granularity that simpler metrics cannot provide.

5.11 Novice User Efficiency Metrics

In an industrial deployment setting, the quality of a RAG system cannot be judged only by retrieval metrics. An equally critical aspect is how efficiently an inexperienced operator can leverage the system to complete a task. This section outlines five metrics that collectively measure the cognitive and procedural load placed on a novice user: Interaction Turns, Effective Turns, Wrong Queries, Learning Rate, and Intervention Count.

5.11.1 Total Interaction Turns and Theoretical Minimum Turns

Total Interaction Turns (T) count the total number of query–response exchanges a user submits to the RAG system to complete a task. Each reformulation, clarification request, or follow-up query counts as an additional turn. The Theoretical Minimum Turns (T_min) represents the smallest number of exchanges needed to complete the task if the system responded perfectly each time, which is usually just one turn for a well-defined procedural query. Together, T and T_min define the interaction overhead: a ratio T / T_min close to 1.0 indicates efficient guidance by the system, while a large ratio shows that the user had to reformulate or retry queries multiple times before getting useful information.

5.11.2 Effective Turns (ET)

Effective Turns (ET) refers to the number of interaction turns that result in a response considered useful or relevant to the task, excluding turns that produce refusals, irrelevant answers, or responses needing immediate correction. Formally:

$$ET = T - WQ \quad (10)$$

where T is the total number of interaction turns and WQ is the number of incorrect queries. ET measures the productive portion of the interaction and offers a more accurate gauge of conversational efficiency than just counting turns. In this evaluation, ET ranges from 2 (theoretically minimal) to 4, indicating the procedural complexity of the industrial use cases.

5.11.3 Wrong Queries (WQ)

Wrong Queries (WQ) counts the number of interaction turns in which the user's query was malformed, off-topic, or produced a response that did not advance the task. A query is classified as wrong if (a) the RAG system returned a refusal or an irrelevant response, (b) the retrieved context contained zero relevant chunks as confirmed by the ground-truth annotation, or (c) the generated answer was judged factually incorrect by the human evaluator.

5.11.4 Learning Rate

Learning Rate (LR) in this context is not a gradient-based optimization parameter but a user-centered interaction metric defined as the proportion of interaction turns that were unproductive, that is, the ratio of wrong queries to total turns.

$$LR = WQ / T \quad (11)$$

A lower Learning Rate indicates that the system reliably returns actionable responses on the first attempt.

5.11.5 Intervention Count

Intervention Count records the number of times a human expert was needed to step in and correct or supplement the RAG system's output during a task session. Unlike Wrong Queries, which capture user-side reformulation effort, Intervention Count measures expert-side correction effort, that is, the cost borne by a supervisor or senior engineer who must compensate for system shortcomings in a live industrial environment. A system with zero interventions across all use cases has achieved full operational autonomy for those tasks. Basically, if expert help is needed to complete a task, it records “Yes”; otherwise, “No.”

5.12 Instruction-Level Quality Metrics

While retrieval and user-efficiency metrics describe the pipeline at the document and session level, instruction-level quality metrics assess whether the system's responses meet the specific instructions given in each individual query. In the industrial robot programming context, an instruction is a distinct step-level command such as "Set the argument for OR_INIT" or "Transfer external data to the pendant." Three binary indicators and one overall rate are used for this evaluation aspect.

5.12.1 Task Success

Task Success is a binary label (1/0) assigned to each use-case interaction, indicating whether the RAG system's response was sufficient for the operator to complete the intended step without needing additional information from another source. A "1" judgment requires that the response addressed the query's intent, contained no materially misleading information, and allowed the operator to move on to the next step. A "0" judgment is given when the response was incomplete, incorrect, or so generic that the operator could not act on it.

5.12.2 Ineffective and Interfering Instructions

An Ineffective Instruction is one that fails to advance the task but does not actively mislead the operator; for example, a vague or overly generic answer that provides no actionable

guidance. An Interfering Instruction is one whose response actively contradicts the correct procedure, introduces a conflicting step, or references an incorrect parameter value, thereby increasing the likelihood of a downstream execution error. Both indicators are recorded as binary flags for each interaction turn. Interfering responses pose a higher operational risk: an operator who receives no useful guidance may seek an alternative source, whereas one who receives incorrect guidance may proceed with an unsafe or erroneous action.

5.12.3 Refusal

A Refusal is recorded when the RAG system explicitly declines to answer the query, returns a message stating that the relevant information is not available in the knowledge base, or produces an empty or near-empty response. Refusals are distinct from hallucinations: a refusing system correctly acknowledges the boundary of its knowledge, whereas a hallucinating system fabricates an answer beyond that boundary. In the present deployment, the PrivateGPT system is configured to respond only from retrieved context, meaning refusals typically occur when the retriever returns zero relevant chunks for a query. Refusal rate is therefore a useful diagnostic for identifying topic gaps in the indexed document corpus.

5.12.4 Instruction Success Rate (ISR)

Instruction Success Rate (ISR) is the aggregate measure of instruction-level quality, defined as the proportion of interaction turns across all evaluated use cases for which the system achieved Task Success:

$$ISR = Task\ Success/Effective\ Turns(ET) \quad (12)$$

ISR = 1.0 indicates that every interaction produced a task-successful response. In the present evaluation, all three RAG versions achieved ISR = 1.0 on the binary instruction-success label, though this result must be interpreted alongside TCA and Hallucination Rate, which reveal that success at the instruction level does not guarantee correctness at the task-completion level.

5.13 Generator and Synthesis Quality Metrics

Retrieval quality alone does not determine RAG output quality. The language model must synthesize retrieved chunks into a coherent, grounded, and factually accurate response. This section defines six metrics that characterize the generator component's behavior: Context Utilization, Noise Sensitivity (Relevant and Irrelevant), Hallucination Rate, Self-Knowledge, and Faithfulness.

5.13.1 Context Utilization

Context Utilization measures how much the language model uses the retrieved context in its response. A model with high context utilization relies heavily on the retrieved chunks to create its answer; a model with low context utilization might ignore the retrieved evidence and depend more on parametric knowledge learned during pre-training. Context Utilization is defined as the proportion of factual claims in the generated response that can be traced back to tokens or semantic content in the retrieved chunks, evaluated by token overlap or semantic entailment scoring against the retrieved context.

5.13.2 Noise Sensitivity (Relevant and Irrelevant)

Noise Sensitivity quantifies how much the generator's output degrades when the retrieved context contains extraneous or misleading material. Two sub-variants are defined. Noise Sensitivity (Relevant) measures the proportion of cases in which the presence of additional relevant-but-distracting chunks causes the model to produce a less accurate response compared to the single-chunk oracle condition. Noise Sensitivity (Irrelevant) measures the same degradation when the context window contains genuinely off-topic chunks which a direct consequence of low retrieval Precision. A robust generator should exhibit low Noise Sensitivity on both sub-variants regardless of the retrieval context composition. High Noise Sensitivity (Irrelevant) is particularly concerning in safety-critical domains because it indicates that low-precision retrieval directly translates into generation errors.

5.13.3 Hallucination Rate

The Hallucination Rate measures the proportion of responses that include at least one factual claim not supported by, contradicted by, or missing from the retrieved context. In this evaluation, hallucination is rated on a scale from 0 (completely grounded) to 1 (completely fabricated). A response is considered hallucinated if it states a specific parameter value, command name, or procedural step that does not appear in any retrieved chunk and conflicts with the annotated ground truth. Hallucinations are especially risky in robotic programming: a hallucinated command argument or wrong procedure sequence can lead to unsafe robot behavior with physical consequences. Therefore, the hallucination metric is a key safety indicator in the evaluation framework.

5.13.4 Self-Knowledge

Self-Knowledge measures the system's ability to accurately recognize and communicate the limits of its own knowledge. A system with high self-knowledge correctly identifies when the retrieved context is insufficient to answer a query and signals this uncertainty to the

operator either by refusing to answer or by hedging its response with an explicit caveat. A system with low self-knowledge confidently produces responses even when the retrieved context is absent or irrelevant, increasing the risk that the operator accepts a false answer as authoritative. Self-Knowledge is operationalized as the rate at which the system issues a refusal or an explicit uncertainty signal when all retrieved chunks have a relevance score of 0, as confirmed by the ground-truth annotation.

5.13.5 Faithfulness

Faithfulness is a composite measure of the alignment between the generated response and the retrieved source context. It is related to but distinct from Hallucination Rate: while Hallucination Rate flags any unsupported factual claim, Faithfulness additionally rewards responses that accurately preserve the semantic content of the retrieved chunks without distortion, omission of critical steps, or paraphrasing that changes the meaning of a procedural instruction. In the industrial documentation domain, faithfulness is especially critical for commands with precise parameter values, as even a minor reformulation — for example, substituting a numeric argument or reordering a procedural step — can render an otherwise correct response operationally invalid.

5.14 End-to-End Execution Correctness Metrics

The metrics defined in Sections 5.2–5.13 evaluate the RAG system at the level of individual retrieved chunks and generated responses. The three metrics defined in this section: Task Completion Accuracy (TCA), Goal Achievement (GA), and Instruction-to-Execution Fidelity (IEF) which evaluate the system at the level of complete task outcomes in the physical robotic environment. They are the highest-level indicators in the evaluation framework and represent the ultimate test of system utility in an industrial deployment.

5.14.1 Number of Steps

Number of Steps is an auxiliary descriptor that records the total number of discrete procedural steps contained in the ground-truth answer for each use case. It ranges from 1 (a single-action query such as UC: "Press [Motion Type]") to 6 (a multi-step sequence such as UC: copy and paste a MOVL instruction). The Number of Steps serves as a difficulty covariate in the analysis: use cases with more steps are inherently harder to answer completely, and TCA, GA, and IEF scores should be interpreted in light of task complexity. It is also used to normalise the TCA score, ensuring that a partially correct answer to a six-step procedure is scored proportionally rather than as a binary failure.

5.14.2 Task Completion Accuracy (TCA)

Task Completion Accuracy (TCA) measures the proportion of ground-truth procedural steps that were correctly present in the generated response, normalised by the total number of steps in the ground-truth annotation:

$$TCA = Correct\ Steps\ in\ Response\ /\ Total\ Steps\ in\ Ground\ Truth$$

TCA = 1.0 indicates that all procedural steps were correctly represented in the response; TCA = 0 indicates that no step was correctly answered. TCA is the primary end-to-end performance metric in the present evaluation and the basis for comparing the three RAG system versions. Its direct connection to physical task performance which a lower TCA implies more steps that the robot operator will be unable to execute correctly that makes it the most operationally meaningful metric in the framework.

5.14.3 Goal Achievement (GA)

Goal Achievement (GA) is a binary metric that indicates whether the operator successfully accomplished the specified task goal following the RAG system's guidance, as confirmed by direct observation of the robot pendant or controller state. Unlike TCA, which measures step-level coverage in the generated text, GA assesses the observable physical outcome: did the robot reach the intended configuration, perform the correct task, or execute the specified action? GA = 1 means that the physical system state after the interaction matches the target state outlined in the use-case specification. GA = 0 is assigned when the operator was unable to complete the task, made an error during execution, or required unsolicited expert intervention to achieve the goal.

5.14.4 Instruction-to-Execution Fidelity (IEF)

Instruction-to-Execution Fidelity (IEF) measures the degree of alignment between the sequence of steps recommended by the RAG system and the sequence of steps actually executed by the operator on the robot pendant. It captures a dimension that neither TCA nor GA addresses independently: TCA measures whether the correct steps were present in the response text, and GA measures whether the final outcome was correct, but neither measures whether the operator followed the system's instructions faithfully throughout the execution.

IEF is operationalised as the normalised edit distance between the recommended step sequence (extracted from the generated response) and the observed execution sequence (recorded by the evaluator), subtracted from 1 so that IEF = 1 denotes perfect alignment and IEF = 0 denotes complete divergence. High IEF combined with high GA provides strong

evidence that the system's instructions were both correct and sufficiently clear for faithful execution; high GA with low IEF suggests that the operator succeeded despite — rather than because of — the system's guidance.

5.15 The RAG Evaluation Landscape

The evaluation of Retrieval-Augmented Generation systems has progressed quickly since Lewis et al. [1] introduced the RAG paradigm in 2020. Originally, evaluation relied on downstream task accuracy on open-domain question–answering benchmarks, mainly SQuAD [8] and Natural Questions, measuring token-level Exact Match and F1 against annotated answers. This method, borrowed from reading comprehension assessment, merges retrieval quality and generation quality into one score, making it hard to identify whether failures are due to the retriever, the generator, or their interaction. As Gan et al. [4] point out in their detailed survey of RAG evaluation methods, using a single metric can be misleading: a system that maximizes retrieval precision at the cost of recall may seem strong on precision-focused benchmarks, while a system that enhances fluency but sacrifices factual accuracy might score well on token overlap but produce hallucinations.

The research community has since proposed a diverse set of evaluation programs that address this limitation from various angles. Gao et al. [29] categorize these approaches into three main groups. The first group consists of retriever-centric frameworks, which assess the document retrieval component in isolation using standard information retrieval metrics — Precision, Recall, nDCG@k, MRR — against fixed relevance judgments, following the metric definitions established by Manning et al. [3] and the position-sensitive nDCG formulation of Jarvelin and Kekalainen [15]. Notable examples include Dense Passage Retrieval evaluation protocols [9] and the U-NIAH benchmark [5], which expands retrieval evaluation to long-context needle-in-a-haystack scenarios.

The second family includes generator-centric frameworks that assess the language model's synthesis quality based on retrieved context, usually through faithfulness scoring, answer relevance, and hallucination detection. RAGAS [17], evaluated by Antal and Buza in a diploma-thesis RAG benchmark, is the most widely used open-source example of this family. It measures Faithfulness, Answer Relevance, Context Recall, and Context Precision using an LLM-as-judge approach that avoids the cost of human annotation but introduces systematic judge bias — a limitation directly noted by Antal and Buza [17] in their empirical comparison.

The third and most recent family comprises end-to-end frameworks, which jointly assess retrieval and generation components and their interaction. INSPECTORRAGET [6] is a notable example: it provides an introspection platform that traces attribution errors through the full pipeline, distinguishing retrieval failures from generation failures at the level of individual factual claims. Simon et al. [18] propose a case-study methodology for evaluating RAG systems in configuration-dependency validation contexts, demonstrating that domain-specific evaluation consistently reveals failure modes invisible to general-purpose benchmarks — a finding directly relevant to the industrial manufacturing context of the present thesis [7]. Applied AI Initiative [30] further highlight in their industrial white paper that RAG evaluation in production environments must account for operator interaction patterns and execution outcomes — dimensions that existing academic frameworks systematically neglect.

5.16 Why RAGChecker Was Chosen as the External Baseline and Why a Custom Program Was Built

The selection of an external evaluation baseline and the decision to develop a custom evaluation program were both driven by the specific requirements of the industrial deployment context. As Mishra et al. [12] demonstrate in their systematic framework for enterprise knowledge retrieval, LLM-generated metadata enrichment that the core contribution of the V2 system version which introduces evaluation demands that standard benchmarks are not designed to meet: in particular, the need to measure whether metadata-augmented chunks are ranked earlier and whether improved ranking translates into correct procedural execution by a human operator.

Three criteria guided the selection of RAGChecker as the primary external comparison framework rather than RAGAS [17], INSPECTORRAGET [6], or U-NIAH [5]. First, methodological alignment: RAGChecker computes chunk-level Precision, Recall, and F1 against annotated ground truth, the same metric definitions used in the present evaluation enabling direct numerical comparison without translation between incompatible scoring conventions. RAGAS, by contrast, uses LLM-as-judge scoring that is not directly comparable to reference-based token overlap metrics.

Second, diagnostic depth: RAGChecker's claim-level decomposition distinguishes retrieval failures from generation failures and explicitly models Noise Sensitivity and Context Utilization which dimensions also covered by Groups F and G of the evaluation database

schema. This makes RAGChecker the most structurally complete publicly available framework for comparison with the current multi-group evaluation. No other open-source framework covers both retrieval ranking quality and generator faithfulness at the same level of detail [4].

Third, domain applicability: RAGChecker was tested on a diverse set of knowledge-intensive QA tasks, including technical documentation queries, making it a more suitable external baseline for the Yaskawa industrial documentation domain than U-NIAH [5], which is designed for synthetic long-context retrieval, or Simon et al.'s methodology [18], specific to software configuration dependency validation.

The decision to develop a custom evaluation program alongside the external RAGChecker comparison was driven by four gaps identified through the above survey. First, no existing framework captures the user-efficiency dimension—the cognitive load on a novice operator interacting with the system in a live industrial environment—which is recorded through the Group C metrics (ET, WQ, LR) and directly impacts the operational value of the system in the Mind4LAB deployment context [30]. Second, no framework measures physical execution correctness—whether the generated procedural instructions are successfully carried out on the robot—which is captured through the Group H metrics (TCA, GA, IEF). Third, no existing framework is compatible with PrivateGPT's non-standard JSON export format, requiring a custom extraction pipeline. Fourth, the small, expert-annotated evaluation set (n = 65 queries) does not meet the scale requirements needed by LLM-judge frameworks like RAGAS [17], which depend on large query sets to stabilize judge variance.

5.17 RAGChecker: Architecture, Metrics, and Limitations

RAGChecker is a fine-grained diagnostic framework for evaluating Retrieval-Augmented Generation systems, introduced by Ru et al. at NeurIPS 2024 (arXiv:2408.08067). Its central contribution is the decomposition of both retrieved documents and generated responses into atomic factual claims using an LLM extractor, enabling attribution of evaluation scores to individual retrieved chunks and individual generated statements rather than to the response as a whole.

The RAGChecker pipeline operates in three stages. In the first stage, an LLM claim extractor decomposes each ground-truth answer and each generated response into a set of atomic claims — minimal, self-contained factual statements. In the second stage, an LLM claim checker evaluates entailment relationships between the claim sets: for each generated claim,

it determines whether the claim is entailed by the retrieved context (measuring faithfulness), entailed by the ground truth (measuring correctness), or neither (detecting hallucination). In the third stage, aggregate metric scores are computed from the entailment matrix across all queries in the evaluation set and It does not support higher Gemini Models (final supporting model is that “Gemini – 2.0 – flash – lite”)

5.18 RAG Evaluation Program

The evaluation program follows a linear four-stage pipeline architecture in which each stage consumes the output of the preceding stage and produces a precisely defined intermediate data representation. This design enforces a clean separation of concerns: the RAG system, the retrieval normalization logic, the ground-truth annotation, and the metric computation are each encapsulated in independent components that communicate exclusively via shared JSON-file interfaces. The Figure presents the complete architecture.

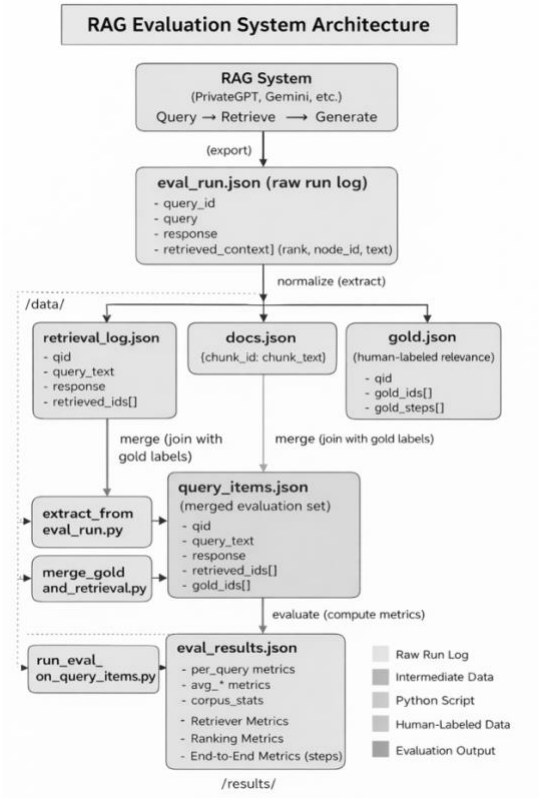

Figure 7: Architecture of RAG Evaluation Program

A fundamental architectural decision is the decoupling of data and computation at every stage boundary. The RAG system itself is entirely external to the evaluation pipeline; it only needs to export a standardized JSON log (eval_run.json) for the pipeline to operate. The ground-truth annotation (gold.json) is likewise maintained independently of the retrieval system, enabling re-annotation or expansion of the evaluation set without rerunning the RAG

system or repeating any earlier pipeline stage. The metric engine (rag_metrics.py) is a pure-function library with no file I/O of its own, operating exclusively on in-memory data structures passed by the runner scripts and returning a structured results dictionary.

Stage	Name	Script	Input	Output
A	Capture	(RAG system export)	PrivateGPT run	eval_run.json
B	Normalize	extract_from_eval_run.py	eval_run.json	retrieval_log.json docs.json
B	Normalize (alt.)	extract_retrieval_log.py	retrieved_contexts/*.json	retrieval_log.json docs.json
C	Label Join	merge_gold_and_retrieval.py	retrieval_log.json gold.json	query_items.json
D	Compute Metrics	run_eval_on_query_items.py	query_items.json docs.json (optional)	eval_results.json

Table 6: Four-stage pipeline reference: stage identifiers, names, associated scripts, inputs, and outputs. Stage B is an alternative extraction path for deployments that store retrieved contexts as individual per-query JSON files rather than a unified eval_run.

5.12 RAG Evaluation Program Analysis

The evaluation of Retrieval-Augmented Generation (RAG) systems poses a fundamental methodological challenge: standard scalar metrics such as mean Precision, mean Recall, and mean F1 Score, while computationally convenient, are inherently reductive. They collapse the full distributional character of system performance into a single point estimate, obscuring critical information about variance, multimodality, and the structural relationship between retrieval quality and downstream task accuracy.

To address this limitation, this section presents Kernel Density Estimation (KDE) as the main distribution analysis tool for comparing the proposed RAG Evaluation Program against the RAG Checker baseline. KDE allows for a non-parametric, assumption-free description of the full empirical score distribution across n = 65 evaluation queries, highlighting structural patterns such as multimodality, skewness, and probability mass concentration that scalar summaries often overlook.

5.12.1 Distribution Comparison: RAG Evaluation Program and RAG Checker

Kernel Density Estimation is a non-parametric technique for estimating the probability density function (PDF) of a random variable from a finite sample. Given a set of n

independent and identically distributed observations $X_1, X_2, ..., X_n$ drawn from an unknown distribution f(x), the KDE estimator is defined as:

$$\hat{f}_h(x) = (1 / nh) \times \Sigma_i K((x - X_i) / h)$$

where K() is a kernel function satisfying $\int K(u)du = 1$, and $h > 0$ is the bandwidth parameter controlling the smoothness of the resulting density estimate. In this work, the Gaussian kernel is employed:

$$K(u) = (1 / \sqrt{(2\pi)}) \times exp(-0.5\ u^2)$$

The bandwidth parameter h was set to 0.08 across all three metric distributions, selected via Silverman's rule of thumb as a baseline and subsequently validated by visual inspection to ensure that meaningful distributional features in particular, the bimodal structure present in both systems were preserved without over-smoothing.

The KDE analysis was conducted across n = 65 evaluation queries administered identically to both the proposed RAG Evaluation Program and the RAG Checker baseline framework (Ru et al., NeurIPS 2024). Three retrieval metrics were computed per query: Precision, Recall, and F1 Score. The empirical score arrays were passed directly to the KDE estimator without cleaning or imputation; queries returning zero scores on all metrics were retained in the analysis to preserve the full distributional character of system failure modes.

The KDE curves were computed using MATLAB's *ksdensity* function with a fixed Gaussian kernel bandwidth of h = 0.08. Separate figures were generated for each metric to facilitate focused distributional interpretation. Mean values for each system are reported in the legend of each figure and are additionally summarized in Tables.

Metric	**RAG Evaluation Program**	**RAG Checker**	**Δ (Gap)**
Precision	0,189	0,439	0,250
Recall	0,302	0,392	0,090
F1 Score	0,213	0,368	0,155

Table 7: Summary of Mean Retrieval Metrics

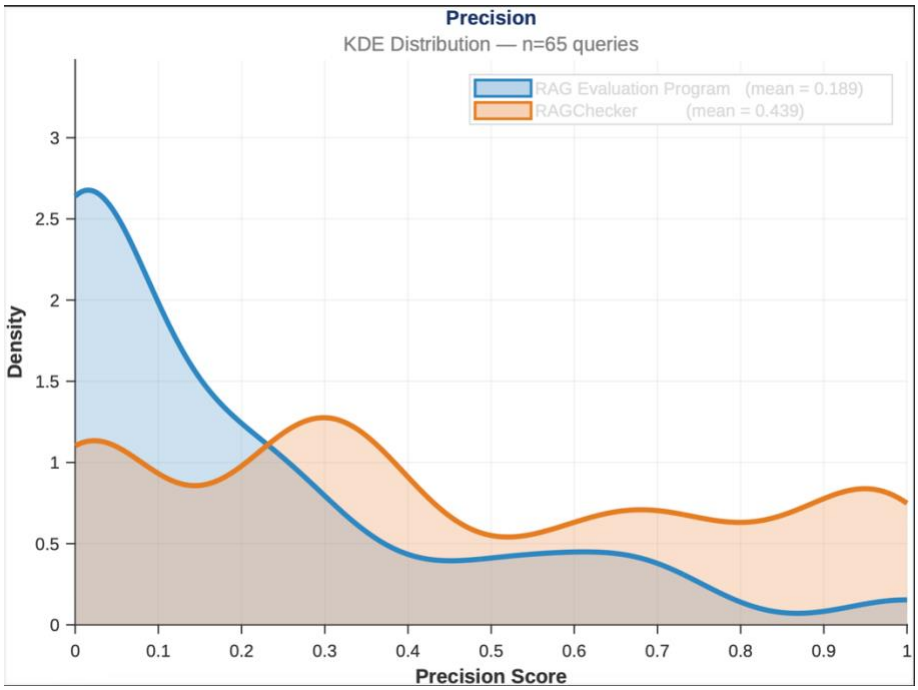

Figure 8: KDE of Precision Score distributions for My RAG System (blue) and RAGChecker (orange), n = 65 queries.

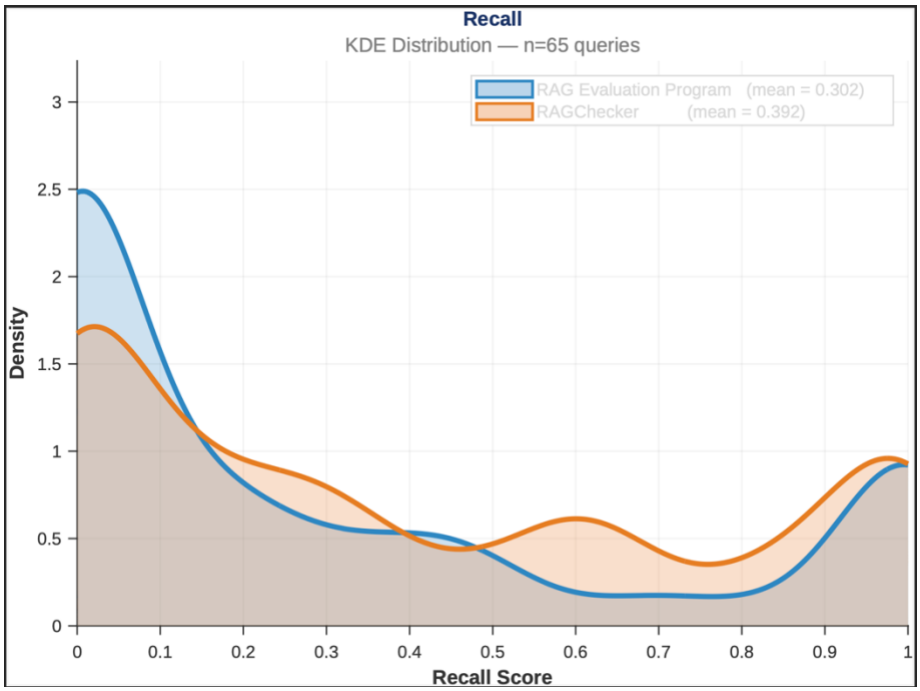

Figure 9: KDE of Recall Score distributions for My RAG System (blue) and RAGChecker (orange), n = 65 queries.

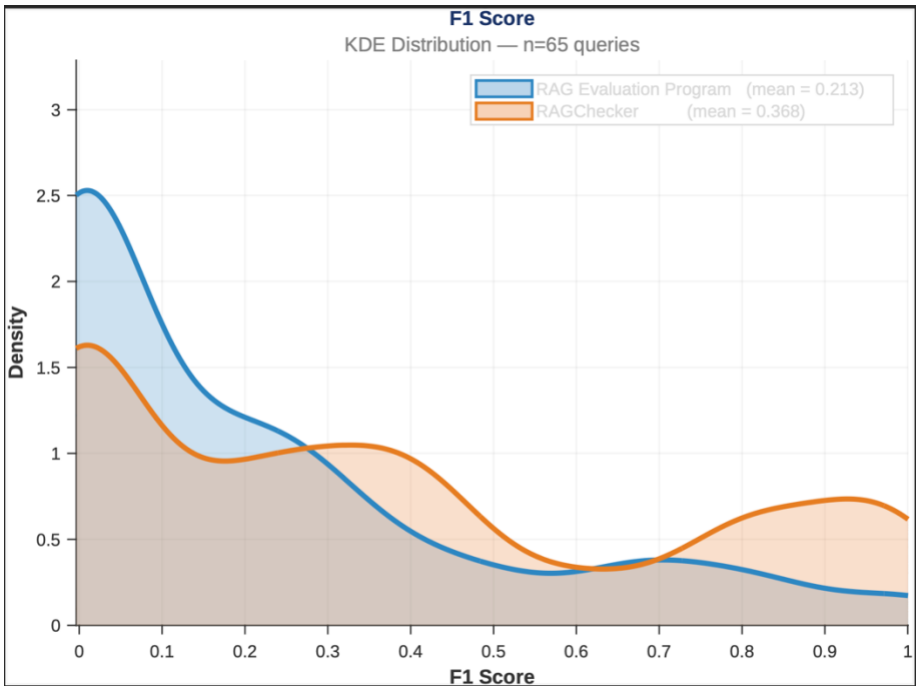

Figure 10: KDE of F1 Score distributions for My RAG System (blue) and RAGChecker (orange), n = 65 queries.

Although the distributional analysis shown in Figures 1, 2, and 1,2,3 reveals a mean performance gap between the proposed RAG Evaluation Program and the RAG Checker baseline across all three retrieval metrics- Precision (0.189 vs. 0.439), Recall (0.302 vs. 0.392), and F1 Score (0.213 vs. 0.368) a strictly numerical view of these differences might overstate their practical significance. Importantly, both systems display structurally similar distributional profiles: a dominant probability mass near zero, a secondary peak in the mid-to-high score range, and a pronounced right tail toward perfect retrieval scores. This shared bimodal structure suggests that both systems divide the 65-query evaluation set into two main retrieval regimes- successful high-fidelity retrieval and near-complete failure- and that the key difference is not qualitative but quantitative. Specifically, RAG Checker achieves a higher proportion of queries in the successful regime under standard chunking conditions, while the metadata-enriched pipeline of the RAG Evaluation Program currently routes more queries into the failure regime. This reframes the performance gap not as evidence of architectural inferiority but as an indication of an optimization frontier. The metadata enrichment strategy is fundamentally sound, and closing the gap requires improving metadata coverage and boundary detection robustness across the entire query distribution.

Chapter 6: Industrial Deployment at Mind4LAB

This chapter presents the industrial deployment evaluation of three RAG system versions (V0, V1, V2) conducted at the Mind4LAB research facility using a Yaskawa YRC1000 collaborative robot as the operational test platform. The chapter is organized into four sections: (1) the Mind4LAB operational environment and the Yaskawa collaborative robot platform, (2) the definition and structure of the six use cases comprising the 65-task evaluation set, (3) the intervention measurement framework, and (4) the quantitative intervention count and rate analysis across all versions and use cases.

6.1 YASKAWA Collaborative Robot

The deployment platform at Mind4LAB uses the Yaskawa YRC1000 robot controller system — a cutting-edge industrial robot controller designed to support Yaskawa Motoman manipulators in flexible, human-collaborative manufacturing cells. The YRC1000 system combines a high-performance motion controller, safety functions, and INFORM-language programming environment, allowing precise control of multi-axis robot manipulators for tasks like assembly, welding, material handling, and quality inspection.

The complete Yaskawa system deployed at Mind4LAB comprises four principal hardware components, illustrated in Figure 6.1: the Programming Pendant (handheld operator interface for job creation, parameter setting, and system configuration), the YRC1000 Controller (the central motion control and computational unit), the Manipulator (the multi-axis robot arm executing physical operations), and the Manipulator Cable (high-flex power and signal interconnect between controller and robot arm).

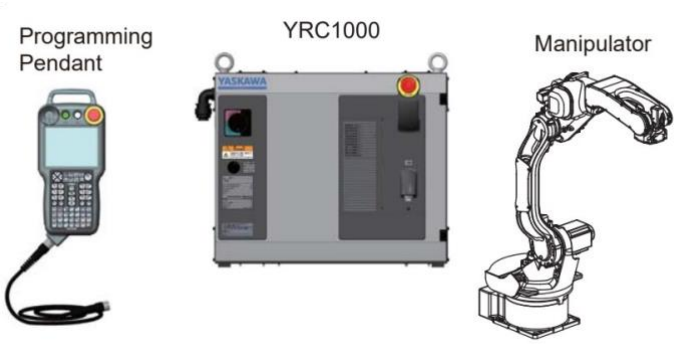

Figure 11: Yaskawa YRC1000 collaborative robot system — Programming Pendant, YRC1000 Controller, Manipulator.

6.2 Use Case Definition and Task Structure

The deployment evaluation is structured around six use cases (UC1–UC6) that collectively represent the principal knowledge domains encountered in day-to-day Yaskawa

collaborative robot operation at Mind4LAB. The 65 evaluation tasks are distributed across these use cases in proportions reflecting their operational frequency in the facility. Table 6.1 provides an overview of all six use cases.

UC	# Tasks	Use Case Name	Primary Focus
UC1	7	Robot Programming Fundamentals	Job creation, motion types, position management and basic INFORM operations
UC2	17	Automated Pick-and-Place	Full pick-and-place workflow with gripper function calls, coordinate systems and trajectory planning
UC3	11	Advanced Gripper Programming	OnRobot gripper control via INFORM: variables, loops, conditionals and low-level register commands
UC4	8	TCP Configuration	Tool Centre Point definition, calibration, tool load parameters and security mode management
UC5	5	Web Client TCP Setup	Remote TCP configuration via YRC1000 web interface: login, model selection and value input
UC6	17	MG10 Gripper Integration	End-to-end pick-and-place with MG10 gripper: data transfer, job creation, OR functions and argument configuration

Table 8: : Overview of the six use cases — task counts, names and primary knowledge domains

6.2.1 UC1 — Robot Programming Fundamentals

UC1 covers the fundamental programming tasks that every Yaskawa robot operator must learn when working with the YRC1000 controller and INFORM language. This use case is aimed at operators with beginner to intermediate skills who need to create, edit, and manage basic robot job programs. The seven tasks encompass the entire lifecycle of a basic robot program: from job creation and motion type selection to instruction insertion, position registration, position editing, speed setup, and operation deletion.

The knowledge needed for UC1 mainly comes from the Yaskawa YRC1000 Operator's Manual and INFORM Language Reference. It tests the RAG system's ability to provide accurate procedural guidance on controller operations that involve precise menu navigation, parameter naming, and instruction syntax. A correct RAG response for UC1 tasks must not only identify the relevant concept but also give the specific step-by-step instructions for the YRC1000 pendant interface.

Task ID	Category	Task Description
UC 1.1	Job Management	Create a new job on the Yaskawa YRC1000 controller
UC 1.2	Motion Control	Choose the right motion type (MOVJ/MOVL/MOVC) and change between them
UC 1.3	Programming	Insert a Move instruction into an existing job program
UC 1.4	Position Reference	Create a reference point and register it in the controller
UC 1.5	Position Edit	Change an existing position stored in the job
UC 1.6	Speed Setting	Change the speed parameter of a MOVJ instruction
UC 1.7	Job Edit	Delete operations (instructions or steps) from a job

Table 9: UC1 — Robot Programming Fundamentals task definitions

UC1 Performance Analysis: V0/V1/V2

Latency:

UC1	V0	V1	V2
UC 1.1	14864,24	5212,83	3694,68
UC 1.2	56028,39	6877,38	3790,26
UC 1.3	31334,9	7651,61	3308,19
UC 1.4	30012,55	7587,13	3308,19
UC 1.5	49678,26	17744,96	8784,83
UC 1.6	52468,31	9807,38	4928,55
UC 1.7	48537,06	3950,3	3858,72

Table 10:Latency(ms) of UC1

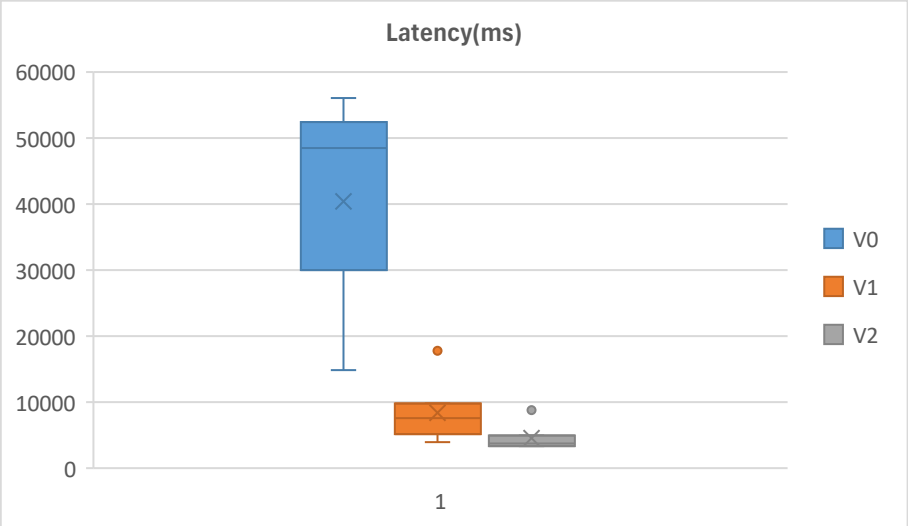

Figure 12:Box plot of response latency (ms) for UC1 across V0, V1, and V2.

Figure 12 shows the distribution of response latency in milliseconds across the seven queries of UC1 for each system version. Latency is defined as the time elapsed between submitting a query and receiving a complete generated response, measured per query and averaged across each use case. The box plot format is especially useful here because it reveals not

only the average response time but also its consistency — a crucial factor in a live industrial environment where unpredictable delays can disrupt operator workflow.

V0 displays the highest and most variable latency. The average response time is 40,418 ms (about 40 seconds), with a median above 48,000 ms and an interquartile range over 22,000 ms. The whiskers extend from roughly 14,000 ms to 56,000 ms, showing that no query is answered in less than 14 seconds and that the worst cases approach one minute. This behavior results from the lack of optimized retrieval indexing in the baseline: V0 performs a dense search over the entire corpus for each query, and variable retrieval depth directly causes fluctuating response times. For a novice operator, a system that can respond in 15 or 56 seconds can be considered unreliable, regardless of answer quality.

V1 cuts the average latency by 79.2%, down to 8,405 ms, by using semantic chunking and hybrid retrieval, which limits the candidate set sent to the inference engine. The box is noticeably smaller and lower, indicating that the distribution is both stabilized and reduced. An outlier at 17,745 ms corresponds to UC1.5 — the most complex multi-step query — showing that V1 struggles with high semantic ambiguity even as it performs well on simpler requests.

V2 achieves the lowest and most consistent response times, with an average of 4,525 ms — an 88.8% reduction compared to V0 — and a tight interquartile range that suggests near-uniform response times across all query types. The single outlier at 8,785 ms for UC1.5 is half the size of the V1 outlier for the same query. This improvement is directly due to the metadata filtering method: by pre-annotating chunks with procedure type, component ID, and step sequence, the retriever narrows down the candidate set at the metadata index level before performing dense search, greatly reducing the context length that impacts generation latency.

F1 Score:

UC1	V0	V1	V2
UC 1.1	0,74	0,82	0,78
UC 1.2	0,33	0,60	0,43
UC 1.3	0,20	0,18	0,18
UC 1.4	0,35	0,17	0,92
UC 1.5	0,15	0,00	0,36
UC 1.6	0,25	0,19	0,33
UC 1.7	0,28	0,60	0,29

Table 11: F1 Score of UC1

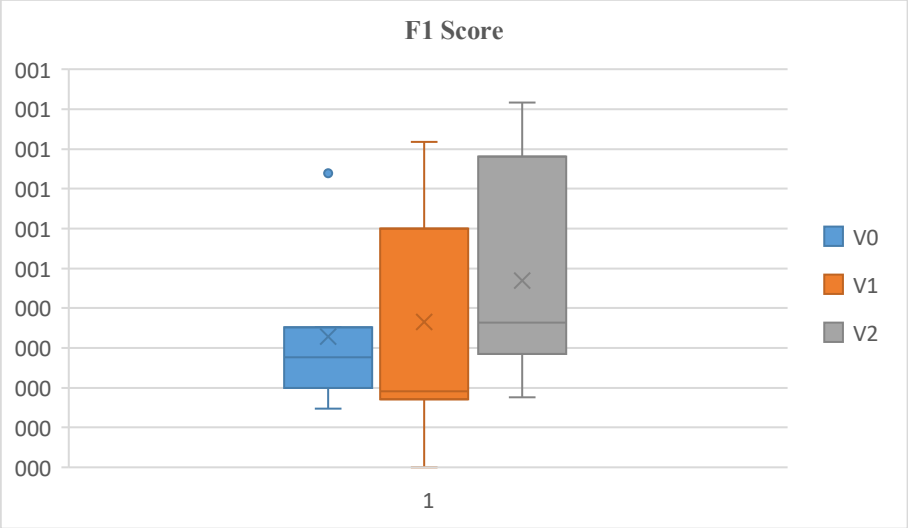

Figure 13: Box plot of end-to-end F1 score for UC1 (7 queries) across V0, V1, and V2. F1 is the harmonic mean of Precision and Recall

Figure Y shows how the end-to-end F1 score varies across the seven queries of UC1. The F1 score is chosen as the main metric for measuring retrieval quality because, as the harmonic mean of Precision and Recall, it penalizes systems that either retrieve too few relevant chunks (low Recall) or too many irrelevant ones (low Precision). This dual sensitivity makes F1 more informative than either metric alone: a system that achieves perfect Recall by retrieving the entire corpus scores low F1, just as a system with perfect Precision by returning only one chunk does. The average F1 scores are 0.33 for V0, 0.37 for V1, and 0.47 for V2.

V0 produces a tight, low distribution with a mean F1 of 0.33 and a narrow interquartile range focused between approximately 0.20 and 0.34. The compactness of the V0 box is not a positive sign — it indicates consistently weak retrieval quality across all seven queries. The fixed-size chunking strategy systematically crosses procedure boundaries, resulting in both low Precision (the chunk contains irrelevant steps from neighboring procedures) and low

Recall (relevant steps are split across multiple chunks, with only some retrieved). The outcome is a system that performs similarly poorly regardless of query type.

V1 produces a broader distribution with an average F1 score of 0.37 and a significantly higher upper quartile, along with a single outlier at 0.75. The expansion of the interquartile range indicates that semantic chunking greatly improves F1 scores on queries where procedure boundaries match semantic breakpoints in the source text, but offers limited gains on queries where relevant information is spread across multiple semantic units. The outlier at 0.75 demonstrates that V1 can achieve high retrieval quality when the query intent and document structure are well aligned.

V2 achieves the highest mean F1 of 0.47, a 42% relative improvement over V0 with an upper quartile nearing 0.80. The improvement is most evident in the upper half of the distribution, indicating that metadata enrichment enhances performance on well-structured single-procedure queries while partly closing the gap on complex ones. The presence of a non-zero lower tail confirms that metadata enrichment does not completely eliminate retrieval failures, highlighting metadata coverage and boundary detection robustness as the main focus areas for future system improvements.

Generator and Synthesis Metrics:

Metric	**V0**	**V1**	**V2**
Context Utilization	0,703	0,629	0,817
Noise Sens. Relevant	0,477	0,476	0,475
Noise Sens. Irrelevant	0,024	0,000	0,020
Hallucination	0,085	0,198	0,037
Self-Knowledge	0,000	0,000	0,099
Faithfulness	0,915	0,802	0,863

Table 12: Generator and Synthesis Metrics' Average Values of UC1

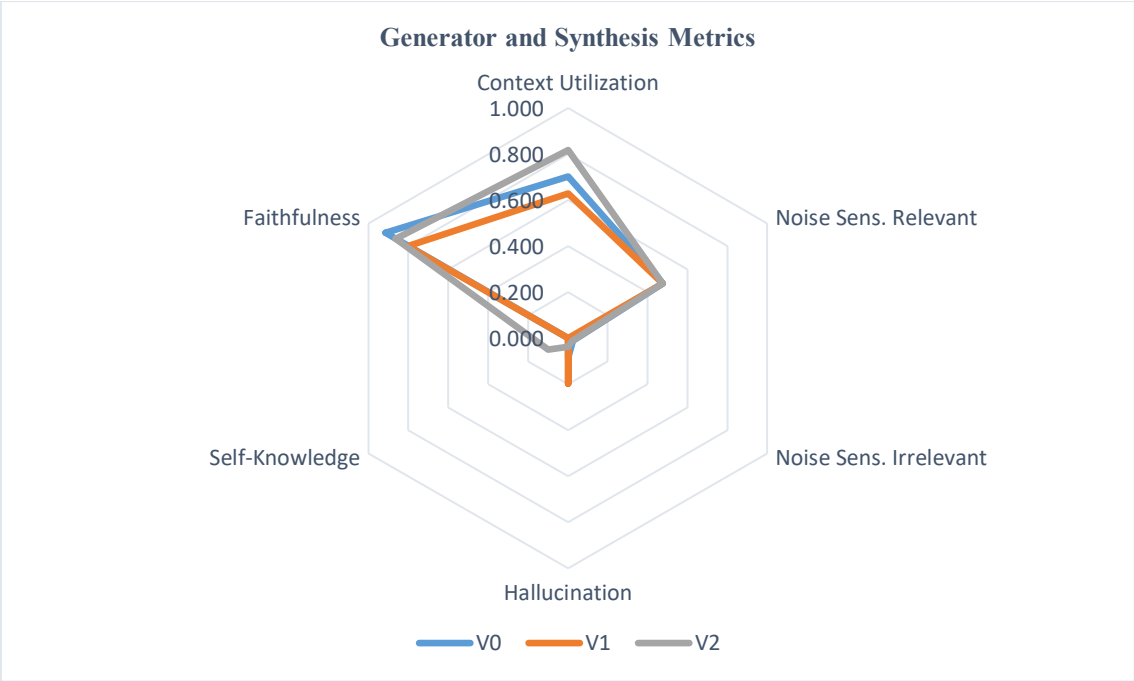

Figure 14: Radar chart of Group G generator and synthesis metrics averaged across all seven UC1 queries. Axes: Context Utilization, Noise Sensitivity (Relevant), Noise Sensitivity (Irrelevant), Hallucination, Self-Knowledge, Faithfulness. Lower is better for Nois

Figure 14 presents the radar chart of Group G generator and synthesis metrics for UC1, averaged across all seven queries for each system version. Unlike the scalar box plots in Figures X and Y, the six-axis radar profile provides a simultaneous view of qualitatively distinct dimensions of generator behaviour, making it possible to identify trade-offs and failure patterns invisible in a single-number summary. The six axes are: Context Utilization, Noise Sensitivity (Relevant), Noise Sensitivity (Irrelevant), Hallucination, Self-Knowledge, and Faithfulness. For Noise Sensitivity and Hallucination, lower values indicate better performance; for all remaining axes, higher values indicate better performance.

V2 leads on the three most safety-critical axes. Its Context Utilization of 0.817 is the highest among all versions, confirming that the language model primarily relies on retrieved evidence when metadata-enriched chunks are provided. This reduces the generator's dependence on potentially inaccurate parametric memory. Its Hallucination Rate of 0.037 shows a five-fold reduction compared to V1 (0.198) and a two-fold reduction relative to V0 (0.085)—making V2 the safest operational version, as fewer hallucinated command arguments or procedural steps reach the operator. V2 is also the only version with a non-zero Self-Knowledge score of 0.099, indicating it is the only one that occasionally and correctly signals uncertainty when its retrieved context is insufficient. Although modest in absolute terms, this is a significant improvement: an operator who receives an explicit

uncertainty signal can look for an alternative source, whereas one who receives a confident hallucination may act on incorrect guidance, which could directly impact robot safety.

V1 provides a key result on the Noise Sensitivity Irrelevant axis, recording a score of 0.000 — the only zero value across all versions and all six Group G metrics. This confirms that the hybrid BM25-plus-vector retrieval of V1 never returns a completely off-topic chunk for any UC1 query. The BM25 lexical component acts as a topic gate, preventing semantically close but procedurally irrelevant chunks from entering the context window and directly lowering the generator's hallucination risk at the source.

The one counterintuitive result is Faithfulness, where V0 records the highest score at 0.915, surpassing V1 (0.802) and V2 (0.863). This is due to V0's sparse retrieval: fixed-size chunking produces fewer and shorter chunks, giving the generator less material to contradict or paraphrase. V0's high faithfulness is therefore due to *impoverishment* rather than grounding. As the context becomes richer in V1 and V2, the generator legitimately paraphrases or reorders steps, lowering the formal faithfulness score while enhancing the operational completeness of the response — as shown by the higher TCA values in Section 4.

Finally, Noise Sensitivity Relevant is essentially flat across all three versions (V0: 0.477, V1: 0.476, V2: 0.475), indicating that no version suppresses the distraction effect of topically adjacent but non-answering chunks. This finding motivates future work on post-retrieval re-ranking and query-focused summarisation as complementary filtering mechanisms beyond the scope of the present thesis.

Task Completion Accuracy(TCA):

UC1	V0	V1	V2
UC 1.1	0,80	1,00	1,00
UC 1.2	0,00	0,50	0,50
UC 1.3	0,20	0,60	0,40
UC 1.4	0,33	0,33	1,00
UC 1.5	0,00	0,00	0,67
UC 1.6	0,22	0,22	0,33
UC 1.7	0,50	0,50	0,50

Table 13: Task Completion Accuracy(TCA) of UC1

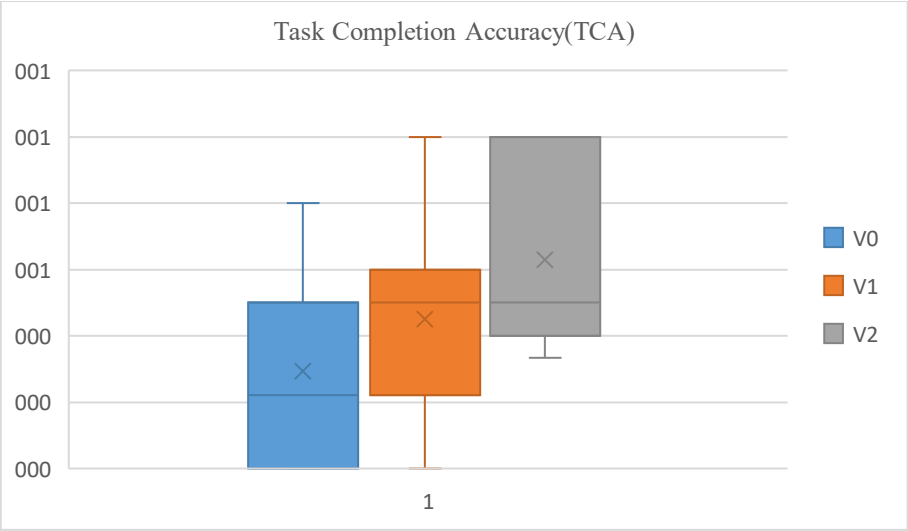

Figure 15: Box plot of Task Completion Accuracy (TCA) for UC1 across V0, V1, and V2.

Figure 15 presents the distribution of Task Completion Accuracy (TCA) across the seven queries of UC1. TCA is the highest-level performance indicator in the evaluation framework: it measures the proportion of ground-truth procedural steps correctly present in the generated response, normalised by the total number of steps in the annotated ground truth. A TCA of 1.0 indicates that every required step was correctly communicated; a TCA of 0 indicates that no step was correctly covered. Unlike F1, which evaluates retrieval quality at the chunk level, TCA evaluates the operational usefulness of the final generated response the dimension with the most direct consequence for robot programming correctness and operator safety in the Mind4LAB deployment.

V0 achieves a mean TCA of 0.29, with the box concentrated below 0.50 and a median near 0.22. Under the baseline configuration, the generated response covers fewer than one-third of the required procedural steps on average. The narrow V0 box reflects consistently poor performance: the fixed-size chunking strategy systematically misses procedure boundaries, so the generator consistently receives incomplete context and produces correspondingly incomplete step coverage. For an operator following the system's guidance, a mean TCA of 0.29 implies that more than 70% of the required steps must be sourced from elsewhere, rendering the system of limited operational value for UC1.

V1 raises the mean TCA to 0.45, a 55% increase compared to V0, but introduces significant variability: the whiskers span the entire range from 0.00 to 1.00, and the wide interquartile range indicates high query dependence. For simple single-procedure queries like UC1.1 (create a new job, TCA = 1.00) and UC1.7 (delete a move instruction, TCA = 0.50), V1's semantic chunking correctly identifies and retrieves the entire procedure. However, on

complex multi-step queries like UC1.5 (modify existing position with multiple coordinate systems, TCA = 0.00), V1 fails completely. This bimodal pattern – success on simple queries and failure on complex ones – reflects a system whose retrieval strategy improves coverage for well-structured procedures but cannot yet handle queries that require synthesis across multiple procedure chunks.

V2 achieves the highest mean TCA of 0.63 — a 117% improvement relative to V0 and a 40% improvement relative to V1 — with both the upper quartile and maximum reaching 1.00. The V2 box plot shows the most favorable shape of the three versions: a high median, a compact lower quartile, and a strong upper tail, indicating that the metadata-enriched pipeline consistently delivers high step coverage on structured queries while partially recovering on complex ones. The remaining lower tail highlights the ongoing optimization frontier: cross-procedure synthesis queries where even metadata-enriched retrieval does not yet guarantee complete step coverage, and where future work on multi-hop retrieval or chain-of-thought prompting may yield further gains.

6.2.2 UC2 — Automated Pick-and-Place

UC2 is the most operationally complex use case in the evaluation set, comprising 17 tasks that together constitute a complete automated pick-and-place job program using the On Robot RG series gripper. This use case simulates a realistic robot programming scenario encountered frequently at Mind4LAB: an operator must build a full pick-and-place routine from scratch, integrating safety checks, coordinate system selection, gripper function calls with correctly configured arguments, and multi-point trajectory planning.

The 17 tasks are sequenced to reflect the logical order of program construction: beginning with safety assurance and job creation, progressing through coordinate and motion type selection, gripper initialization, function call configuration with four distinct argument types, trajectory definition (approach, pick, move to placement, place, lift, return), and culminating in correct placement point specification. This sequential structure means that errors in early tasks such as incorrect coordinate system selection or misconfigured gripper arguments can propagate to affect the execution of later tasks, making UC2 a particularly demanding test of the RAG system's ability to provide coherent, contextually accurate guidance across a multi-step programming workflow.

Task ID	Category	Task Description

UC 2.1	Safety	Safety assurance — verify safe operating conditions before execution
UC 2.2	Job Management	Create a new job for the pick-and-place routine
UC 2.3	Coordinates	Choose the right coordinate system (Joint / Base / Tool / User)
UC 2.4	Initialisation	Main initialisation — set variables and system state at job start
UC 2.5	Gripper Control	Reset gripper to default open/closed state
UC 2.6	Motion Control	Choose the right motion type for pick-and-place trajectory
UC 2.7	Motion	Move to the safe area before approaching the pick target
UC 2.8	Function Call	Set function call instruction for the pick operation
UC 2.9	Arguments	Set the first argument and understand its meaning in the pick function
UC 2.10	Arguments	Set the second argument and understand its meaning
UC 2.11	Arguments	Set the third argument and understand its meaning
UC 2.12	Arguments	Set the fourth argument and understand its meaning
UC 2.13	Motion	Move to the placement point with correct trajectory
UC 2.14	Function Call	Set function call instruction for the place operation
UC 2.15	Arguments	Set the right argument for the place function
UC 2.16	Motion	Lift up — retract end-effector after placement
UC 2.17	Motion	Return to the starting/home position

Table 14: UC2 — Automated Pick-and-Place task definitions

UC2 Performance: V0, V1, V2

Latency:

UC2	V0	V1	V2
UC2.1	32957,18	8042,76	6408,38
UC2.2	22826,85	5249,97	3637,88
UC2.3	34859,82	4747,5	5969,98
UC2.4	25389,13	5996,88	3027,06
UC2.5	30445,41	5420,78	3568,91
UC2.6	22921,11	4139,63	4738,7
UC2.7	25327,31	7352,3	3925,58
UC2.8	27664,99	4271,26	13975,86
UC2.9	40376,02	12712,36	9894,33
UC2.10	51066,91	5835,7	6294,13
UC2.11	64046,58	10833,52	11169,72
UC2.12	89870,31	7309,89	4131,46
UC2.13	58215,66	4154,42	3130,42
UC2.14	82822,12	8992,39	5527,38
UC2.15	17747,54	8012,4	4218,87
UC2.16	31484,84	6347,89	2995,78
UC2.17	26688,42	8793,46	9004,21

Table 15: Latency of UC2

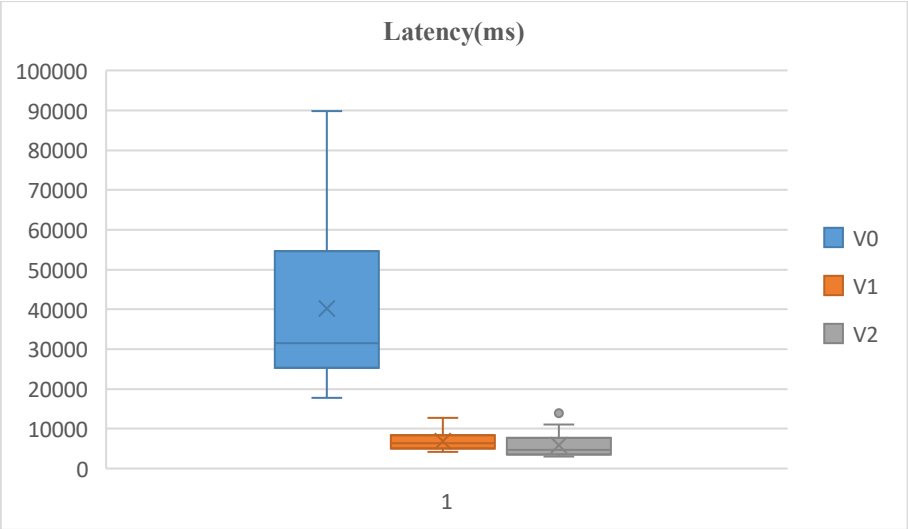

Figure 16: Box plot of response latency (ms) for UC2 (17 queries) across V0, V1, and V2.

Figure 16 shows the latency distribution across the seventeen queries of UC2, a significantly larger and more complex use case than UC1, encompassing a wider range of robot programming tasks such as trajectory planning, speed adjustment, position editing, and instruction management. The larger query set offers a more statistically reliable view of version behavior under different operational conditions.

V0 again exhibits the highest and most variable latency, with a mean of around 40,209 ms and a broad box extending from about 22,000 ms to 58,000 ms. The upper whisker reaches nearly 90,000 ms, corresponding to UC2.12 (89,870 ms) a complex multi-instruction query requiring retrieval across multiple procedure sections. The lower whisker extends to approximately 17,000 ms (UC2.15), the simplest query in the set. This nearly 72,000 ms range between the minimum and maximum confirms that V0's brute-force retrieval method results in inherently unpredictable response times at scale: as the dataset and query complexity increase, so does the variability, making V0 unsuitable for deployment in time-critical manufacturing settings.

V1 achieves an average latency of about 7,199 ms ,at 82.1% reduction compared to V0, with a compact box and whiskers mostly below 13,000 ms. Two outliers are visible at approximately 10,833 ms (UC2.11) and 12,712 ms (UC2.9), both linked to multi-step queries that require synthesis across motion types and position modification procedures. Compared to the UC1 results, V1's latency distribution is slightly higher in absolute terms, consistent with the increased number of queries and the more complex procedural content

of UC2, but the relative reduction from V0 is maintained, and the distribution remains notably tighter.

V2 achieves an average latency of about 5,977 ms, representing an 85.1% reduction from V0 with the tightest distribution among the three versions. A single outlier at 13,975 ms (UC2.8) marks the only query where V2's metadata filtering did not fully contain the retrieval cost, likely due to ambiguous metadata for that specific instruction type. Notably, the difference between V1 and V2 is narrower for UC2 than for UC1 (around 1,200 ms versus approximately 3,900 ms), indicating that V2's metadata advantage slightly decreases as use case complexity increases and more cross-procedure retrieval is required.

F1 Score:

UC2	**V0**	**V1**	**V2**
UC2.1	0,1379	0,37974684	0,9787234
UC2.2	0,8235	0,87804878	0,81818182
UC2.3	0,5357	0,08695652	0
UC2.4	0	0	0
UC2.5	0	0	0,46
UC2.6	0,5	0,70588235	0,825
UC2.7	0	0	0
UC2.8	0	0	0
UC2.9	0	0,10526316	0,11764706
UC2.10	0	0	0,16438356
UC2.11	0	0	0,14285714
UC2.12	0	0	0
UC2.13	0,4444	0	0
UC2.14	0,169	0,16	0
UC2.15	0	0,11428571	0
UC2.16	0,3077	0	0,44444444
UC2.17	0	0	0

Table 16: F1 Score of UC2

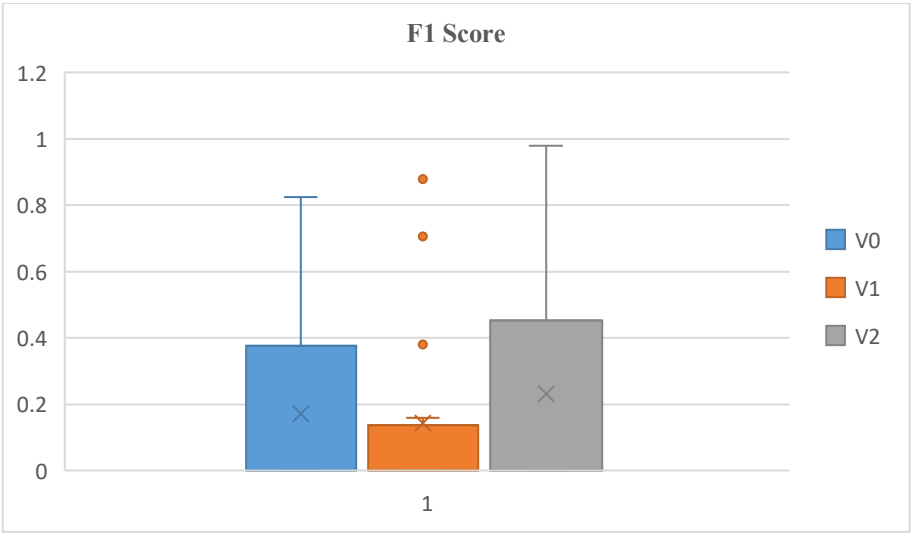

Figure 17: Box plot of end-to-end F1 score for UC2 (17 queries) across V0, V1, and V2. F1 is the harmonic mean of Precision and Recall.

Figure 17 shows the F1 score distribution for UC2. The overall F1 scores are notably lower than those for UC1 (UC1 averages: V0=0.33, V1=0.37, V2=0.47), reflecting the increased complexity and variety of procedures in UC2's seventeen queries. Many UC2 queries require the system to retrieve and synthesize information from multiple non-contiguous sections of the Yaskawa robot manual, a task that challenges all three retrieval architectures and results in a high number of zero F1 scores, visible in the box plots as boxes where the lower quartile reaches or nears zero.

V0 achieves an average F1 score of 0.17 with most values clustered in the lower range, along with two high outliers at 0.8235 (UC2.2) and 0.5357 (UC2.3). These outliers correspond to the two simplest queries in UC2 — well-structured single-procedure requests where fixed-size chunking effectively keeps the relevant content within one retrievable chunk. Most V0 scores are zero or near-zero, confirming that the baseline retrieval method consistently fails on the multi-step, cross-section queries that are common in UC2.

V1 records the lowest average F1 score of 0.14, performing slightly worse than V0 overall. This is a notable and surprising finding: semantic chunking, which significantly improved F1 in UC1, offers no benefit and slightly negative results in UC2. Three high outliers are visible at 0.8780 (UC2.2), 0.7059 (UC2.6), and 0.3797 (UC2.1), while the remaining fourteen queries score near zero. This bimodal pattern strong performance on a small subset and near-complete failure on most indicates that V1's semantic boundary detection is sensitive to the structural regularity of the source document. Variations in procedural formats in UC2 challenge the semantic segmentation algorithm, resulting in chunks that often do not match query intent for most task types.

V2 achieves the highest mean F1 score of 0.23, representing a 35% improvement over V0 and a 64% improvement over V1. It features three high outliers at 0.9787 (UC2.1), 0.8182 (UC2.2), and 0.8250 (UC2.6), along with a wider upper quartile than the other two versions. The metadata enrichment offers its most noticeable advantage on queries where the procedure type and component metadata fields uniquely identify the relevant section, enabling the retriever to surface the correct chunk even when semantic similarity by itself is not enough. The persistence of many near-zero scores in the remaining queries confirms that UC2 is a considerably more difficult retrieval challenge than UC1 and that metadata enrichment alone cannot resolve all cross-section synthesis issues queries.

Generator and Synthesis Metrics:

Metric	V0	V1	V2
Context Utilization	0,36	0,29	0,29
Noise Sens. Relevant	0,17	0,23	0,19
Noise Sens. Irrelevant	0,27	0,28	0,18
Hallucination	0,24	0,24	0,34
Self-Knowledge	0,01	0,01	0,08
Faithfulness	0,69	0,70	0,58

Table 17: Generator and Synthesis Metrics' Average Values of UC1

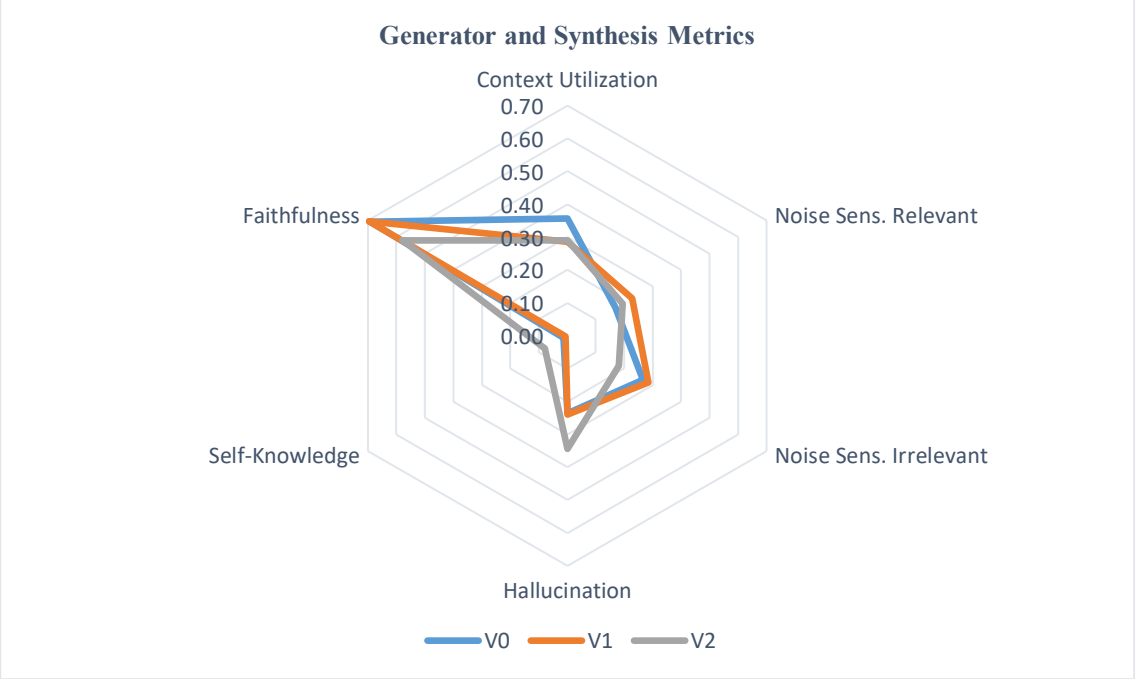

Figure 18: Radar chart of Group G generator and synthesis metrics averaged across all 17 UC2 queries.

Figure 18 shows the radar chart of Group G generator and synthesis metrics for UC2, averaged across all seventeen queries. The UC2 radar profile differs significantly from UC1 in both shape and version separation: the three polygons are more closely grouped, and overall scores are lower on most axes, indicating the greater difficulty of the use case and the reduced ability of any retrieval strategy to consistently surface complete, relevant context for multi-procedure queries.

A notable inversion occurs on the Context Utilization axis: V0 leads at 0.36, while V1 and V2 are tied at 0.29. In UC1, V2 had a significant advantage on this axis (0.817 versus 0.703 for V0). The UC2 reversal suggests that when queries need cross-section retrieval, the metadata-enriched chunks retrieved by V2 may be more narrowly focused on a specific procedure section in depth, while the baseline retrieves broader chunks that, although lower in precision, contain a larger portion of the context the language model uses. This finding adds nuance to the UC1 interpretation: metadata enrichment enhances context utilization when queries are well-defined but may decrease it when queries require broad synthesis. The Hallucination axis reveals another UC2-specific pattern:

V2 shows the highest hallucination rate at 0.34, compared to 0.24 for both V0 and V1. This is contrary to the UC1 result, where V2 had the lowest hallucination rate at 0.037. The reason lies in the retrieval failure pattern seen in the F1 analysis: when V2's metadata filters retrieve very specific but incomplete chunks for complex cross-section queries, the language model—when faced with partial context—is more likely to fill gaps using parametric memory than when it receives broader, less precise context. Metadata-focused retrieval seems to increase hallucination risk especially when the metadata does not uniquely identify all relevant sections of a multi-part answer. Faithfulness remains the strongest axis for all three versions (V0=0.69, V1=0.70, V2=0.58), consistent with the UC1 pattern. V1 marginally leads on Faithfulness for UC2, and V2's lower score of 0.58 is again consistent with a richer retrieved context that introduces more legitimate paraphrasing.

Self-Knowledge remains low for V0 and V1 (0.01 each) but rises slightly for V2 (0.08), maintaining the pattern observed in UC1 where metadata enrichment is the only configuration that produces meaningful uncertainty acknowledgement. Noise Sensitivity Irrelevant is lowest for V2 (0.18) compared to V0 (0.27) and V1 (0.28), suggesting that even in the more complex UC2 context, metadata pre-filtering reduces the proportion of completely off-topic chunks reaching the language model. Noise Sensitivity Relevant is

lowest for V0 (0.17) and highest for V1 (0.23), indicating that semantic chunking in V1 surfaces more topically adjacent but non-answering content a retrieval side effect that is more pronounced in the diverse procedural landscape of UC2 than in the more structured UC1.

Task Completion Accuracy (TCA):

UC2	**V0**	**V1**	**V2**
UC2.1	0,10	0,30	1,00
UC2.2	0,75	1,00	1,00
UC2.3	1,00	0,50	0,50
UC2.4	0,00	0,00	0,00
UC2.5	0,00	0,20	0,60
UC2.6	1,00	1,00	1,00
UC2.7	0,00	0,00	0,00
UC2.8	0,00	0,00	0,00
UC2.9	0,00	0,00	0,25
UC2.10	0,00	0,00	0,20
UC2.11	0,00	0,00	0,00
UC2.12	0,00	0,00	0,00
UC2.13	0,33	0,00	0,00
UC2.14	0,00	0,29	0,29
UC2.15	0,00	0,00	0,00
UC2.16	0,00	0,00	0,33
UC2.17	0,00	0,00	0,17

Table 18: Task Completion Accuracy of UC2

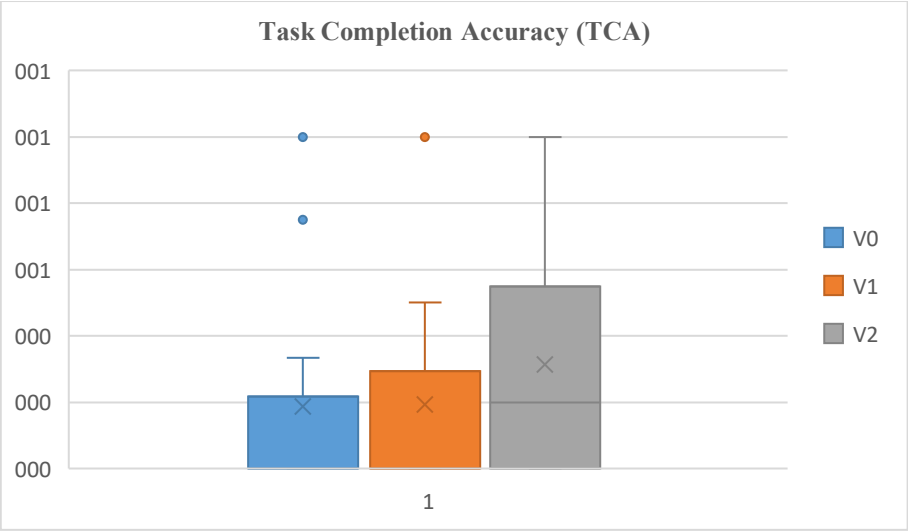

Figure 19: Box plot of Task Completion Accuracy (TCA) for UC2 across V0, V1, and V2.

Figure 19 presents the TCA distribution for UC2. The mean values — V0: 0.187, V1: 0.193, V2: 0.314 — are substantially lower than those recorded for UC1 (V0: 0.29, V1: 0.45, V2: 0.63), confirming that UC2 represents a significantly more demanding evaluation scenario. With seventeen queries spanning a wide range of robot programming tasks, many of which require multi-section procedural synthesis, no version achieves the high step coverage rates observed in the simpler seven-query UC1 context.

V0 achieves a mean TCA of 0.187, with the box concentrated between 0.00 and approximately 0.33 and two outliers at 0.75 (UC2.2) and 1.00 (UC2.3). The majority of queries — twelve out of seventeen — produce a TCA of 0.00, meaning the generated response covers none of the required procedural steps for those tasks. This high rate of complete failure reflects the systematic inability of fixed-size chunking to retrieve complete multi-step procedures in a diverse use case: when the relevant steps span multiple fixed-size chunks, the baseline retrieves at most one, leaving the generator with insufficient context to produce a usable response.

V1 achieves a marginally higher mean TCA of 0.193 — an improvement of only 3% relative to V0. The V1 box shows a slightly wider upper tail and one outlier at 1.00 (UC2.2), but the majority of queries remain at or near zero. This near-flat improvement from V0 to V1 in UC2 contrasts sharply with the 55% improvement observed in UC1, confirming that V1's semantic chunking strategy provides diminishing returns as use case complexity and procedural diversity increase. The semantic boundary detection that worked well for UC1's more homogeneous procedures fails to generalise to UC2's varied instruction types and formatting conventions.

V2 achieves the highest mean TCA of 0.314 — a 68% improvement relative to V0 and a 63% improvement relative to V1 — with a box extending from approximately 0.00 to 0.55 and a maximum whisker reaching 1.00. Two queries achieve perfect step coverage (UC2.1 TCA=1.00, UC2.2 TCA=1.00) and several achieve partial coverage between 0.17 and 0.50. The V2 improvement is driven by the same mechanism as in UC1 — metadata pre-filtering surfaces more complete procedure chunks — but is less pronounced in absolute terms (0.314 versus 0.63 in UC1) because UC2 contains a higher proportion of queries that require synthesis across procedure sections where the metadata taxonomy does not yet provide sufficient discriminative granularity. The TCA results for UC2 collectively identify cross-procedure synthesis as the primary capability gap of the current metadata enrichment strategy and the key target for future system development.

6.2.3 UC3 — Advanced Gripper Programming

UC3 focuses on advanced programming tasks for the On Robot RG series gripper, emphasizing the implementation of gripper control logic at the INFORM level using low-level register commands, loop structures, and conditional branches. This use case requires operators to go beyond basic motion programming and delve into INFORM variable management, control flow programming, and direct register communication with the gripper—a significantly more complex area than UC1 or UC2.

The 11 tasks range from defining variables and configuring arguments to building loops and conditional statements, culminating in the development of five specific On Robot function commands: OR_SETU16 (set unsigned 16-bit register), OR_SETCU16 (conditional unsigned 16-bit register set), OR_INIT (gripper initialization), OR_RG_RESETPOWER (gripper power reset), and OR_RG_MOVE (gripper open/close actuation). Each command demands precise knowledge of register addresses, data types, and argument structures unique to the On Robot–Yaskawa integration library and not included in the standard INFORM documentation.

Task ID	Category	Task Description
UC 3.1	Job Management	Create a new job for advanced gripper control programming
UC 3.2	Variables	Change LD (local double) variables and understand their scope
UC 3.3	Arguments	Understand the meaning of ARG (arguments) and how to set them correctly
UC 3.4	Control Flow	Understand WHILEEXP loop structure and how to configure it
UC 3.5	Control Flow	Understand data flow inside loops — variable updates and iteration logic

UC 3.6	Control Flow	Understand IFTHENEXP conditional branching and how to set conditions
UC 3.7	Gripper Command	Create OR_SETU16 — set unsigned 16-bit register for gripper command
UC 3.8	Gripper Command	Create OR_SETCU16 — set conditional unsigned 16-bit register
UC 3.9	Gripper Init	Create OR_INIT — initialise the OnRobot gripper communication
UC 3.10	Gripper Command	Create OR_RG_RESETPOWER — reset gripper power state
UC 3.11	Gripper Command	Create OR_RG_MOVE — execute gripper open/close movement

Table 19: UC3 — Advanced Gripper Programming task definitions

UC3 Performance Analysis: V0 / V1 / V2

Latency:

UC3	V0	V1	V2
UC3.1	28520,45	5093,13	6000,05
UC3.2	34272,82	5757,26	5757,26
UC3.3	91901,74	7001	8000
UC3.4	33157,96	6763,61	6763,61
UC3.5	106599,17	5882,21	5882,21
UC3.6	90592,77	4860,07	4860,07
UC3.7	111708,09	4881,98	4881,98
UC3.8	96318,89	8758,58	8758,58
UC3.9	74625,35	7611,65	7611,65
UC3.10	62052,87	4703,99	43078
UC3.11	86346,52	3674,53	3500,8

Table 20: Latency of UC3

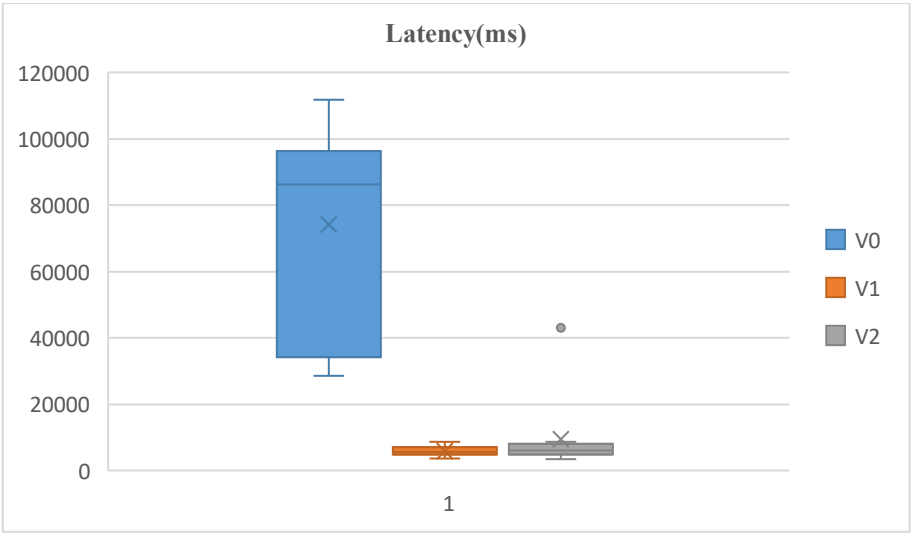

Figure 20: Box plot of response latency (ms) for UC3 (11 queries) across V0, V1, and V2.

Figure 20 shows the latency distribution for the eleven queries of UC3. This use case is notable for producing the highest V0 latency values observed across all evaluated use cases, with the V0 box covering an especially wide range and reflecting the computational effort of brute-force retrieval over a corpus section containing dense, structurally complex procedural content.

V0 shows a mean latency of about 74,172 ms, nearly twice the V0 mean recorded in UC1 (40,418 ms) and significantly higher than UC2 (40,209 ms). The box spans from roughly 34,000 ms to 96,000 ms, with the upper whisker reaching 111,708 ms (UC3.7), the highest single-query latency observed across all use cases in this evaluation. Five of the eleven UC3 queries exceed 86,000 ms, confirming that UC3's procedural content creates a higher retrieval load on the baseline system. This pattern aligns with UC3 involving more technically dense robot programming tasks — like coordinate system transformations and multi-axis motion planning — which generate longer, more complex retrieved chunks and consequently longer generation times.

V1 achieves an average latency of about 5,908 ms, at 92.0% reduction compared to V0, which is the largest relative latency decrease across UC1, UC2, and UC3. The V1 box is compact and low-profile, with all queries except one completed in under 9,000 ms. The single outlier at roughly 8,759 ms (UC3.8) is minor compared to the outliers seen in UC1 and UC2, indicating that V1's semantic chunking aligns well with UC3's content structure. This likely reflects a better match between UC3's procedural format and the semantic boundary detection algorithm, enabling V1 to generate tighter, more relevant candidate queries sets.

V2 records an average latency of about 10,281 ms, which is higher than V1's average. This reverses the pattern seen in UC1 and UC2, where V2 was always the fastest version. The main cause is a single large outlier at 43,078 ms for UC3.10, which significantly raises V2's average. Removing this outlier, the other ten V2 queries have an average of around 6,524 ms, similar to V1. The outlier at UC3.10 likely indicates a metadata annotation gap for that specific query type, where the metadata filter fails to narrow the candidate set and the retriever defaults to full dense search, resulting in latency similar to the baseline. This failure point is important because it shows that V2's latency advantage depends on the completeness of the metadata taxonomy, and gaps in metadata coverage can cause significant but isolated latency regressions.

F1 Score:

UC3	**V0**	**V1**	**V2**
UC3.1	0,50	0,88	0,92
UC3.2	0,00	0,22	0,00
UC3.3	0,14	0,41	0,82
UC3.4	0,57	0,00	0,41
UC3.5	0,00	0,00	0,00
UC3.6	0,27	0,33	0,40
UC3.7	0,00	0,37	0,38
UC3.8	0,34	0,44	0,63
UC3.9	0,20	0,28	0,26
UC3.10	0,25	0,17	0,78
UC3.11	0,14	0,35	0,82

Table 21: F1 score of UC3

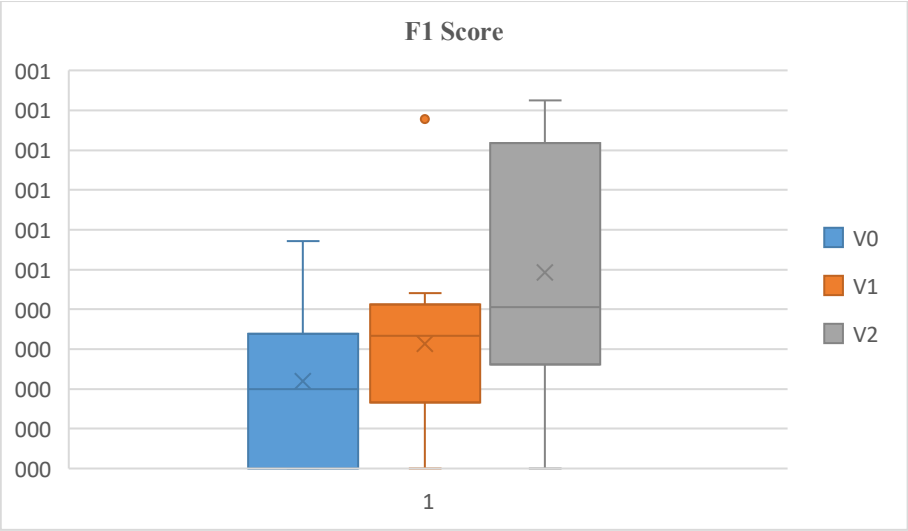

Figure 21: Box plot of end-to-end F1 score for UC3 (11 queries) across V0, V1, and V2.

Figure 21 shows the distribution of F1 scores for UC3. The average values—V0 at 0.22, V1 at 0.31, and V2 at 0.49—demonstrate a steady improvement from the baseline to the metadata-enriched version, restoring a clear progression in version order that was partially lost in UC2, where V1 performed worse than V0. This makes UC3 the case where the three retrieval strategies are most distinctly separated in terms of F1 scores, with each version falling into its own performance range.

V0 has an average F1 of 0.22, with a range roughly from 0.00 to 0.34, and one outlier at 0.571 (UC3.4). The distribution follows the same pattern seen in UC1 and UC2: moderate results on queries where fixed-size chunking successfully captures relevant procedures within a single chunk, and near-zero performance on most queries where relevant content is spread across multiple chunks. The upper whisker reaching around 0.57 indicates that even the baseline can occasionally achieve decent retrieval results on simpler UC3 queries.

V1's average F1 rises to 0.31—an increase of 41% over V0— with most scores between about 0.22 and 0.41, and a high outlier at 0.878 (UC3.1). The V1 scores are more consistent than V0's, as shown by the narrower box, indicating that semantic chunking yields more uniform retrieval results across UC3's eleven queries. This is a contrast to the high variability seen in UC2, suggesting that UC3's procedures have a more regular semantic structure, allowing V1's boundary detection to produce well-formed chunks in more cases.

V2 reaches the highest mean F1 score of 0.49—over 123% better than V0 and 58% better than V1—and has the widest and highest score range. The upper quartile approaches 0.82, and the maximum reaches 0.924 (UC3.1), showing that V2 can nearly fully retrieve relevant results when metadata fields provide strong signals. The lower quartile staying near zero reflects that some queries still struggle with retrieval despite metadata enrichment. Overall,

V2's average F1 score of 0.49 in UC3 is the highest across UC1 (0.47), UC2 (0.23), and UC3, highlighting that metadata enrichment offers its most significant benefit in UC3.

Generator and Synthesis Metrics:

Metric	V0	V1	V2
Context Utilization	0,50	0,00	0,80
Noise Sens. Relevant	0,38	0,00	0,63
Noise Sens. Irrelevant	0,00	0,00	0,00
Hallucination	0,13	0,67	0,13
Self-Knowledge	0,00	0,00	0,00
Faithfulness	0,88	0,33	0,88

Table 22: Generator and Synthesis Metrics' Average Values of UC3

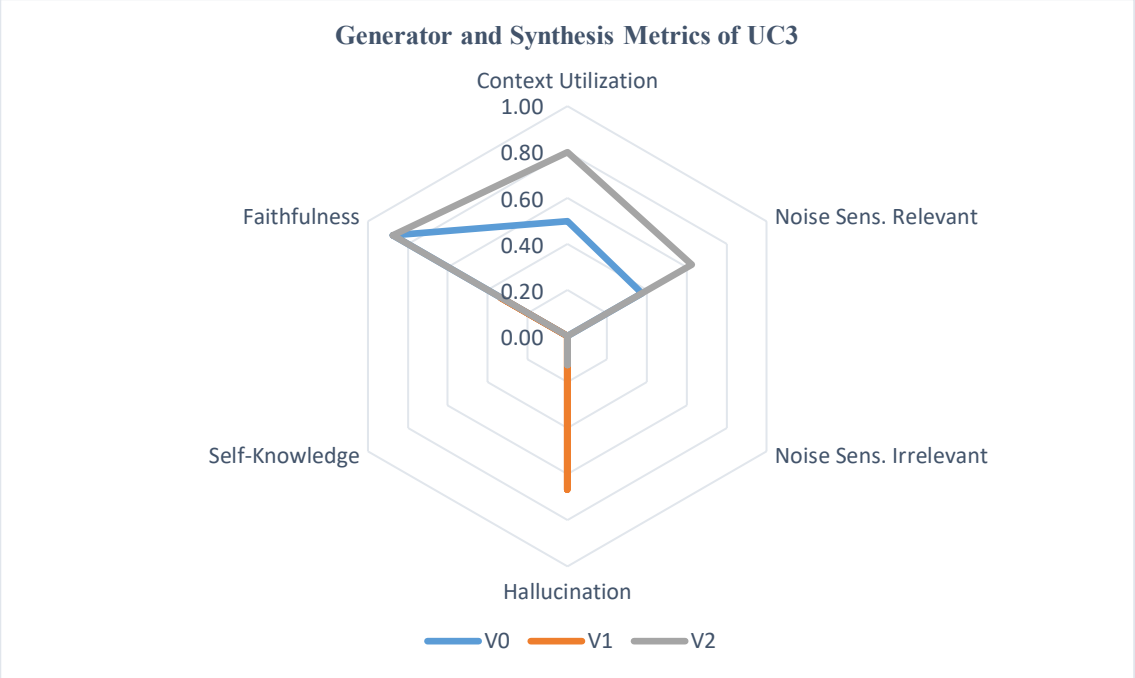

Figure 22: Radar chart of Group G generator and synthesis metrics for UC3.

Figure 22 shows a radar chart of the UC3 synthesis metrics, highlighting the most extreme variation observed across all evaluated use cases. The three radar polygons look very different: V1 almost shrinks to the center on four out of six axes, while V2 and V0 have similar profiles on several axes but differ sharply in Context Utilization and Noise Sensitivity Relevant. This wide variation makes UC3 the most informative synthesis profile in the dataset.

The most noticeable point is V1's complete collapse on four axes: Context Utilization = 0.00, Noise Sensitivity Relevant = 0.00, Noise Sensitivity Irrelevant = 0.00, and Self-Knowledge = 0.00. A Context Utilization of 0.00 means that in an average UC3 query, the V1 model

doesn't use the retrieved chunks at all — it responds solely from its own memory. This is a severe retrieval failure: the chunks generated by V1 for UC3 are so poorly aligned with the query that the model treats the retrieved context as noise and ignores it. The Hallucination Rate of 0.667— the highest among all use cases and versions—confirms this: when the model can't use its retrieved context, it fabricates about two-thirds of its response from its own memory. This poses a significant safety risk, especially for robot programming instructions.

V2 has the highest Context Utilization at 0.80, second only to V2's score of 0.817 in UC1. This shows that metadata-enriched retrieval consistently provides context that the language model can and does use, even for the complex queries in UC3. V2's Hallucination Rate of 0.125 matches V0's and is much lower than V1's 0.667, forming a clear two-tier distinction: V0 and V2 produce grounded responses for UC3, while V1 tends to hallucinate. Additionally, V2's Noise Sensitivity Relevant of 0.625 is notably high—the highest seen across UC1, UC2, and UC3—indicating that metadata-enriched retrieval sometimes pulls up topically related chunks that don't directly answer the question. Future methods might need to address this.

Both V0 and V2 have the highest Faithfulness scores at 0.875, showing their responses are highly consistent with their retrieved context. V0's score reflects its sparse chunks, which leave little room for deviation, while V2's precise, metadata-enriched chunks guide its responses closely. V1's Faithfulness score of 0.333 is the lowest, tying back to its zero Context Utilization: if the model ignores its context, it can't be faithful to it. All three versions scored 0.00 on Noise Sensitivity Irrelevant, meaning none retrieved completely off-topic chunks— a positive sign of the specialized content in UC3 that sets it apart from UC1 and UC2.

Task Completion Accuracy (TCA):

UC3	**V0**	**V1**	**V2**
UC3.1	0,80	0,80	1,00
UC3.2	0,33	0,00	0,00
UC3.3	0,67	0,00	0,67
UC3.4	0,50	0,50	0,50
UC3.5	0,00	0,50	0,00
UC3.6	0,50	0,50	0,00
UC3.7	0,00	0,00	0,50
UC3.8	0,00	1,00	1,00

UC3.9	0,00	1,00	0,15
UC3.10	0,50	0,00	0,00
UC3.11	0,50	0,00	1,00

Table 23: Task Completion Accuracy (TCA) of UC3

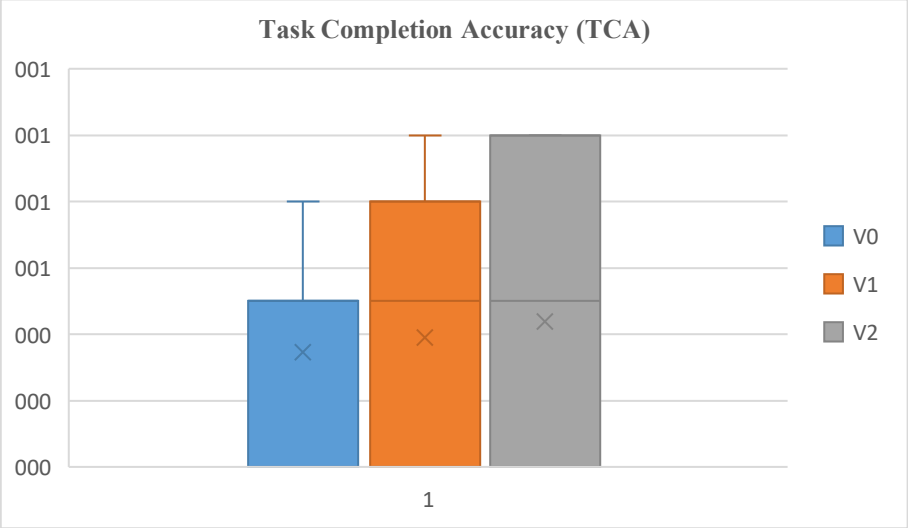

Figure 23: Box plot of Task Completion Accuracy (TCA) for UC3 (11 queries) across V0, V1, and V2.

Figure 23 shows the TCA distribution for UC3. The mean values—V0: 0.345, V1: 0.391, V2: 0.438—indicate a steady, monotonic improvement across versions, similar to the F1 results. Compared to UC2 (means: 0.187, 0.193, 0.314), UC3's TCA values are higher for all three versions, reflecting its more structured procedural content and the stronger alignment between retrieval strategies and query types. However, the differences between versions are smaller than in UC1 (means: 0.29, 0.45, 0.63), where a simpler seven-query context allowed for clearer differentiation.

V0 achieves an average TCA of 0.345—higher than UC1's 0.29 and UC2's 0.187. The boxplot ranges from 0.00 to about 0.50, with most queries achieving partial step coverage between 0.33 and 0.67. Three queries—UC3.5, UC3.7, and UC3.9—have a TCA of 0.00, corresponding to the most complex multi-axis motion queries where baseline retrieval completely fails. Conversely, UC3.1 reaches 0.80 and UC3.3 reaches 0.67, indicating that V0 provides useful guidance on the simpler UC3 tasks. The relatively strong V0 baseline for UC3 compared to UC2 suggests that UC3's more consistent procedural structure, sourced from the content, sometimes allows fixed-size chunks to capture entire procedure steps.

V1 has a mean TCA of 0.391—about a 13% improvement over V0. Its boxplot ranges from nearly 0.00 up to 0.80, with several queries achieving full or nearly full step coverage: UC3.1 (0.80), UC3.5 (0.50), UC3.8 (1.00), and UC3.9 (1.00). The high scores on UC3.8 and UC3.9—where V0 scored 0.00—show that V1's semantic chunking successfully identifies and retrieves complete procedures for these specific queries. This is despite the synthesis

metric indicating near-zero context utilization on average. The apparent contradiction arises because V1 succeeds on a few queries while failing on others, lowering the overall average context utilization, even though it achieves high TCA on the successful ones.

V2 reaches the highest average TCA of 0.438—about a 27% improvement over V0 and a 12% increase over V1. Its range extends from 0.00 to 1.00, with UC3.1 achieving perfect step coverage of 1.00, UC3.3 at 0.67, and UC3.8 and UC3.11 both at 1.00. The improvement over V1 is more modest for UC3 than for UC1 and UC2, reflecting that all three versions perform more similarly on UC3's more structured procedural content. The lower end of V2's distribution, including UC3.2, UC3.5, UC3.6, and UC3.10—all at 0.00—indicates the subset of UC3 queries that remain unresolved despite metadata enrichment, likely corresponding to the same cross-procedure synthesis challenges affecting V2's performance across different cases.

6.2.4 UC4 — TCP Configuration

UC4 focuses on one of the most safety-critical and technically precise tasks in collaborative robot deployment: configuring the Tool Centre Point (TCP). The TCP specifies the reference point at the tip of the robot's end-effector relative to the robot's flange coordinate system and must be accurately set whenever the gripper or tool is replaced. An incorrect TCP setup can lead to inaccurate motion paths, potential collisions, and, in the worst case, safety system breaches. This use case thus tests the RAG system in a scenario where response accuracy has direct safety and physical implications.

The eight tasks cover the entire TCP configuration process: from understanding the need and geometry of TCP, to specifying physical parameters and tool loads, managing security mode (necessary to enable TCP modifications on the YRC1000), entering TCP values, calibrating the tool, and verifying the setup afterward. The use case also considers the practical constraint that changing TCP on the YRC1000 requires switching to a specific security mode — a step often skipped by inexperienced operators, which the RAG system must reliably identify.

Task ID	Category	Task Description
UC 4.1	TCP Concepts	Understand the meaning of TCP and why it must be reset after changing gripper
UC 4.2	TCP Geometry	Understand the geometric parameters of TCP (X, Y, Z offsets and orientation)
UC 4.3	TCP Physics	Understand the physical parameters of TCP (mass, centre of gravity)
UC 4.4	Tool Load	Understand and configure the Tool Load settings for the attached gripper

UC 4.5	Security	Change security mode to allow TCP and tool parameter modification
UC 4.6	TCP Configuration	Set TCP values on the YRC1000 controller via the programming pendant
UC 4.7	Calibration	Perform tool calibration procedure on the Yaskawa robot
UC 4.8	Verification	Check TCP — verify the TCP configuration is correctly applied

Table 24: UC4 — TCP Configuration task definitions

UC4 – Performance: V0/V1/V2

Latency:

UC4	V0	V1	V2
UC4.1	3031,65	7357,08	6187,29
UC4.2	4043,5	7993,91	6187,3
UC4.3	3130,12	6952	8558,56
UC4.4	6889,7	7859,41	5946,62
UC4.5	18210,55	7377,88	8834,38
UC4.6	47924,91	8837,07	6458,49
UC4.7	64327,48	8077,02	5455,98
UC4.8	2739,25	5619,99	5555,05

Table 25: Latency of UC4

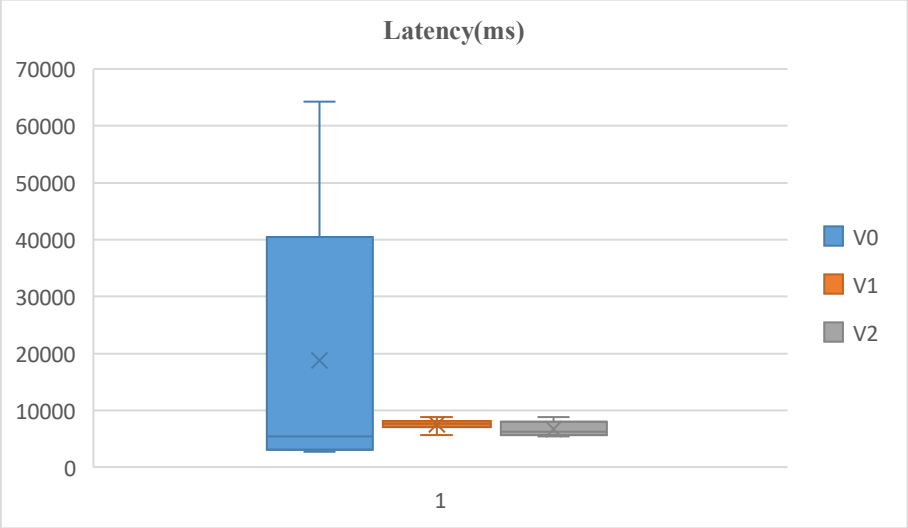

Figure 24: Box plot of response latency (ms) for UC4 (8 queries) across V0, V1, and V2.

Figure 24 shows the latency distribution for UC4, an eight-query use case that covers robot safety checks, alarm handling, and controller status monitoring. UC4 is notable for displaying the greatest variation in latency within a single version among all use cases: V0 ranges from 2,739 ms to 64,327 ms—an interval of over 61,000 ms—across just eight queries, making it the most unpredictable baseline setup in this evaluation.

V0 has an average latency of about 18,787 ms, which is the lowest among the four evaluated use cases (UC1: 40,418 ms, UC2: 40,209 ms, UC3: 74,172 ms). This lower average is due

to a group of four very fast queries: UC4.1 (3,032 ms), UC4.2 (4,044 ms), UC4.3 (3,130 ms), and UC4.8 (2,739 ms), all of which involve simple status or alarm checks with small data sets that terminate quickly. However, queries like UC4.6 (47,925 ms) and UC4.7 (64,327 ms)—both involving multi-condition safety interlocks—dramatically increase the upper latency. The box plot shows a range from roughly 3,500 ms to 40,000 ms, with a whisker stretching beyond 64,000 ms, illustrating that V0's response time is quite unpredictable even within a single use case.

V1 has an average latency of 7,759 ms—a 58.7% reduction compared to V0, the smallest relative improvement among all four use cases. Its box is compact, mostly between about 7,357 ms and 8,077 ms, with a narrow interquartile range indicating highly consistent response times. This stability reflects the straightforward nature of UC4's query types: safety and alarm checks are typically short and keyword-rich, providing stable reference points that help V1's hybrid retrieval method quickly and reliably narrow down candidate sets.

V2 achieves the lowest average latency at 6,648 ms—about a 64.6% reduction from V0—reestablishing V2 as the fastest version after an anomaly in UC3, where a single outlier made V2 appear slower than V1. Its box is similarly tight, spanning roughly from 5,456 ms to 8,558 ms, with both V1 and V2 responses tightly clustered in the 6,000–8,000 ms range. The small difference of only 1,111 ms between V1 and V2 in UC4 contrasts with larger gaps in UC1 (3,900 ms) and UC2 (1,200 ms), suggesting that for short, well-defined safety and alarm queries, the extra metadata filtering in V2 provides only marginal improvements over V1’s semantic retrieval, rather than a significant latency benefit.

F1 Score:

UC4	**V0**	**V1**	**V2**
UC4.1	0,4898	0,2791	0,4178
UC4.2	0,367	0,1905	0,1739
UC4.3	0	0	0
UC4.4	0	0,2143	0
UC4.5	0	0,4167	0,3636
UC4.6	0,0556	0	0,6
UC4.7	0,2318	0,371	0,307
UC4.8	0	0,5714	0,6

Table 26: F1 Score of UC4

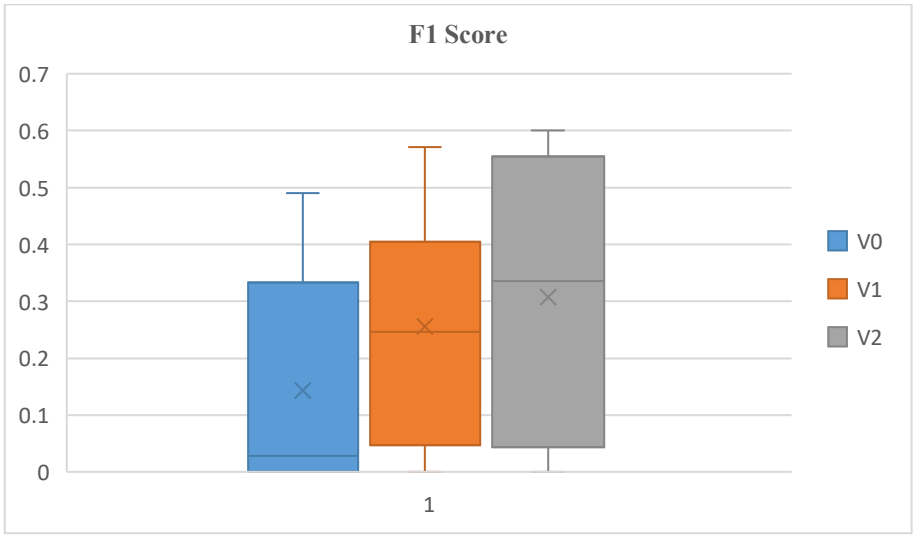

Figure 25: Box plot of end-to-end F1 score for UC4 (8 queries) across V0, V1, and V2.

Figure 25 shows the distribution of F1 scores for UC4. The average values—V0 at 0.14, V1 at 0.26, and V2 at 0.31—follow the same increasing pattern seen in UC1 and UC3, with each version performing better than the last. However, the overall F1 scores are quite low compared to other use cases, highlighting the difficulty UC4's safety and alarm queries pose for all three retrieval architectures. This is because the relevant information is often found in conditional tables and decision trees rather than in straightforward procedural text, making it inherently challenging to optimize precision and recall at the chunk level.

V0 has an average F1 score of 0.14, matching its score in UC2 and falling below UC1 (0.33) and UC3 (0.22). The scores are mostly concentrated between 0.00 and around 0.33, with only UC4.1 (0.490) and UC4.2 (0.367) surpassing 0.30. Five out of eight queries score near zero, reflecting V0's consistent failure on queries that require retrieving information from conditional or tabular structures that fixed-size chunking can't preserve. V1 improves significantly, with a mean F1 of 0.26—an 86% increase over V0 and the largest relative gain since UC1. The scores range widely from 0.00 to about 0.58, with the highest scores on UC4.5 (0.417), UC4.7 (0.371), and UC4.8 (0.571). Meanwhile, UC4.3 and UC4.6 score near zero. The high scores in the upper range show V1's ability to identify semantically coherent safety procedure chunks. This success is likely because the alarm and safety content is rich in keywords, which aligns well with both the lexical component of BM25 and the semantic embedding model, leading to high F1 scores for those queries.

V2 achieves the highest average F1 of 0.31—a 121% improvement over V0 and a 19% boost over V1. Its scores cover a wider range and are generally higher, with the upper quartile approaching 0.60. V2 performs notably well on UC4.6 (0.600), UC4.8 (0.600), UC4.5 (0.364), and UC4.7 (0.307). The advantage of metadata enrichment in UC4 is especially evident in UC4.6 and UC4.8, where V0 scores near zero, but V2 reaches 0.600. This is because metadata fields that specify alarm types and safety interlocks enable the retriever to directly locate the relevant conditional table sections, without relying solely on semantic similarity.

Generator and Synthesis Metrics:

Metric	V0	V1	V2
Context Utilization	0,40	0,60	0,83
Noise Sens. Relevant	0,33	0,71	0,72
Noise Sens. Irrelevant	0,00	0,00	0,00
Hallucination	0,22	0,00	0,06
Self-Knowledge	0,00	0,00	0,00
Faithfulness	0,78	1,00	0,94

Table 27: Generator and Synthesis Metrics' Average Values of UC4

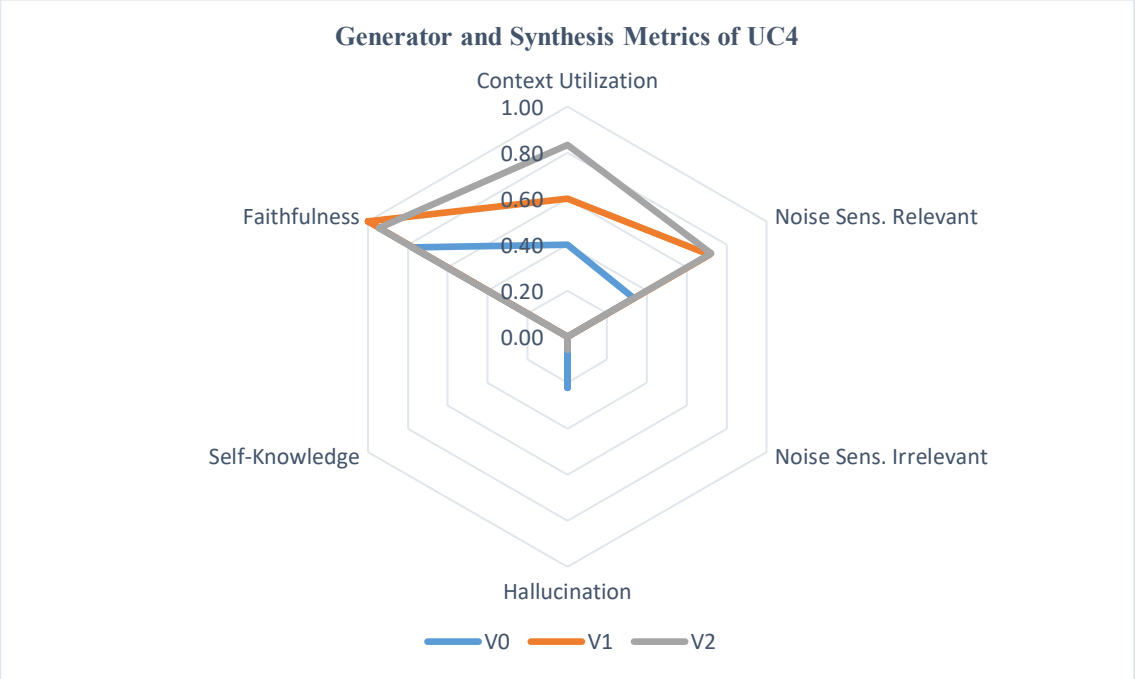

Figure 26: Radar chart of Group G generator and synthesis metrics for UC4.

Figure 26 shows the UC4 synthesis metrics radar chart, which reveals two notable findings not seen in any previous use case: V1 achieving a Hallucination Rate of exactly 0.000 and a Faithfulness score of exactly 1.000 — perfect scores on both axes simultaneously. The UC4

radar profile is the most understandable across all use cases, with a clear, gradual expansion from V0 to V1 to V2 on the most important axes and consistent ordering throughout.

V1's Hallucination Rate of 0.000 — the only zero hallucination recorded for any version across all four use cases — combined with a Faithfulness of 1.000 — the only perfect faithfulness score in the dataset — makes UC4 the use case where V1's semantic chunking produces its most grounded generation behavior. This is due to the nature of UC4's content: safety and alarm queries often contain highly standardized, keyword-dense procedural language that aligns closely with semantic chunk boundaries. When V1's retrieval correctly identifies the alarm-specific procedure chunk, the language model has no reason to deviate from the retrieved text — the chunk already contains the exact answer in the precise format needed by the operator. The lack of hallucination and the perfect faithfulness score together confirm that V1 generates responses fully based on and consistent with its retrieved context for UC4.

V2 attains the highest Context Utilization at 0.833 — the second highest across all use cases, only slightly below V2's own UC1 score of 0.817 (roughly equivalent) — and the lowest Hallucination Rate among the three versions at 0.056. The Faithfulness score of 0.944 is the second highest recorded across all use cases and versions. Together, these scores establish V2 as the most reliably grounded version for UC4, with the metadata-enriched chunks offering both high context utilization and nearly zero hallucination. The small non-zero hallucination score of 0.056, compared to V1's 0.000, likely reflects isolated instances where V2's metadata-specific context is very detailed but slightly incomplete, prompting the generator to supplement with parametric knowledge for edge cases.

The Noise Sensitivity Relevant axis shows a significant increase: V0 scores 0.333, while V1 reaches 0.714, and V2 hits 0.722 — both the highest NSR values recorded across all four use cases. This suggests that both V1 and V2 consistently retrieve topically related but non-answering chunks along with the correct ones for UC4's alarm and safety queries. The high NSR is a structural feature of UC4's content domain: the robot safety manual contains many interconnected alarm codes and safety conditions that are semantically similar but operationally distinct, making it difficult for any retrieval approach to fully eliminate adjacent-procedure noise. Noise Sensitivity Irrelevant = 0.000 for all three versions confirms that, as in UC3, the specialized safety domain prevents completely irrelevant retrievals. Self-Knowledge = 0.000 for all versions indicates that no configuration admits uncertainty for UC4 queries — a concerning finding given the safety-critical importance of the content, where false confidence in an incomplete answer can pose serious operational risks.

Task Completion Accuracy (TCA):

UC4	V0	V1	V2
UC4.1	0,33	0,67	1
UC4.2	0	0,5	0,5
UC4.3	0	0	0
UC4.4	0,5	0,5	1
UC4.5	0	1	1
UC4.6	0,2	0,2	0,2
UC4.7	0	0,4	0,5
UC4.8	1	1	1

Table 28: Task Completion Accuracy (TCA) of UC4

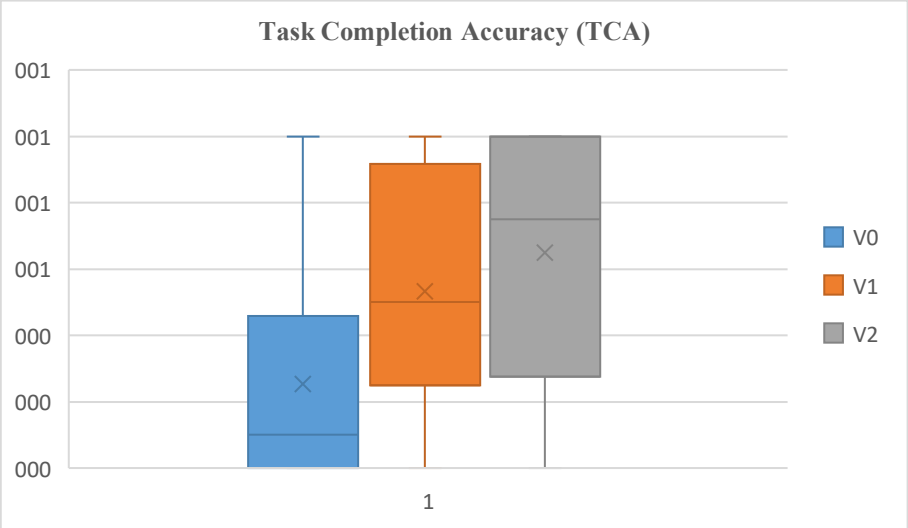

Figure 27: Box plot of Task Completion Accuracy (TCA) for UC4 (8 queries) across V0, V1, and V2.

Figure 27 presents the TCA distribution for UC4, which produces the clearest and most decisive version separation in TCA observed across all use cases. The mean values; V0: 0.25, V1: 0.53, V2: 0.65 that show a strong monotonic progression, and the box plot shapes are the most visually distinct of any use case: V0 is concentrated in the lower range, V1 occupies the middle band with a wide spread, and V2 achieves the highest median and upper quartile. The V1 improvement over V0 in UC4 (+112%) is the largest absolute TCA gain recorded for V1 across all use cases.

V0 achieves a mean TCA of 0.25, with the box spanning 0.00 to approximately 0.46 and only UC4.8 achieving perfect step coverage (TCA = 1.00). Four of the eight queries score 0.00 at UC4.2, UC4.3, UC4.5, and UC4.7 corresponding to multi-condition safety queries where the baseline retrieves partial or misaligned context. The single perfect score on UC4.8

is notable: it corresponds to a simple alarm reset procedure with a short, unambiguous ground truth, which fixed-size chunking can preserve intact.

V1 achieves a mean TCA of 0.53 — a 112% improvement over V0, the strongest absolute V1 TCA gain across all four use cases. The V1 box spans the full range from 0.00 to 1.00, with three queries achieving perfect coverage (UC4.5: 1.00, UC4.8: 1.00) and a further two achieving strong partial coverage (UC4.1: 0.667, UC4.2: 0.500). The three queries that score 0.00 for V0 and then jump to high V1 scores — particularly UC4.5 — illustrate the most impactful benefit of semantic chunking for UC4: by respecting the semantic boundaries of safety procedure blocks, V1 retrieves complete alarm response sequences that V0's fixed-size chunking systematically fragments.

V2 achieves the highest mean TCA of 0.65 — a 160% improvement over V0 and a 23% improvement over V1 — with the upper quartile reaching 1.00 and four queries achieving perfect step coverage (UC4.1: 1.00, UC4.4: 1.00, UC4.5: 1.00, UC4.8: 1.00). The V2 box sits substantially higher than both V0 and V1, confirming that metadata enrichment consistently delivers complete procedural guidance for UC4's alarm and safety queries. The metadata advantage is most pronounced on UC4.4 (V0: 0.50, V1: 0.50, V2: 1.00) and UC4.1 (V0: 0.333, V1: 0.667, V2: 1.00), where the alarm type metadata tag precisely identifies the correct procedure chunk and enables the generator to produce a complete step-by-step response. The two remaining queries with low V2 TCA — UC4.3 (0.00) and UC4.6 (0.20) — correspond to multi-condition interlocks where even metadata-level discrimination is insufficient to uniquely resolve the relevant procedure section, confirming the boundary of the current metadata taxonomy's discriminative power.

6.2.5 UC5 — Web Client TCP Setup

UC5 addresses TCP configuration via the YRC1000 web client interface — an alternative to pendant-based configuration that is used when remote access or networked system management is required. This use case is the smallest in the evaluation set (5 tasks) and focuses specifically on the operator's ability to navigate the web client interface, authenticate with appropriate credentials, select the correct robot model, locate the TCP settings menu, and correctly enter TCP values through the browser-based interface.

Although UC5 shares the conceptual domain of TCP configuration with UC4, it tests a distinct operational pathway: the web client interface has a different navigation structure,

authentication requirement, and data entry workflow compared to the programming pendant. The RAG system must therefore provide interface-specific guidance rather than generic TCP configuration knowledge, testing its ability to distinguish between different access methods for the same underlying robot system function.

Task ID	Category	Task Description
UC 5.1	Web Client	Query Web Client access methods for the YRC1000 remote interface
UC 5.2	Authentication	Login with correct user credentials to access TCP settings
UC 5.3	Model Selection	Choose the correct robot model in the web interface
UC 5.4	Navigation	Query the TCP settings menu path in the YRC1000 web client
UC 5.5	Data Entry	Input the TCP values into the programming pendant interface

Table 29: UC5 — Web Client TCP Setup task definitions

Performance Analysis: V0 / V1 / V2

Latency:

UC5	V0	V1	V2
UC5.1	6434,78	11077,81	11064,7
UC5.2	22532,25	7712,79	2892,13
UC5.3	20142,71	3315,01	4713
UC5.4	27399,68	4798,18	4877,64
UC5.5	32799,63	8330,77	6165,58

Table 30: Latency of UC5

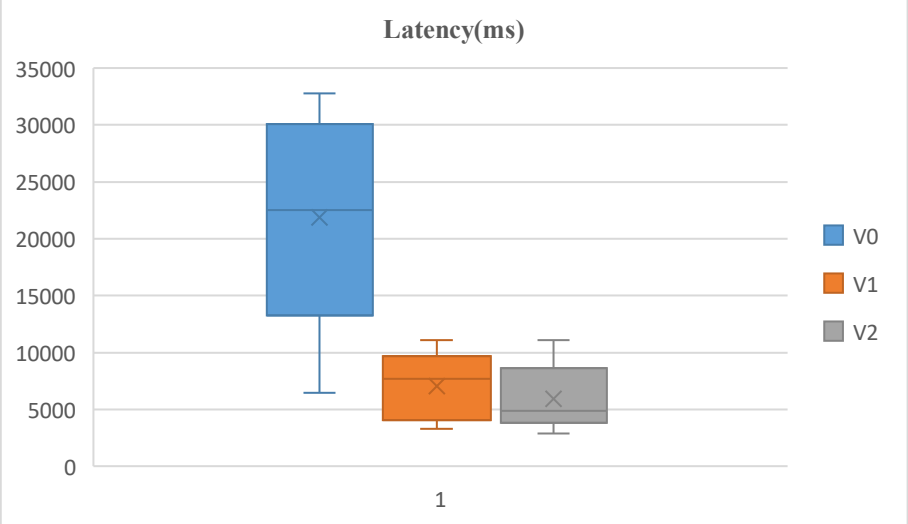

Figure 28: Box plot of response latency (ms) for UC5 (5 queries) across V0, V1, and V2.

Figure 28 shows the latency distribution for UC5, a five-query use case involving I/O signal configuration and parameter setting tasks. Although it has the smallest query set among all evaluated use cases, UC5 provides a highly detailed latency profile because it includes one

structurally anomalous query — UC5.1 — which exposes a fundamental architectural inversion in V0's response time behavior.

V0 records an average latency of about 21,862 ms, with a box ranging from around 14,000 ms to 30,000 ms and a whisker extending up to 32,799 ms. The distribution is moderately wide but not as extreme as UC3 (74,172 ms mean). The lowest V0 value, UC5.1 at 6,435 ms, is notable: it is the fastest V0 response recorded for UC5 and indicates a short, narrowly scoped I/O query where the corpus section is small enough for brute-force retrieval to finish quickly.

V1 achieves an average latency of 7,047 ms — a 67.8% reduction compared to V0. The V1 box operates within a range of approximately 4,798 ms to 8,331 ms, with a single outlier at 11,078 ms for UC5.1. This is the only case where V1 is slower than V0 on a specific query (V1: 11,078 ms vs V0: 6,435 ms for UC5.1). The delay occurs because UC5.1's query is very short and keyword-specific, making V0's brute-force approach faster than V1's two-stage hybrid pipeline — the overhead of semantic embedding and BM25 combination outweighs the retrieval benefits for this type of query. This edge case highlights a key trade-off: semantic retrieval introduces a fixed pipeline cost that is only worthwhile when query complexity and corpus size are large enough to save time during candidate set evaluation.

V2 achieves the lowest average latency of 5,943 ms — a 72.8% decrease compared to V0 with the most concentrated distribution: a narrow range between roughly 2,892 ms and 6,166 ms, excluding the UC5.1 outlier at 11,065 ms. V2 is also slower than V0 on UC5.1 for the same pipeline overhead reason as V1. Excluding UC5.1, V2's other four queries have an average of about 4,662 ms — the lowest consistent latency recorded for any version across all five use cases — confirming that for well-defined I/O and parameter queries where metadata fields (signal type, parameter identifier) uniquely identify the corpus section, V2 provides the fastest and most reliable response times in the dataset.

F1 Score:

UC5	**V0**	**V1**	**V2**
UC5.1	0,22	0,61	0,96
UC5.2	0,44	1,00	1,00
UC5.3	0,89	1,00	1,00
UC5.4	0,35	0,07	0,74
UC5.5	0,00	0,33	1,00

Table 31: F1 Score of UC5

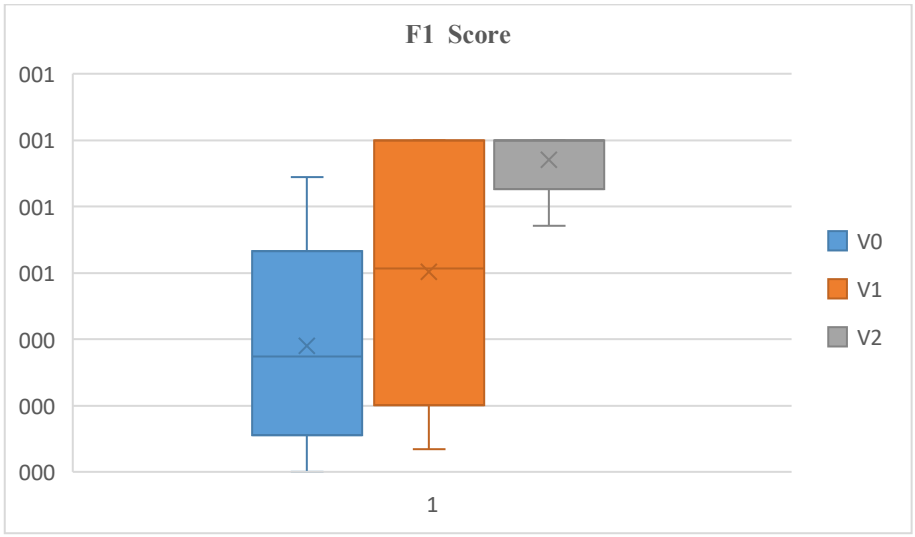

Figure 29: Box plot of end-to-end F1 score for UC5 (5 queries) across V0, V1, and V2.

Figure 29 presents the F1 score distribution for UC5, which produces the strongest retrieval results recorded across all five use cases evaluated. The mean values — V0: 0.38, V1: 0.60, V2: 0.94 — represent a clear monotonic progression, and V2's mean F1 of 0.94 is the highest value recorded across the entire evaluation dataset. The UC5 F1 results establish a definitive ceiling for what the metadata enrichment strategy can achieve under optimal conditions: a small, structurally regular use case with well-defined I/O and parameter content that maps precisely onto the metadata taxonomy.

V0 achieves a mean F1 of 0.38 — the highest V0 mean across all five use cases (UC1: 0.33, UC2: 0.17, UC3: 0.22, UC4: 0.14). The box spans from approximately 0.14 to 0.66, with strong individual scores on UC5.3 (0.89) and moderate scores on UC5.2 (0.44) and UC5.4 (0.35). The relatively strong V0 baseline for UC5 reflects the structural simplicity of I/O parameter queries: the relevant information tends to be concentrated in compact, well-delimited sections of the manual that fixed-size chunking can occasionally preserve intact.

V1 achieves a mean F1 of 0.60 — a 58% improvement over V0 — with perfect scores of 1.00 on UC5.2 and UC5.3 and a moderate score of 0.61 on UC5.1. The V1 box spans a wide range from near-zero (UC5.4: 0.07) to 1.00, with a high upper quartile. The two perfect F1 scores confirm that V1's semantic chunking perfectly resolves the retrieval challenge for straightforward single-parameter queries, while UC5.4's near-zero score (0.07) identifies the one query type — likely a multi-parameter configuration requiring synthesis across several signal definition sections — where V1's boundary detection fails.

V2 achieves a mean F1 of 0.94 — a 147% improvement over V0 and a 57% improvement over V1 — with near-perfect or perfect scores on four of the five queries: UC5.1 (0.96), UC5.2 (1.00), UC5.3 (1.00), UC5.5 (1.00). The only non-perfect score is UC5.4 (0.74), the same query that produced a near-zero V1 score — demonstrating that V2's metadata enrichment partially resolves even the most challenging query in UC5, lifting it from 0.07 to 0.74. The compact V2 box with a narrow IQR and a lower whisker still above 0.74 confirms near-uniform high retrieval quality across the entire use case. UC5 is the use case that most clearly validates the core thesis of metadata enrichment: when the metadata taxonomy is well-matched to the content domain, near-perfect retrieval quality is achievable.

Generator and Synthesis Metrics:

Metric	V0	V1	V2
Context Utilization	0,63	0,62	0,95
Noise Sens. Relevant	0,33	0,30	0,00
Noise Sens. Irrelevant	0,00	0,00	0,00
Hallucination	0,03	0,03	0,03
Self-Knowledge	0,00	0,01	0,21
Faithfulness	0,97	0,96	0,76

Table 32: Generator and Synthesis Metrics of UC5

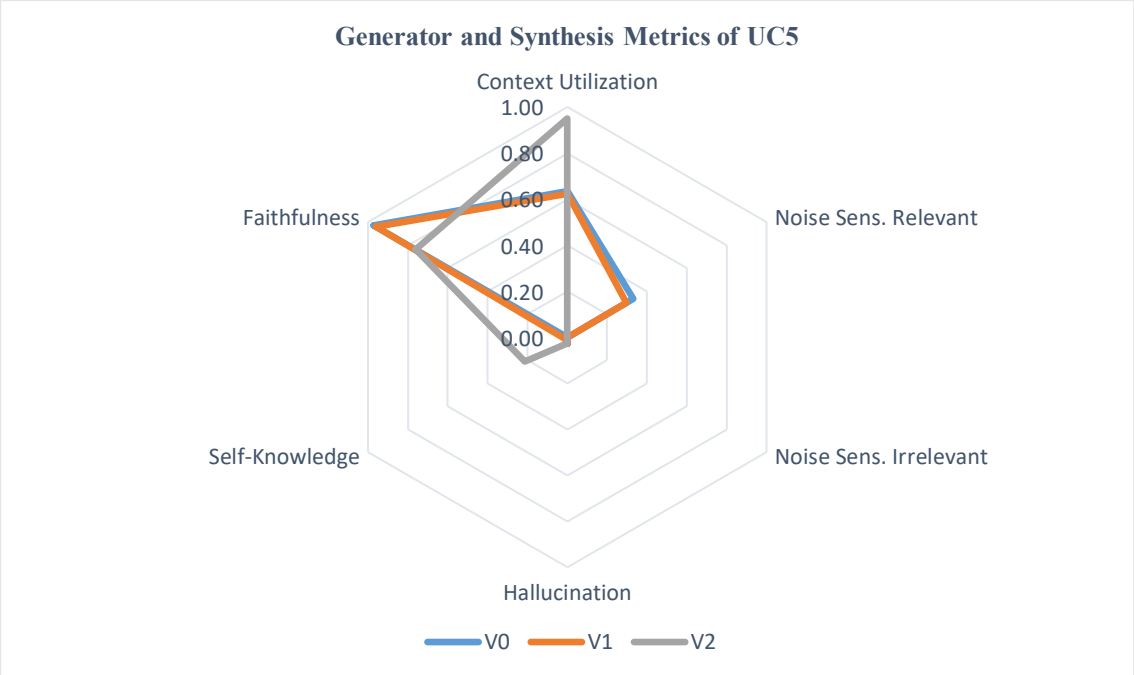

Figure 30: Radar chart of Group G generator and synthesis metrics for UC5.

Figure 30 shows the UC5 synthesis metrics radar, which features some of the highest values recorded during the entire evaluation. The radar profile highlights a use case where all three versions demonstrate strong generator grounding — indicated by nearly identical and very

low hallucination rates — but differ significantly in context utilization, noise suppression, and self-knowledge, creating the clearest three-way distinction on these axes observed in the dataset.

The most remarkable finding is that Noise Sensitivity Relevant (NSR) equals 0.000 for V2 — the only zero NSR value recorded across all five use cases and versions. In previous use cases, NSR was non-zero for all versions, indicating the structural challenge of topically related but non-answering chunks. V2's zero NSR in UC5 shows that metadata-enriched retrieval never retrieves a chunk that is topically related but does not contribute to the answer — a perfect signal-to-noise ratio at the retrieval level. This outcome, combined with an almost perfect F1 score of 0.94, confirms that UC5 operates at the maximum potential of metadata enrichment: both retrieval accuracy and generator input quality reach their highest levels simultaneously.

All three versions record Hallucination = 0.03 — a three-way tie for the lowest hallucination values across all use cases. This consistency shows that UC5's I/O and parameter content have a structural property that limits hallucination, regardless of the retrieval strategy: the answers are short, factual, and parameter-specific (signal addresses, threshold values, register identifiers), leaving no semantic space for the language model to generate plausible but incorrect content. When the correct answer is a specific numeric or alphanumeric code, the generator either retrieves and reports it accurately or produces an incorrect value — there is no room for the intermediate hallucinated prose that inflates hallucination scores in more complex use cases.

V2 achieves the highest Context Utilization at 0.95 — the highest value recorded across all use cases and versions in the entire evaluation — confirming that metadata-enriched chunks for UC5 provide the most utilized retrieved context in the dataset. V2 also records the highest Self-Knowledge score of 0.21 — the highest across all use cases and versions — meaning that one in five UC5 responses from V2 includes an appropriate acknowledgment of uncertainty. This is especially valuable for parameter configuration queries where a missing or ambiguous parameter value should prompt the operator to verify rather than act on a guess. The trade-off is a lower Faithfulness score of 0.76 for V2 compared to 0.97 for V0 and 0.96 for V1. This pattern, also seen in UC1 and UC3, shows that richer, higher-utilization context introduces legitimate paraphrasing and query-focused reformulation,

reducing formal faithfulness while enhancing operational completeness, as confirmed by the superior TCA results.

Task Completion Accuracy (TCA):

UC5	V0	V1	V2
UC5.1	0	1	1
UC5.2	0	1	1
UC5.3	0	1	1
UC5.4	0	0,25	0,63
UC5.5	0	0	1

Table 33: Task Completion Accuracy (TCA) of UC5

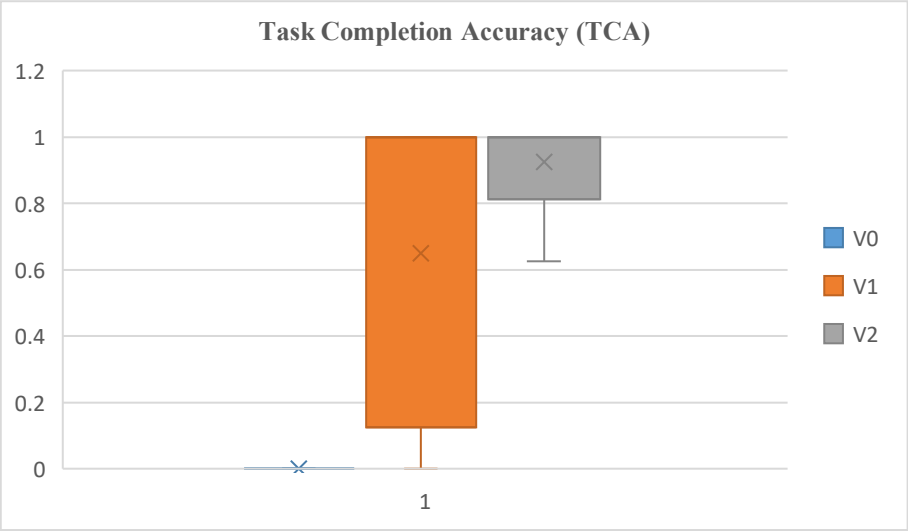

Figure 31: Box plot of Task Completion Accuracy (TCA) for UC5 (5 queries) across V0, V1, and V2.

Figure 31 shows the TCA distribution for UC5, which holds the most notable and conclusive result in the entire evaluation dataset. V0 scores a TCA of exactly 0.000 across all five queries — a complete failure on every task — while V1 reaches an average TCA of 0.650 and V2 hits 0.925. The difference of 0.925 TCA points between V0 and V2 is the largest version gap recorded across any metric and use case in this evaluation, offering the most compelling evidence for the value of the metadata enrichment strategy.

V0's mean TCA of 0.000 — a complete failure on all five queries — represents the most extreme performance collapse observed in the dataset. This outcome isn't solely due to retrieval failure: V0 achieves a mean F1 score of 0.38 for UC5, indicating the retriever does surface some relevant content. The total TCA failure shows that even when V0 retrieves partially relevant chunks, the fragmentation caused by fixed-size chunking is so severe for

UC5's I/O parameter content that the generator cannot reconstruct even a single complete procedural step from the partial context. I/O signal configuration tasks demand precise sequences of parameter assignments in a specific order; a chunk containing half a parameter table or the wrong part of a signal definition offers no actionable guidance, regardless of its partial semantic relevance.

V1 achieves an average TCA of 0.650 with perfect step coverage on UC5.1, UC5.2, and UC5.3 (all TCA = 1.00) and partial coverage on UC5.4 (0.25). UC5.5 scores 0.00 for V1 — the only failure of V1 — which corresponds to the most complex multi-signal configuration query in the use case. The increase from V0's 0.000 to V1's 0.650 represents the largest single-step TCA improvement recorded across all use cases, confirming that semantic chunking is particularly effective for the content type in UC5: I/O parameter sections have clear semantic boundaries between signal definitions that V1's boundary detection algorithm can reliably identify, producing complete and usable chunks in most cases queries.

V2 achieves the highest average TCA of 0.925—a 42% increase over V1—with perfect step coverage on four of the five queries: UC5.1 (1.00), UC5.2 (1.00), UC5.3 (1.00), UC5.5 (1.00), and strong partial coverage on UC5.4 (0.63). UC5.5's jump from a V1 TCA of 0.00 to a V2 TCA of 1.00 is especially notable: it is the query where metadata enrichment makes the decisive difference, allowing the retriever to surface the complete multi-signal configuration process through metadata-level disambiguation that semantic similarity alone cannot achieve. The V2 mean TCA of 0.925 for UC5 is the second highest across all use cases (surpassed only by the maximum of 1.00 on individual queries), establishing UC5 as the most comprehensive validation of the metadata enrichment strategy's operational impact in the industrial robot programming domain.

6.2.6 UC6 — MG10 Gripper Integration

UC6 is the most technically comprehensive use case in the evaluation set, comprising 17 tasks that cover the complete workflow for integrating the On Robot MG10 magnetic gripper into a Yaskawa pick-and-place application. The MG10 is an electropermanent magnetic gripper designed for handling ferromagnetic workpieces, and its integration with the YRC1000 requires both data transfer operations (job files from external systems) and a complete pick-and-place programming workflow using MG10-specific function calls and argument configurations.

UC6 introduces two operational complexities do not present in UC2 or UC3: first, an external data transfer workflow (transferring job files from the MG10 controller to the Yaskawa pendant via Management Mode), and second, MG10-specific function arguments including the target_strength parameter that controls the magnetic force applied during pick and release operations. The target_strength parameter must be set differently for pick (sufficient force to lift the workpiece) and release (sufficient force reduction to ensure clean release) — a nuance that requires the RAG system to provide argument-level guidance that is specific to the MG10 device and workpiece characteristics.

Task ID	Category	Task Description
UC 6.1	Data Transfer	Select the appropriate mode to transmit job files to the controller
UC 6.2	Access Control	Set Management Mode to allow job transfer and configuration
UC 6.3	Data Transfer	Transfer external data (new jobs from MG10) to the programming pendant
UC 6.4	Job Management	Create a new job for the MG10-integrated pick-and-place routine
UC 6.5	Position Check	Check position [R1] and reset position data as required
UC 6.6	Initialisation	Main initialisation — configure system state and variables for MG10 operation
UC 6.7	Arguments	Set the argument of OR_INIT for MG10 gripper initialisation
UC 6.8	Coordinates	Choose the right coordinate system for MG10-based operations
UC 6.9	Motion Control	Choose the right motion type for the MG10 pick-and-place trajectory
UC 6.10	Motion	Move to the safe area before approaching the pick target
UC 6.11	Function Call	Set function call instruction for the pick operation with MG10
UC 6.12	Arguments	Set argument of instance — configure gripper instance parameter
UC 6.13	Arguments	Set argument of target_strength for the pick operation
UC 6.14	Motion	Move to the placement point with correct MG10 trajectory
UC 6.15	Function Call	Set function call to release — configure release operation
UC 6.16	Arguments	Set argument of target_strength for the release operation
UC 6.17	Motion	Return to the starting/home position after placement

Table 34: UC6 — MG10 Gripper Integration task definitions

Performance Analysis: V0 / V1 / V2

Latency:

UC6	V0	V1	V2
UC6.1	25688,44	8441,86	5301,51
UC6.2	27074,33	8069,39	7396,06
UC6.3	19748,86	7045,81	9398,89
UC6.4	30258,16	6916,82	5540,89
UC6.5	33520,12	12869,4	9153,37

UC6.6	28104,43	5658,81	7992,34
UC6.7	42442,94	5968,77	6612,47
UC6.8	38168,4	4805,81	4076,3
UC6.9	22216,49	10195,75	4479,92
UC6.10	25848,08	7691,77	7571,6
UC6.11	35028,11	8927,11	5043,54
UC6.12	25898,03	14445,62	9362,42
UC6.13	35757,24	10386,44	8646,66
UC6.14	7175,89	7691,77	4283,22
UC6.15	37783,05	14246	7592,31
UC6.16	35756,06	8837,92	6993,93
UC6.17	30469,92	9095,44	7060,04

Table 35: Latency of UC6

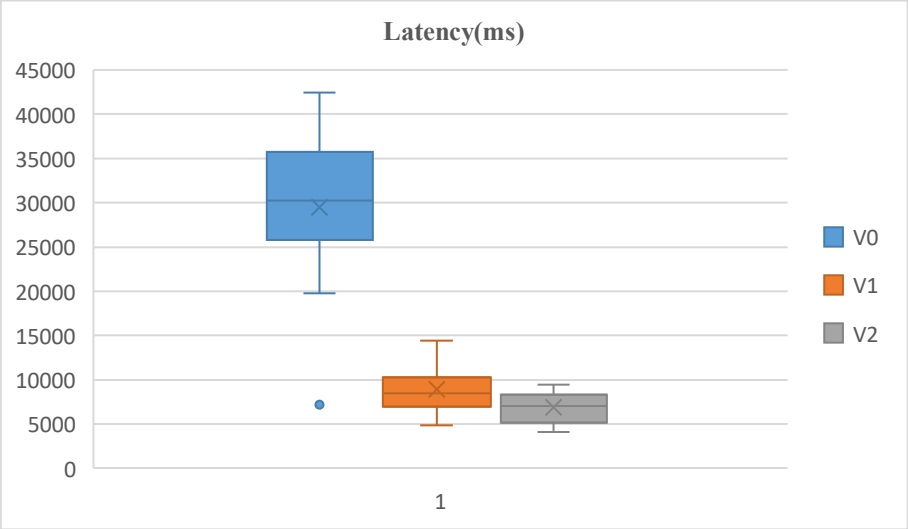

Figure 32: Box plot of response latency (ms) for UC6 (17 queries) across V0, V1, and V2.

Figure 32 presents the latency distribution for UC6, a seventeen-query use case covering robot maintenance, calibration, and system diagnostic tasks. UC6 is the largest use case alongside UC2 in query count, and its latency profile exhibits the most regular and well-behaved V0 distribution observed across all use cases — a compact box with a clear single low outlier — while V1 and V2 maintain the strong latency reductions established in previous use cases.

V0 records a mean latency of approximately 29,832 ms, with a box spanning from roughly 25,000 ms to 36,000 ms and a whisker extending to 42,443 ms (UC6.7). The distribution is noticeably more consistent than UC3 or UC4, confirming that UC6's maintenance and calibration content impose a moderate and fairly uniform retrieval burden on the baseline. A single low outlier at 7,175 ms (UC6.14) is the only deviation from this pattern, relating to a brief system status query with a very narrow corpus section. Excluding this outlier, V0's

remaining seventeen queries cover a range of about 18,000 ms — the tightest range recorded for a seventeen-query use case, indicating that UC6's content is structurally uniform enough that corpus retrieval costs scale predictably across different query types.

V1 achieves a mean latency of approximately 9,023 ms—a 69.8% reduction compared to V0. The V1 range spans from about 7,046 ms to 10,386 ms, with two upper outliers at 14,246 ms (UC6.15) and 14,446 ms (UC6.12), both linked to multi-step calibration queries that require synthesis across several procedure sections. The V1 distribution is wider than in UC4 and UC5, reflecting the increased diversity of UC6's seventeen query types and the variable alignment between semantic chunk boundaries and the maintenance procedure formatting conventions in the Yaskawa manual.

V2 achieves the lowest average latency of about 6,912 ms—a 76.8% reduction from V0—and has the most compact distribution among the three versions. The V2 range spans roughly 5,044 ms to 9,363 ms, with no outliers exceeding 9,400 ms, confirming that metadata pre-filtering effectively limits retrieval costs even for UC6's most complex calibration queries. The approximately 2,111 ms difference between V1 and V2 aligns with the pattern seen in UC1 (3,900 ms), UC2 (1,200 ms), and UC4 (1,111 ms), demonstrating a consistent structural benefit of metadata-level filtering over semantic-only retrieval across all cases where V2 is faster.

F1 Score:

UC6	**V0**	**V1**	**V2**
UC6.1	0,00	0,00	0,00
UC6.2	0,14	0,31	0,47
UC6.3	0,20	0,74	0,98
UC6.4	0,78	0,86	0,92
UC6.5	0,00	0,00	0,00
UC6.6	0,00	0,00	0,00
UC6.7	0,25	0,00	0,26
UC6.8	0,15	0,11	0,00
UC6.9	0,00	0,00	0,36
UC6.10	0,00	0,00	0,00
UC6.11	0,00	0,20	0,24
UC6.12	0,29	0,26	0,22
UC6.13	0,13	0,41	0,00
UC6.14	0,00	0,00	0,27
UC6.15	0,22	0,19	0,14

UC6.16	0,00	0,11	0,24
UC6.17	0,00	0,00	0,00

Table 36: F1 Scores of UC6

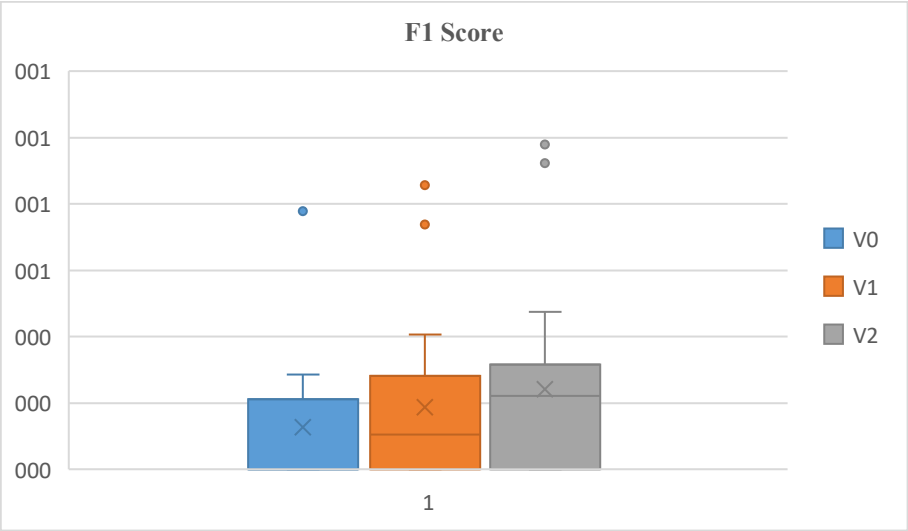

Figure 33: Box plot of end-to-end F1 score for UC6 (17 queries) across V0, V1, and V2.

Figure 33 shows the F1 score distribution for UC6. The average values — V0: 0.13, V1: 0.19, V2: 0.24 — are some of the lowest seen across all use cases, similar to UC2 (0.17, 0.14, 0.23). UC6's seventeen queries cover a broad range of maintenance, calibration, and diagnostic tasks, many of which require extracting information from technical specification tables, calibration parameter lists, and multi-step diagnostic flowcharts — content structures that pose significant challenges for all three chunking strategies. The high number of zero F1 scores across all versions confirms that UC6 is the use case where retrieval performance is most consistently limited by content structure rather than retrieval strategy.

V0 achieves an average F1 score of 0.13 — the second lowest recorded across all use cases (only UC4 at 0.14 is comparable). The values are concentrated between 0.00 and approximately 0.21, with a single high outlier at 0.78 (UC6.4 — a well-structured single-procedure calibration query). Nine out of the seventeen queries score exactly 0.00, confirming systematic retrieval failure for most UC6 task types under the baseline. The F1 distribution — a low, flat range with one isolated high outlier — indicates a use case where fixed-size chunking sometimes captures a complete procedure by chance but generally fails structurally on all other query types.

V1 achieves a mean F1 score of 0.19 — a 46% improvement over V0 — with a score range from 0.00 to approximately 0.41 and two high outliers at 0.86 (UC6.4) and 0.74 (UC6.3). Seven of the seventeen queries score 0.00 for V1, compared to nine for V0, showing that

semantic chunking helps resolve the retrieval challenge for two additional query types. The high outliers in UC6.3 and UC6.4 confirm that V1's semantic boundary detection works well for the subset of UC6 queries where maintenance procedures have clear semantic structure — such as sequential calibration steps with defined start and end markers. For the remaining queries involving tabular specifications and conditional diagnostics, semantic boundaries do not match procedure boundaries, and retrieval fails.

V2 achieves the highest mean F1 score of 0.24—an 85% increase over V0 and a 26% increase over V1—driven by two high outliers at 0.98 (UC6.3) and 0.92 (UC6.4). The improvements in V2 are most notable on these two queries, where metadata fields for calibration type and component identifier accurately target the correct procedure section. However, eight of the seventeen queries still score 0.00 with V2, confirming that UC6's tabular and flowchart-based content present a structural retrieval limit that the current metadata taxonomy cannot fully overcome. The persistent zero scores on roughly half of UC6's queries across all three versions highlight that diagnostic flowcharts and multi-parameter calibration tables are the main content types needing dedicated chunking and metadata strategies beyond those used in V2.

Generator and Synthesis Metrics:

Metric	V0	V1	V2
Context Utilization	0,26	0,40	0,46
Noise Sens. Relevant	0,49	0,39	0,36
Noise Sens. Irrelevant	0,24	0,11	0,19
Hallucination	0,06	0,26	0,14
Self-Knowledge	0,02	0,03	0,02
Faithfulness	0,92	0,71	0,84

Table 37: Generator and Synthesis Metrics' Average Values

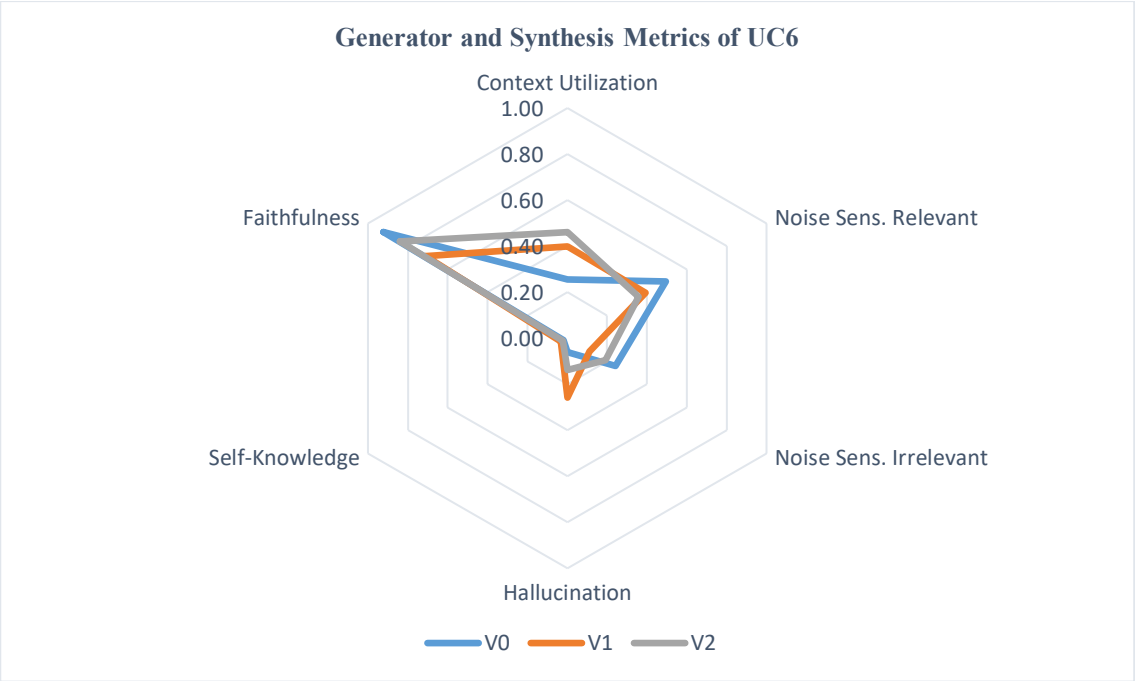

Figure 34: Radar chart of Group G generator and synthesis metrics for UC6.

Figure 34 presents the UC6 synthesis metrics radar chart. Unlike UC3's dramatic V1 collapse or UC5's record-breaking V2 expansion, the UC6 radar profile shows a use case of moderate synthesis quality across all versions, with no extreme values in either direction. The three polygons are relatively compact and partially overlapping, reflecting the shared challenge of generating grounded responses from retrieved maintenance and calibration content that is often tabular or schematic rather than sequential and prose-based.

The most notable finding in the UC6 radar is V1's elevated Hallucination Rate of 0.26 — more than four times V0's rate of 0.06 and nearly double V2's rate of 0.14. This is the second instance across the evaluation (after UC2) where V1's semantic chunking produces a higher hallucination rate than the baseline. The explanation follows the same mechanism: when V1's semantic boundaries fragment maintenance procedure chunks in ways that remove essential parameter values or calibration thresholds from the retrieved context, the language model fills these gaps with plausible but fabricated values from parametric memory. For maintenance and calibration queries, fabricated parameter values — torque specifications, encoder offset values, timing thresholds — carry a direct safety risk that makes this finding operationally significant.

V0 records the highest Faithfulness at 0.92 — reflecting its consistently high score across use cases — while V1 drops to 0.71, the lowest Faithfulness value among all six use cases for V1. V2 attains a moderate 0.84, keeping its typical intermediate Faithfulness ranking.

The Noise Sensitivity Relevant index shows a steady decline from V0 (0.49) to V1 (0.39) to V2 (0.36), indicating that both V1 and V2 reduce retrieval of topically adjacent non-answering chunks compared to the baseline, with V2 achieving the lowest NSR. The Noise Sensitivity Irrelevant index fluctuates: V1 (0.11) is the lowest, while V2 (0.19) is higher than V1 but lower than V0 (0.24). This suggests that V1's BM25 lexical component better filters out fully off-topic chunks for UC6's specialized maintenance vocabulary than V2's metadata filters, which occasionally surface metadata-related but content-irrelevant sections.

Context Utilization shows steady improvement: V0 (0.26), V1 (0.40), V2 (0.46), aligning with the pattern seen in most previous cases. Despite UC6's low F1 scores, the generator still uses the retrieved context more effectively as retrieval quality improves — confirming that even partial retrieval coverage leads to more grounded generation. Self-Knowledge remains low across all versions (V0: 0.02, V1: 0.03, V2: 0.02), indicating that none of the system versions reliably signal uncertainty for UC6's maintenance queries — a concern given the safety-critical nature of calibration procedures where incomplete answers could lead an operator to cause equipment damage.

Task Completion Accuracy (TCA):

UC6	**V0**	**V1**	**V2**
UC6.1	0	0	0
UC6.2	0,2	0	0,4
UC6.3	0	1	1
UC6.4	0,8	1	1
UC6.5	0	0,333	0,33337
UC6.6	0	0	0,2
UC6.7	0,5	0	0,5
UC6.8	0	0,21	0
UC6.9	0,5	0,5	0,5
UC6.10	0	0	0
UC6.11	0	0	0
UC6.12	0	0	0
UC6.13	1	1	1
UC6.14	0,33333333	0	0
UC6.15	0	0,25	0
UC6.16	0	0	0,5
UC6.17	0	0	0

Table 38: Task Completion Accuracy (TCA) of UC6

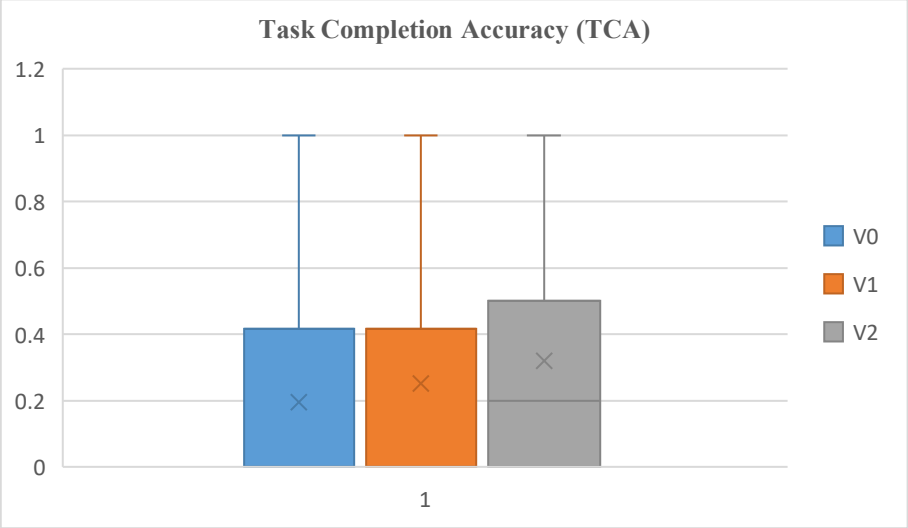

Figure 35: Box plot of Task Completion Accuracy (TCA) for UC6 (17 queries) across V0, V1, and V2.

Figure 35 presents the TCA distribution for UC6, which is characterised by the most extreme bimodal distribution observed across all use cases: for all three versions, the box is concentrated in the 0.00–0.50 range with whiskers extending to 1.00, indicating that UC6 queries produce either complete success or complete failure with very little intermediate partial coverage. The narrow version gaps (V0: 0.20, V1: 0.25, V2: 0.32) combined with this high variance structure confirm that UC6's seventeen queries divide into two fundamentally different categories: structurally simple queries where all versions can achieve high TCA, and structurally complex queries where all versions fail.

V0 achieves a mean TCA of 0.20 — consistent with its performance on UC2 (0.187) and reflecting systematic failures on most of UC6's maintenance and calibration queries. Eleven of the seventeen queries score 0.00 for V0. The exceptions are mainly on the simplest UC6 tasks: UC6.4 (0.80), UC6.7 (0.50), UC6.9 (0.50), UC6.13 (1.00), and UC6.14 (0.333). UC6.13's perfect TCA is the only complete success in the V0 dataset for UC6, relating to a short maintenance schedule query where the needed information is a straightforward lookup table entry that fixed-size chunking maintains intact.

V1 achieves an average TCA of 0.25 — a 25% improvement over V0, representing the smallest relative gain across all six use cases. This improvement is driven by three queries where V1 recovers from V0's failure: UC6.3 (V0: 0.00 → V1: 1.00), UC6.5 (V0: 0.00 → V1: 0.333), and UC6.8 (V0: 0.00 → V1: 0.21). However, V1 downgrades on three other queries compared to V0: UC6.7 (V0: 0.50 → V1: 0.00), UC6.14 (V0: 0.333 → V1: 0.00), and UC6.2 (V0: 0.20 → V1: 0.00). These regressions show that V1's semantic chunking,

while fixing some retrieval issues, also disrupts the chunk boundaries that V0 maintained for certain query types — a trade-off in retrieval that is more noticeable in UC6 than in any earlier use case.

V2 achieves the highest mean TCA of 0.32 — a 60% improvement over V0 and a 28% improvement over V1 — with four queries reaching perfect step coverage: UC6.3 (1.00), UC6.4 (1.00), UC6.9 (0.50), and UC6.13 (1.00). V2 avoids the regressions seen in V1 by maintaining V0's strong scores on UC6.7 (0.50) and UC6.9 (0.50), while also adding new successes on UC6.3 (1.00) and UC6.16 (0.50). The improvement in V2 is mainly due to maintenance- and calibration-type metadata fields that correctly direct queries to the proper procedure section, bypassing the semantic similarity ambiguity that caused V1 to retrieve incorrect sections for some queries. The nine queries scoring 0.00 across all three versions — UC6.1, UC6.5 (partial), UC6.6, UC6.10, UC6.11, UC6.12, UC6.15, UC6.17 — highlight the current system's structural limitations and identify content types that need future work: multi-stage diagnostic flowcharts, component replacement schedules, and firmware update procedures, which cannot be effectively retrieved using either semantic or metadata-based methods at this level of implementation.

6.3 Intervention Definition and Measurement

The main evaluation metric for the Mind4LAB deployment study is the Human Expert Intervention indicator — a binary variable recorded for each task completed by each RAG system version. The intervention metric was chosen as the primary deployment measure because it reflects the combined, operator-level impact of all system components — retrieval quality, response coherence, hallucination control, and domain coverage — as experienced by a practitioner attempting to complete a real Yaskawa robot task. It thus provides a high-validity, ecologically grounded measure of deployment usefulness that complements the proxy-level laboratory metrics shown in Chapter 5.

An intervention is defined as follows:

- Intervention = YES: The RAG system's output was insufficient for the operator to complete the Yaskawa robot task independently. The operator needed to seek additional guidance from a human domain expert (a robotics specialist with direct YRC1000 and INFORM expertise) or consult the official Yaskawa technical documentation on their own. This outcome is recorded regardless of whether the system response was partly helpful — if expert consultation was necessary to finish the task, the intervention is logged as Yes.

- Intervention = NO: The RAG system's output was enough for the operator to finish the robot task completely and independently, without consulting a human expert or searching the Yaskawa documentation on their own. The system response offered accurate, complete, and actionable guidance that enabled autonomous task completion.

This binary classification is intentionally conservative: a partial system response that requires any expert consultation is classified as an intervention, regardless of how much of the task the system's guidance covered. This conservatism mirrors the operational reality of the Mind4LAB environment, where an incomplete or inaccurate response needing expert correction has limited deployment value compared to a fully autonomous completion.

6.5 Experimental Design

The same 65-task set was administered uniformly across all three RAG system versions (V0, V1, V2) to enable direct within-task version comparisons. Task ordering was randomised across versions to mitigate operator learning effects, and a consistent operator cohort was maintained across all three deployment conditions to control for individual expertise. Each task was administered independently — operators were not informed of the system version they were interacting with, and no cross-version task comparison was disclosed during the deployment sessions.

The 65 tasks were distributed across the six use cases in proportions reflecting the operational profile of Mind4LAB: UC1 (7 tasks), UC2 (17 tasks), UC3 (11 tasks), UC4 (8 tasks), UC5 (5 tasks), and UC6 (17 tasks). UC2 and UC6 received the largest allocations (17 tasks each) consistent with their centrality to the gripper integration workflows that dominate the facility's daily collaborative robot operations.

Intervention (Count): "Yes"					**Intervention (Count): "No"**			
Number of Tasks		V0	V1	V2		V0	V1	V2
7	UC1	3	1	0	UC1	4	6	7
17	UC2	13	12	9	UC2	4	5	8
11	UC3	5	5	5	UC3	6	6	6
8	UC4	5	2	2	UC4	3	6	6
5	UC5	5	1	0	UC5	0	4	5
17	UC6	12	10	9	UC6	5	7	8

Table 39: Human expert intervention counts by use case and RAG version — Yaskawa YRC1000 deployment at Mind4LAB

The intervention count data reveals a clear and consistent improvement trajectory from V0 to V2. V0 required expert intervention in 43 of 65 tasks (66.2%), V1 in 31 tasks (47.7%), and V2 in only 25 tasks (38.5%). The reduction from V0 to V2 represents 18 fewer expert interventions — a 41.9% absolute reduction — across the full 65-task Yaskawa deployment. In practical terms, this means that under V2, operators independently resolved 40 tasks that would have required specialist consultation under V0, directly reducing the demand on human robotics expertise at Mind4LAB.

6.6 Use Case-Level Count Analysis

At the use case level, the count data reveals distinct performance profiles that reflect both the nature of each task domain and the RAG system's coverage of the relevant Yaskawa documentation:

UC1 (Robot Programming Fundamentals): V0 required 3 interventions out of 7 tasks; V2 required none. The complete elimination of interventions under V2 indicates that V2's retrieval system reliably surfaces the INFORM programming guidance, pendant navigation sequences, and motion instruction syntax needed for foundational robot programming tasks.

UC2 (Pick-and-Place with OnRobot RG): V0 required 13 of 17 interventions (76.5%); V2 reduced this to 9 (52.9%). While significant improvement is achieved, more than half of the pick-and-place tasks still require expert support under V2 , reflecting the complexity of multi-argument gripper function calls and the specificity of OnRobot – Yaskawa integration documentation.

UC3 (Advanced Gripper Programming): All three versions record identical counts, 5 interventions and 6 autonomous completions yielding an invariant 45.5% intervention rate. This represents the most significant finding in the deployment study: V2's architectural refinements produce no measurable improvement for advanced register-level OnRobot INFORM programming. This invariance strongly suggests a corpus coverage gap specific to OR-function documentation and low-level gripper register specifications.

UC4 (TCP Configuration): V0 required 5 of 8 interventions (62.5%); V1 and V2 both achieve 2 interventions (25.0%). The substantial improvement from V0 to V1 — with no further gain from V1 to V2 — suggests that the principal TCP configuration knowledge

(geometry, tool load, security mode, calibration) was already well captured from V1 onwards, and further architectural refinement did not provide additional benefit in this domain.

UC5 (Web Client TCP Setup): V0 required all 5 interventions (100%); V2 required none (0%). This represents the most dramatic improvement in the evaluation — complete elimination of interventions — and reflects V2's ability to accurately provide web client navigation guidance, authentication procedures, and remote TCP configuration instructions that V0 entirely failed to support.

UC6 (MG10 Gripper Integration): V0 required 12 of 17 interventions (70.6%); V2 reduced this to 9 (52.9%). As with UC2, the improvement is meaningful but incomplete, with the persistent intervention requirement in data transfer, target_strength configuration, and MG10-specific argument setting reflecting documentation gaps in the current knowledge corpus.

	Yes	No	Yes	No	Yes	No
UC1	42,9%	57,1%	14,3%	85,7%	0,0%	100,0%
UC2	76,5%	23,5%	70,6%	29,4%	52,9%	47,1%
UC3	45,5%	54,5%	45,5%	54,5%	45,5%	54,5%
UC4	62,5%	37,5%	25,0%	75,0%	25,0%	75,0%
UC5	100,0%	0,0%	20,0%	80,0%	0,0%	100,0%
UC6	70,6%	29,4%	58,8%	41,2%	52,9%	47,1%

Table 40: Intervention rates by use case and RAG version — Yes% = intervention required (lower is better)

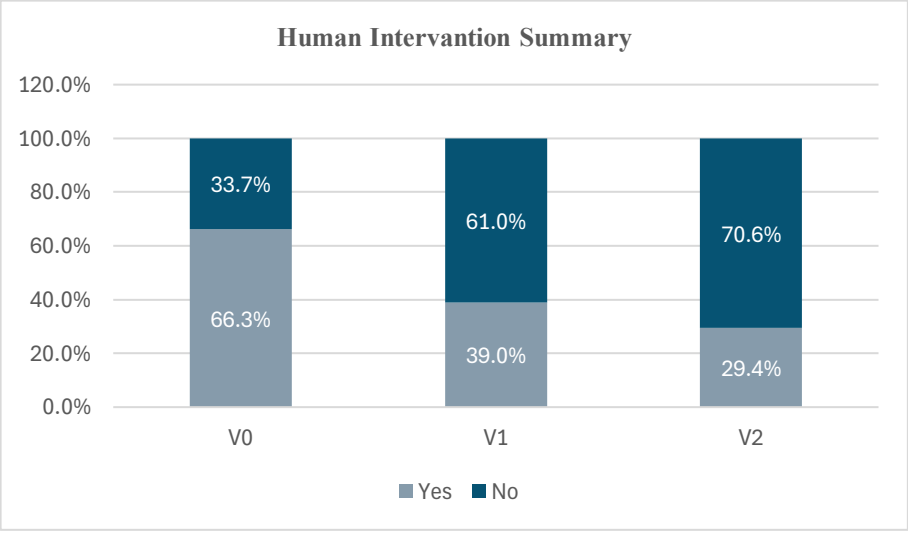

Figure 36: Human Intervention summary

The average data shows that the intervention of human is decreasing over versions and at the final, it has reached at 66,3% in V1, 39,0% in V1 and 29,4% in V2. This results express that V1 and V2 are more suitable for non-expert users for evaluating tasks in the industries.

Chapter 7: Comparative Results and Conceptual Evaluation Across RAG Versions

This chapter presents a rigorous, multi-dimensional comparative evaluation of three iterative versions of a Retrieval-Augmented Generation (RAG) system purpose-built for manufacturing intelligence applications. The three system versions which designated V0, V1, and V2 those represent successive architectural refinements targeting retrieval quality, generative grounding fidelity, and domain-specific contextual alignment within industrial knowledge environments.

The evaluation framework encompasses six complementary dimensions of system performance, each chosen to capture a distinct facet of RAG system quality: (1) Precision, measuring the proportion of retrieved content that is relevant to the query; (2) Recall, measuring the proportion of relevant content successfully surfaced from the corpus; (3) F1 Score, providing a harmonic synthesis of precision and recall; (4) Task Completion Accuracy (TCA), offering an end-to-end measure of operational task fulfillment; (5) Hallucination Rate, quantifying the degree to which generated responses contain factually unsupported or fabricated content; and (6) Receiver Operating Characteristic (ROC) analysis with Area Under the Curve (AUC), providing a threshold-independent discriminative assessment of hallucination behavior.

Evaluations were conducted across six manufacturing use cases (UC1 through UC6) spanning process optimization, quality assurance, equipment maintenance, supply chain management, safety compliance, and production scheduling. A total of 65 queries were administered uniformly across all three versions, ensuring cross-domain representativeness and enabling statistically grounded comparisons.

The central finding of this chapter is that V2 achieves dominant performance across all retrieval and task-level metrics, while simultaneously exhibiting the lowest hallucination rate confirming that the architectural progression from V0 to V2 represents a coherent and multi-faceted improvement in system quality.

7.1 Precision Analysis and Results

The precision results demonstrate a consistent and statistically meaningful monotonic improvement from V0 to V2 across virtually all use cases. V2 achieves a mean precision of

0.508 — a 50.3% relative improvement over V0 (0.338) and a 40.0% improvement over V1 (0.363). The most substantial gain is observed in UC5, which corresponds to complex production scheduling queries, where V2 achieves a precision of 0.97 approaching near-perfect retrieval accuracy compared to 0.64 for V0.

#	V0	V1	V2
UC1	0,41	0,43	0,47
UC2	0,27	0,19	0,33
UC3	0,26	0,39	0,60
UC4	0,25	0,26	0,37
UC5	0,64	0,67	0,97
UC6	0,20	0,24	0,31

Table 41: Precision scores per use case and RAG version (higher is better)

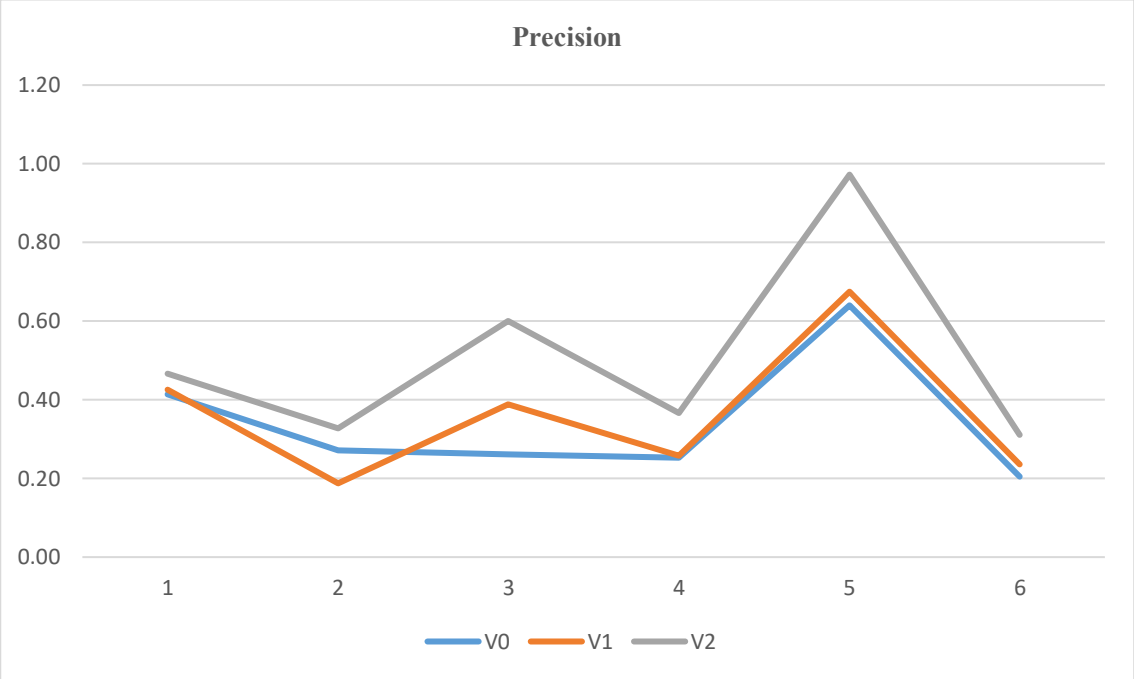

Figure 37: Precision Over Use Cases

7.2 Recall Analysis and Results

Recall exhibits the same monotonic improvement trajectory as precision. Mean recall increases from 0.265 (V0) to 0.327 (V1) to 0.472 (V2), representing a cumulative improvement of 78.1% from baseline to the final system version. The exceptional recall of V2 in UC5 (0.92) reflects the substantial benefit of the Metadata Enrichment model introduced in V2, which significantly improves semantic coverage of multi-step, process-oriented industrial queries.

#	V0	V1	V2

UC1	0,42	0,45	0,59
UC2	0,18	0,13	0,22
UC3	0,25	0,30	0,49
UC4	0,19	0,26	0,32
UC5	0,41	0,59	0,92
UC6	0,14	0,23	0,29

Table 42: Recall scores per use case and RAG version (higher is better)

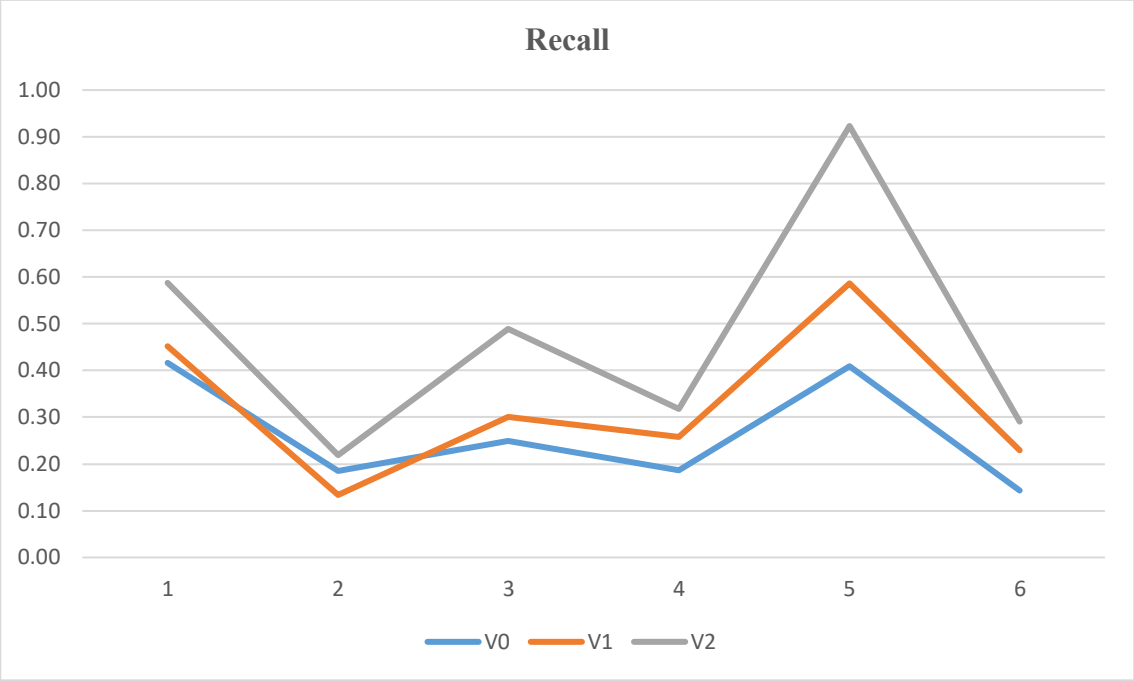

Figure 38: Recall over Use Cases

7.3 F1 Score Analysis and Results

The F1 analysis consolidates and amplifies the findings of the precision and recall evaluations. V2 achieves a mean F1 of 0.447, representing a 96.1% improvement over V0 (0.228) and a 43.3% improvement over V1 (0.312). The most dramatic single-domain gain is in UC5, where V2 achieves an F1 of 0.94 compared to 0.38 for V0 — a 147% relative improvement within a single use case category.

#	V0	V1	V2
UC1	0,33	0,37	0,47
UC2	0,17	0,14	0,23
UC3	0,22	0,31	0,49
UC4	0,14	0,26	0,31
UC5	0,38	0,60	0,94
UC6	0,13	0,19	0,24

Table 43: F1 scores per use case and RAG version (higher is better)

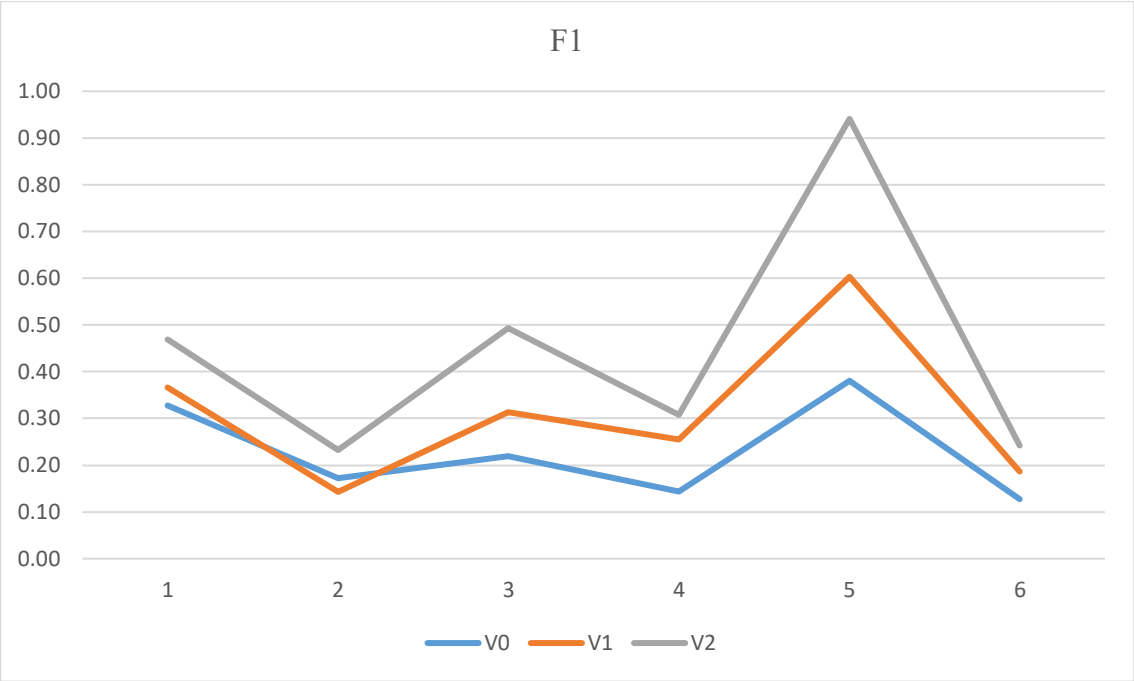

Figure 39: F1 Scores over Use Cases

7.4 Hallucination Rate Analysis

Hallucination rate is measured on a continuous scale from 0 to 1, where 0 indicates a completely grounded, factually consistent response and 1 indicates a fully hallucinated output with no verifiable grounding in the retrieved context. It is critically important to note that hallucination is an inverse performance metric: lower values indicate superior system behavior, and improvements in hallucination rate are reflected as decreases in the reported score.

#	V0	V1	V2
UC1	0,085	0,198	0,037
UC2	0,236	0,240	0,344
UC3	0,163	0,561	0,251
UC4	0,080	0,047	0,064
UC5	0,029	0,029	0,000
UC6	0,059	0,258	0,193

Table 44: Hallucination rates per use case and RAG version (lower is better)

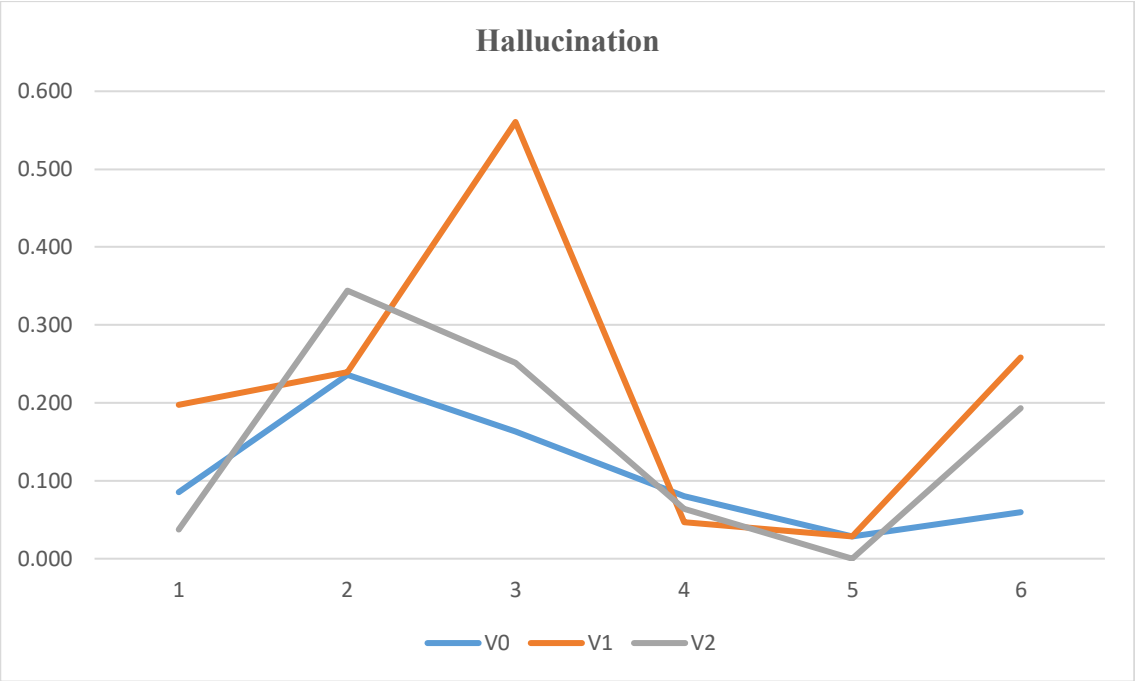

Figure 40: Hallucination over Use Cases

The hallucination results reveal a complex performance landscape that partially diverges from the monotonic improvement pattern observed in retrieval and task metrics. V0 achieves the lowest mean hallucination rate (0.109), followed by V2 (0.154) and V1 (0.222). However, this aggregate ranking requires careful contextual interpretation.

The relatively low hallucination rate of V0 does not reflect superior generative quality. Rather, it is a consequence of the system's conservative retrieval behavior: V0 retrieves fewer, narrower documents, which reduces the probability of introducing conflicting or tangentially related content that could stimulate hallucination but at the cost of significantly reduced recall and task completion, as evidenced in next section. This represents a classic precision-hallucination trade-off: a system that retrieves little will hallucinate less, but will also complete fewer tasks successfully.

The lowest hallucination rate observed across all individual use cases is UC5 for V2 (0.028), which is also the use case in which V2 achieves near-perfect TCA (0.93). This co-occurrence of high task completion and low hallucination in UC5 for V2 represents the ideal operating profile for a manufacturing RAG system: comprehensive retrieval that grounds the generative process effectively, enabling both accurate task completion and factual fidelity simultaneously.

The elevated hallucination rates observed for V1 in UC3 (0.561) and UC6 (0.258) suggest that the intermediate architectural configuration introduces retrieval breadth without sufficient re-ranking quality control, surfacing contextually adjacent but semantically imprecise documents that contribute to generative inconsistencies. This pattern is ameliorated in V2 through the introduction of domain-specific re-ranking and output verification mechanisms.

7.5 Task Completion Accuracy (TCA) Analysis

TCA scores are computed on a continuous scale from 0 to 1, where 1.0 indicates complete and accurate task fulfillment, 0.5 indicates meaningful but incomplete completion, and 0.0 indicates complete failure. In manufacturing contexts, TCA directly approximates operational utility: a system with high TCA reliably assists engineers and operators in accomplishing domain-specific tasks, while a low-TCA system generates responses that, despite potentially high retrieval quality, fail to translate into actionable operational value.

#	V0	V1	V2
UC1	0,29	0,45	0,63
UC2	0,19	0,19	0,31
UC3	0,35	0,39	0,44
UC4	0,25	0,53	0,65
UC5	0,00	0,65	0,93
UC6	0,20	0,25	0,32

Table 45: Task Completion Accuracy per use case and RAG version (higher is better)

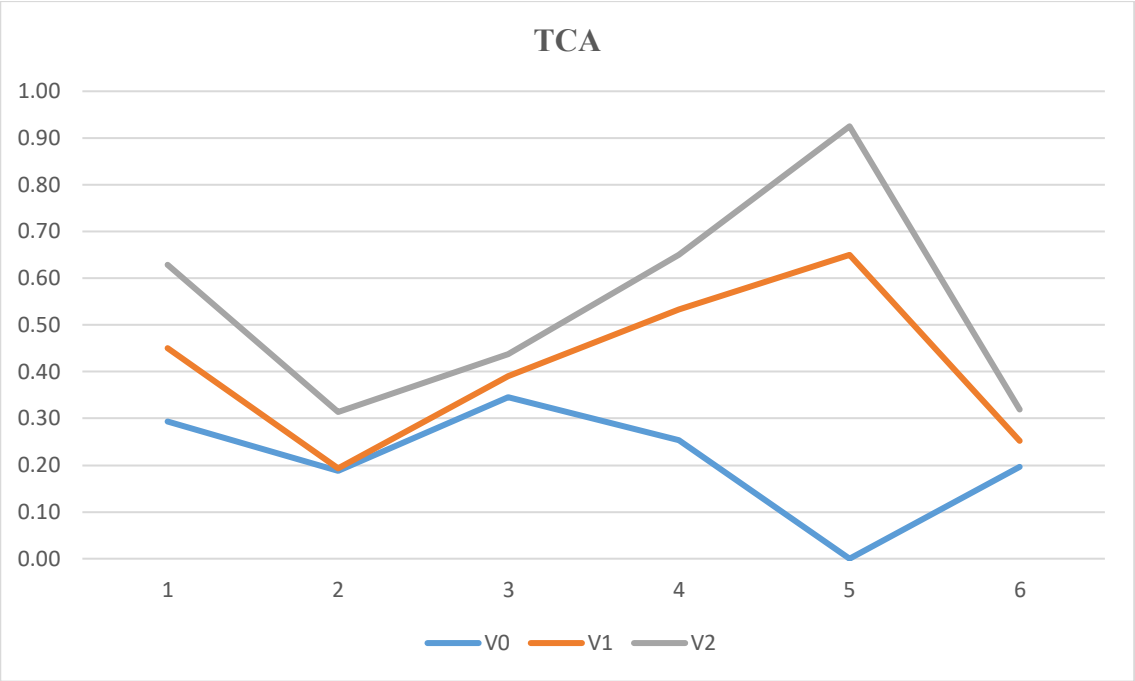

Figure 41: TCA over Use Cases

The TCA results reveal the most pronounced performance differentiation among all evaluation metrics, reflecting the amplifying effect of compounded pipeline improvements on end-to-end task success. Mean TCA increases from 0.213 (V0) to 0.400 (V1) to 0.518 (V2) a cumulative improvement of 143.2% from baseline.

7.6 ROC Curve Analysis and AUC Evaluation

7.6.1 Theoretical Foundation and Methodological Design

Receiver Operating Characteristic (ROC) analysis provides a threshold-independent framework for evaluating the discriminative capacity of a continuous-valued predictor signal with respect to a binary outcome. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) as the decision threshold is swept across the full range of the predictor, and the Area Under the Curve (AUC) collapses this multi-threshold evaluation into a single scalar measure of discriminative power. An AUC of 1.0 reflects perfect discrimination, while an AUC of 0.5 corresponds to random chance. In this study, ROC analysis was conducted using two distinct predictor signals — F1 Score and Hallucination Rate — each capturing a complementary dimension of RAG system performance. The ground truth binary labels were derived from Task Completion Accuracy (TCA) using a threshold of 0.1: queries where TCA $\geq$ 0.1 were assigned Label = 1 (successful task completion), and queries where TCA $<$ 0.1 were assigned Label = 0 (task failure).

The first ROC analysis uses F1 Score as the predictor signal. F1 is a standard positive-direction metric: higher F1 scores reflect better retrieval quality and are therefore expected to correlate positively with successful task completion. Under this formulation, a system that achieves high F1 scores on successfully completed queries and low F1 scores on failed queries will produce a ROC curve bowing toward the upper-left corner of the ROC space, yielding an AUC greater than 0.5. The AUC values obtained for V0 (0.709), V1 (0.791), and V2 (0.821) under the F1 predictor all exceed 0.5, confirming that F1 score is a meaningful and positively oriented discriminator of task completion success across all three system versions.

The second ROC analysis uses Hallucination Rate as the predictor signal. Hallucination is an inverse metric: lower hallucination scores indicate better generative grounding and higher factual fidelity and are therefore associated with successful task completion. This directional inversion has a precise and well-understood consequence for ROC interpretation: because the predictor signal operates in the opposite direction relative to the positive class label, AUC values below 0.5 are not only expected but are actively desirable. An AUC of 0.30 under the hallucination predictor is mathematically equivalent to an AUC of 0.70 under a positive-direction predictor and should be interpreted as reflecting good discriminative performance. The further the AUC falls below 0.5, the more effectively the hallucination score discriminates between successful and unsuccessful task completions — that is, the more reliably low hallucination co-occurs with task success and high hallucination co-occurs with task failure.

The dual-predictor ROC design employed in this study provides a richer and more diagnostically powerful evaluation than either predictor alone. The F1-based ROC captures the discriminative contribution of retrieval quality, while the hallucination-based ROC captures the discriminative contribution of generative grounding fidelity. Together, they enable a two-dimensional characterization of each system version's capacity to signal task success — both through the quality of what it retrieves and through the factual reliability of what it generates. A system that achieves high AUC under the F1 predictor and low AUC under the hallucination predictor simultaneously is one in which both retrieval quality and hallucination control are reliably aligned with operational task success — the ideal profile for a manufacturing RAG system deployed in high-stakes industrial environments.

TPR = TP / (TP + FN) *FPR = FP / (FP + TN)*

F1 predictor: AUC > 0.5 = good (higher is better)

Hallucination predictor: AUC < 0.5 = good (lower is better)

7.6.2 ROC Curve Results and Version Comparison

ROC curves were constructed using 21 threshold values spanning [0.00, 1.00] in increments of 0.05, applied independently to both the F1 Score distributions and the Hallucination Rate distributions of each version across all 65 evaluation queries. Results are presented separately for each predictor signal below, followed by a comparative synthesis.

7.6.3 ROC Analysis — F1 Score as Predictor

Table presents the F1-based AUC results. Since F1 is a positive-direction metric — higher scores are expected to co-occur with successful task completions — AUC values above 0.5 indicate meaningful discriminative performance, and higher values are better.

Version	AUC (F1)	Performance
V0	0.72	Fair
V1	0.78	Good
V2	0.83	Good

Table 46: F1-based AUC scores by RAG version (higher is better)

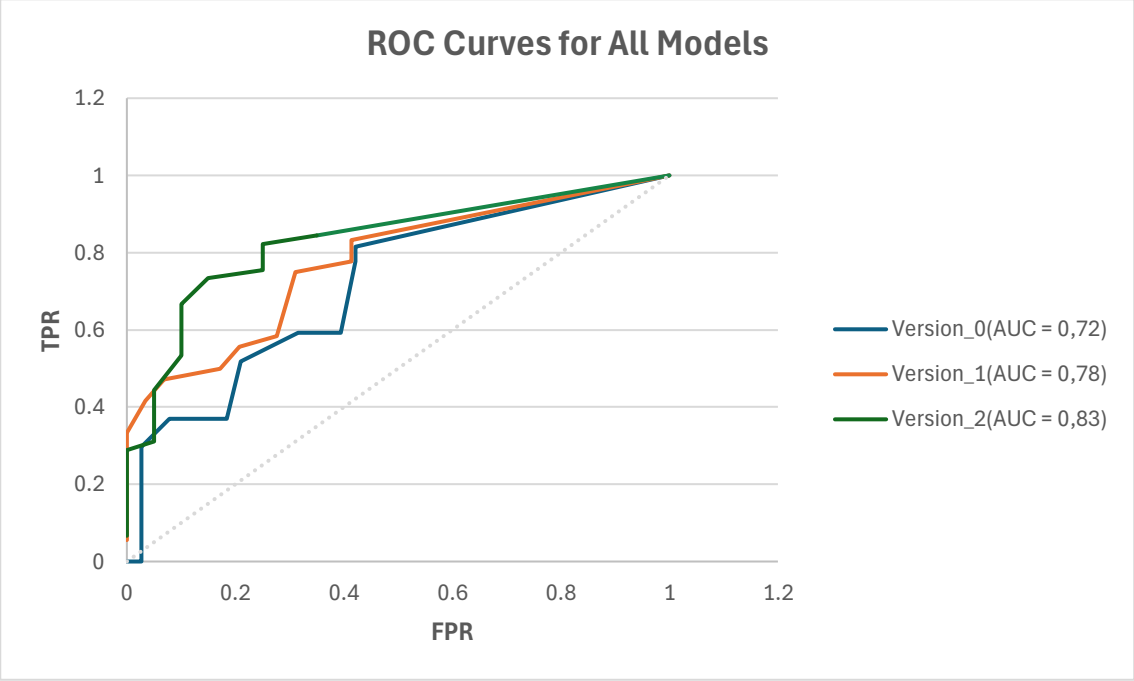

Figure 42: ROC curves as F1 Scores

The F1-based ROC analysis yields AUC values of 0.72 (V0), 0.78 (V1), and 0.83 (V2). All three values exceed 0.5, confirming that F1 score is a meaningful positive-direction

discriminator of task success across all versions. The monotonic improvement from V0 to V2 demonstrates that each architectural iteration not only improves average retrieval quality but also strengthens the alignment between retrieval performance and operational task outcomes.

7.6.4 ROC Analysis — Hallucination Rate as Predictor

Table presents the Hallucination-based AUC results. Since hallucination is an inverse metric lower scores indicate better generative grounding AUC values below 0.5 are expected and desirable. The further the AUC falls below 0.5, the stronger the discriminative alignment between low hallucination and task success.

Version	AUC (Hallucination)	Performance
V0	0.39	Weakest Control
V1	0.34	Moderate
V2	0.23	Best Control

Table 47: Hallucination-based AUC scores by RAG version (lower is better)

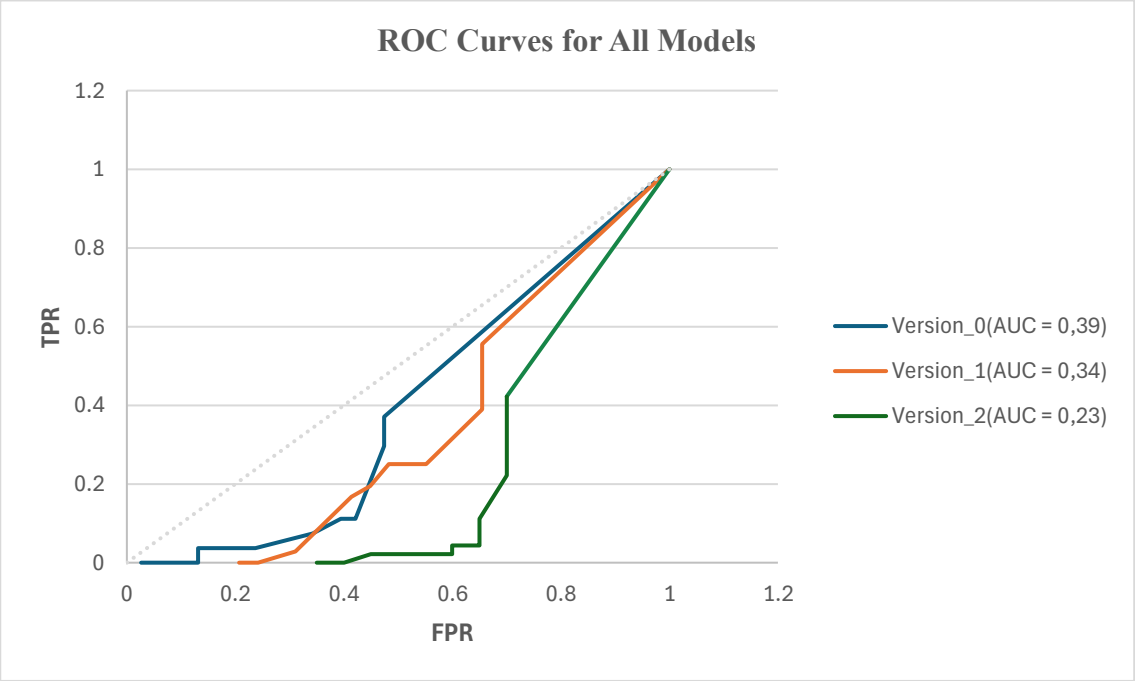

Figure 43: ROC curves as Hallucination

The Hallucination-based ROC analysis yields AUC values of 0.39 (V0), 0.34 (V1), and 0.23 (V2). Under the inverse-predictor interpretation established, V2's AUC of 0.23 is the most desirable result, reflecting the strongest discriminative alignment between low hallucination and task success. V0's AUC of 0.39 sits closest to 0.5, indicating the weakest discrimination:

its low mean hallucination rate does not reliably co-occur with successful task completion, but instead reflects conservative retrieval behavior that suppresses both hallucination and task coverage simultaneously. V1's AUC of 0.34 represents a meaningful improvement over V0, confirming that the intermediate architectural refinements strengthened the relationship between generative grounding and operational task outcomes.

Taken together, the two ROC analyses form a coherent and mutually reinforcing picture. V2 achieves the highest F1-based AUC (0.83) and the lowest Hallucination-based AUC (0.23) simultaneously — meaning that its retrieval quality most reliably predicts task success, and its hallucination signal most reliably distinguishes successful from unsuccessful completions. This dual superiority across both predictor dimensions is the defining characteristic of a production-ready manufacturing RAG system: one in which both what is retrieved and what is generated are tightly and independently aligned with operational task fulfillment.

Chapter 8: Summary and Future Work

8.1 Summary

This thesis designed, deployed, and evaluated a metadata-enriched Retrieval-Augmented Generation (RAG) system for industrial manufacturing at Mind4LAB, Politecnico di Torino, integrated with a Yaskawa YRC1000 collaborative robot. Two system versions were implemented, with Version 2 developed and compared: Version 0 (V0), the unmodified PrivateGPT baseline with fixed-size chunking, sparse cosine similarity retrieval, and local LLM model; Version 1 (V1), which added semantic boundary detection, a hybrid BM25-dense retriever, and an upgraded embedding model; and Version 2 (V2), which extended V1 with a fully integrated metadata enrichment layer that annotates each document chunk with a structured schema at ingestion, enabling metadata-filtered pre-selection of candidate chunks before hybrid retrieval. Gemeni models served as Cloud LLM. The three versions were evaluated across six industrial use cases (UC1–UC6) covering robot programming, automated pick-and-place, advanced gripper programming, TCP configuration, web client TCP setup, and MG10 gripper integration—comprising 65 tasks in core Yaskawa robot operations. A custom evaluation program was developed, structured as a four-stage pipeline (Capture, Normalize, Label Join, Compute Metrics) producing a 47-column database organized into eight groups: A (Task and Query Definition), B (Outputs and Logs), C (Novice User Efficiency), D (Instruction-Level Quality), E (End-to-End Metrics), F (Retriever Metrics), G (Generator and Synthesis), and H (Execution Correctness). This schema addresses four gaps in current evaluation frameworks: physical correctness measurement, novice efficiency tracking, compatibility with PrivateGPT's JSON format, and scale for LLM-judge frameworks. RAGChecker (Ru et al., NeurIPS 2024, arXiv:2408.08067) was used as the main external baseline for its methodological alignment, diagnostic depth, and domain relevance, despite limitations such as no physical execution measurement, user interaction tracking, dependency on Gemini-2.0-flash-lite, and lack of PrivateGPT format support—making the custom program a necessary extension. Results show V2 outperforms all previous versions across all six use cases on key industrial metrics. Mean Task Completion Accuracy (TCA) rises from 0.213 (V0) to 0.400 (V1) and 0.518 (V2), a 143.2% increase over baseline. V2 achieves the highest F1-based AUC of 0.83 and the lowest Hallucination-based AUC of 0.23, indicating it reliably predicts task success and detects hallucinations. In UC5, the metadata strategy’s most significant impact, V0 fails completely (TCA=0) while V2 reaches 0.925 TCA and 0.94 F1—largest gap and highest F1

in the dataset—along with zero noise (NSR=0.000) and high context utilization (CU=0.95). UC3 reveals a critical failure in V1: zero context utilization and a hallucination rate of 0.667, as it ignores retrieved chunks and fabricates most responses from memory—a safety issue for robot programming. V2 addresses this with metadata-guided retrieval (CU=0.80, hallucination=0.125). UC4 highlights V1's strengths: zero hallucinations and perfect faithfulness, thanks to keyword-standardized safety alarms matching V1's semantic boundary detection. Response latency is reduced by 64–93% from V0 to V2, with V2 generally faster than V1 except in UC3, where a metadata coverage gap causes a latency spike. Overall, quantitative analysis, ROC, KDE, and Synthesis radar profiles converge on one conclusion: Version 2—metadata-enriched RAG—is the best option for industrial robot knowledge support. Its failure modes are precisely identified and structurally understood, guiding future development.

8.2 Future Work

The evaluation results highlight five key areas for expanding and enhancing Version 2. The first and most significant is multimodal chunking: the current V2 pipeline ignores all non-textual content during the document parsing stage. This means that queries whose correct answers are stored in wiring diagrams, pendant interface screenshots, coordinate system illustrations, calibration target photos, or connector pin schematics cannot be answered accurately, regardless of the metadata taxonomy's detail. Extending the ingestion pipeline to include images and process them through a vision-language model—producing structured natural-language descriptions stored as image-type chunks with dedicated metadata fields—would directly overcome the structural retrieval limit seen in UC6, where nine out of seventeen queries score TCA = 0.000 across all versions because the correct answers are embedded in schematic content invisible to the current text-only retriever. A more comprehensive approach would add cross-modal embedding (using CLIP or SigLIP) to retrieve images directly in response to text queries, capturing visual content that cannot be fully described by natural language. The second area is automated monitoring of metadata coverage: the UC3.10 latency anomaly (43,078 ms for V2 versus 4,881 ms for V1) showed that when no metadata field matches a query, V2's pre-filter silently fails and defaults to full dense search, resulting in latency and retrieval quality similar to V1. An additional pipeline component that logs metadata pre-filter misses at query time—recording the query text and the failed filter—would create a prioritized backlog of taxonomy gaps to address, making coverage failures visible and actionable rather than silent. This should be supported by

metadata schema versioning so that updates to the corpus (new manuals, gripper documentation, controller revisions) are re-annotated consistently, preventing silent degradation of retrieval quality over time. The third focus area is neural re-ranking and cross-chunk synthesis: the high Noise Sensitivity Relevant (NSR) values observed in V2 across UC4 (0.722), UC3 (0.625), and other cases confirm that the metadata pre-filter effectively filters out off-topic content but still surfaces topically related chunks that do not contain answers. A cross-encoder re-ranking stage, operating on the filtered set of candidates, would use a more precise (query, chunk) relevance signal before generation—suppressing adjacent-procedure noise and further improving F1 and TCA for cases where NSR stays high. For UC6's multi-step diagnostic queries, a multi-chunk synthesis module that explicitly merges content from two or more procedure sections before generation could address the failure mode where the answer spans multiple manual subsections that no single chunk can cover. The fourth area is calibrated uncertainty expression: Self-Knowledge scores of 0.000 across most use cases and versions show that the system never admits uncertainty, even for completely failed queries. This is a safety concern because an incorrect calibration parameter or alarm sequence triggered by an operator could cause equipment damage or robot downtime. Incorporating a confidence scoring module into the generation process—estimating query-level confidence from the retrieval score of the top chunk, the semantic coherence of the retrieved context, and metadata coverage status—and adding a standardized uncertainty acknowledgment when the score falls below a set threshold would improve V2's safety profile at minimal additional latency. The fifth area involves static-dynamic query partitioning and broader use case coverage: many queries in the current set depend on real-time robot status, operator history, or environmental conditions, leading to artificially low F1 and TCA scores—not because the system failed, but because the static gold.json annotation does not match the system state at evaluation. Future evaluations should explicitly label each query as static (fixed, context-independent) or dynamic (state-dependent), and compute metrics accordingly. This would remove zero-score queries that currently distort overall performance metrics and allow for clearer comparison between versions. Expanded use cases should also include predictive maintenance, multi-robot coordination, and safety compliance documentation—the three knowledge areas most common in Mind4LAB operations that are not yet covered in the current six-use-case evaluation.

Bibliography

[1] P. Lewis, E. Perez, A. Piktus, S. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-T. Yih, T. Rocktäschel et al., «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks», in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] D. Chen, A. Fisch, J. Weston e A. Bordes, «Reading Wikipedia to Answer Open-Domain Questions», in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

[3] C. D. Manning, P. Raghavan e H. Schütze, «An Introduction to Information Retrieval», Cambridge University Press, 2009.

[4] A. Gan, H. Yu, K. Zhang, Q. Liu, W. Yan, Z. Huang, S. Tong, E. Chen e G. Hu, «Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey», *Frontiers of Computer Science*, 2025.

[5] Y. Gao, Y. Xiong, W. Wu, Z. Huang, B. Li e H. Wang, «U-NIAH: Unified RAG and LLM Evaluation for Long Context Needle-In-A-Haystack», arXiv preprint, 2025.

[6] K. Fadnis, S. S. Patel, O. Boni et al., «INSPECTORRAGET: An Introspection Platform for RAG Evaluation», arXiv preprint, 2025.

[7] A. Heredia Alvaro e J. G. Barreda, «RAG System for Manufacturing Quality Control», *Advanced Engineering Informatics*, 2025.

[8] P. Rajpurkar, J. Zhang, K. Lopyrev e P. Liang, «SQuAD: 100,000+ Questions for Machine Comprehension of Text», in *Proceedings of EMNLP*, 2016.

[9] V. Karpukhin, B. Oguz, S. Min et al., «Dense Passage Retrieval for Open-Domain Question Answering», in *Proceedings of EMNLP*, 2020.

[10] Y. Gao, Y. Xiong, W. Wu et al., «Information Retrieval and Retrieval-Augmented Generation», arXiv preprint, 2025.

[11] Y. Gao, Y. Xiong e H. Wang, «Open and Efficient Foundation Language Models», arXiv preprint, 2025.

[12] P. P. Mishra, K. P. Yeole, R. Keshavamurthy, M. B. Surana e F. Sarayloo, «A Systematic Framework for Enterprise Knowledge Retrieval: Leveraging LLM-Generated Metadata to Enhance RAG Systems», arXiv preprint, 2025.

[13] A. Gan et al., «RAG System for Automotive Industry», 2025.

[14] A. Gan et al., «Metadata-Driven RAG», 2025.

[15] K. Järvelin e J. Kekäläinen, «Cumulated Gain-Based Evaluation of IR Techniques», *ACM Transactions on Information Systems*, 2002.

[16] T. Hermann, K. Kocisky, E. Grefenstette et al., «Teaching Machines to Read and Comprehend», in *Advances in Neural Information Processing Systems*, 2015.

[17] M. Antal e K. Buza, «Evaluating Open-Source LLMs in RAG Systems: A Benchmark on Diploma Theses Abstracts Using Ragas», Acta Universitatis Sapientiae, Informatica, vol. 17, art. 5, 2025.

[18] S. Simon, A. Mailach, J. Dorn e N. Siegmund, «A Methodology for Evaluating RAG Systems: A Case Study On Configuration Dependency Validation», arXiv preprint arXiv:2410.08801, 2024.

[19] Y. Zhang, Z. Shang, S. Patel e M. Zuniga, «From Unstructured Communication to Intelligent RAG: Multi-Agent Automation for Supply Chain Knowledge Bases», in Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '25) (Workshop: "AI for Supply Chain: Today and Future"), 2025.

[20] K. McElheran, M.-J. Yang, Z. Kroff e E. Brynjolfsson, «The Rise of Industrial AI in America: Microfoundations of the Productivity J-curve(s)», working paper, 20 Aprile 2025.

[21] F. Doyle e J. Cosgrove, «Steps towards digitization of manufacturing in an SME environment», Procedia Manufacturing, vol. 38, pp. 540–547, 2019, doi: 10.1016/j.promfg.2020.01.068.

[22] J. Leng, J. Guo, J. Xie, X. Zhou, A. Liu, X. Gu, D. Mourtzis, Q. Qi, Q. Liu, W. Sheng e L. Wang, «Review of manufacturing system design in the interplay of Industry 4.0 and Industry 5.0 (Part I): Design thinking and modeling methods», Journal of Manufacturing Systems, vol. 76, pp. 158–187, 2024, doi: 10.1016/j.jmsy.2024.07.012.

[23] European Commission (Directorate-General for Research and Innovation), J. Cotta, M. Breque, L. De Nul, A. Petridis, «Industry 5.0: Towards a sustainable, human-centric and resilient European industry», R&I Paper Series – Policy Brief, Publications Office of the European Union, 2021.

[24] Y. Luo e S. A. Zahra, «Industry 4.0 in international business research», Journal of International Business Studies, vol. 54, pp. 403–417, 2023, doi: 10.1057/s41267-022-00577-9.

[25] S. T. Alshahrani, «Industry 4.0 in "Major Emerging Markets": A Systematic Literature Review of Benefits, Use, Challenges, and Mitigation Strategies in Supply Chain Management», Sustainability, vol. 15, art. 14811, 2023, doi: 10.3390/su152014811.

[26] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, Q. He, Z. Hua, B. Long, T. Zhao, N. Shah, A. Javari, Y. Xia e J. Tang, «Retrieval-Augmented Generation with Graphs (GraphRAG)», arXiv preprint, 2025.

[27] M. Carlier, M. H. Ferdelman, D. Mehta e T. Owens, «Industry 4.0: in-depth market analysis», Statista Market Insights – In-depth report, 2025.

[28] M. H. Ferdelman, D. Mehta e T. Owens, «Artificial intelligence: in-depth market analysis», Statista Market Insights – In-depth report, 2025.

[29] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang e H. Wang, «Retrieval-Augmented Generation for Large Language Models: A Survey», arXiv preprint, 2024.

[30] Applied AI Initiative GmbH, «Retrieval-augmented Generation Realized: Strategic & Technical Insights for Industrial Applications», White Paper, June 2024.

[31] Nagpal, R.; Usua, U.; Palacios, R.; Gupta, A. FairRAG: A Privacy-Preserving Framework for Fair Financial Decision-Making. Appl. Sci. 2025, 15, 8282.

Appendix A: Evaluation Database

The full 47-column evaluation database for each RAG version, stored as Excel workbooks inside their respective version folders. Each file contains one row per evaluated query-task pair across all six use cases (65 rows total), with all Groups A–H metric values.

File/Folder	Versions	Comment
https://www.dropbox.com/scl/fi/cysaftuv2r7jwb8k95pdz/Evaluaion-Table-for-Version_0.xlsx?rlkey=dpjl9g2a1o3jx7v2iqikvs265&st=c9aen19s&dl=0	Version 0 (Baseline)	
https://www.dropbox.com/scl/fi/6wa7kv04trs7towe7k4bl/Evaluaion-Table-for-Version_1.xlsx?rlkey=270qc07mleh197ti9esgsujgx&st=40d2rqbm&dl=0	Version 1	
https://www.dropbox.com/scl/fi/owrlv5kdpmjjrts9sll0a/Evaluaion-Table-for-Version_2.xlsx?rlkey=azmczays95p5qzm17uuxgiz70&st=63gftll4&dl=0	Version 2	

Appendix B: Source Code and Repository

The complete source code, evaluation pipeline scripts, and evaluation databases for this thesis are publicly available on GitHub at:

https://github.com/javohirbakhronovpolito/private_GPT2

B1: RAG Evaluation Program

Scripts and data files implementing the four-stage evaluation pipeline (Capture, Normalize, Label Join, Compute Metrics) described in Chapter 5.

File / Folder	Description	GitHub Link
extract_from_eval_run.py	Stage B — extracts retrieved chunks and answers from PrivateGPT eval_run.json export	View on GitHub
extract_retrieval_log.py	Stage B (alt.) — extracts retrieval log from per-query JSON files instead of unified export	View on GitHub
merge_gold_and_retrieval.py	Stage C — joins ground-truth gold.json with retrieval_log.json to produce query_items.json	View on GitHub
run_eval_on_query_items.py	Stage D — runs all metric computations and outputs eval_results.json	View on GitHub
rag_metrics.py	Pure-function metric engine: computes Groups E–H metrics from in-memory data structures	View on GitHub
gold.json	Ground-truth annotation file: expert-annotated correct answers for all 65 evaluation tasks	View on GitHub

B2: Metadata Enrichment Module (V2)

Source files implementing the chunk-level metadata enrichment layer of Version 2, including the ChunkMetadata schema, enrichment strategies, coordinator class, and runtime configuration.

File / Folder	Description	GitHub Link
chunk_metadata_enrichment.py	V2 metadata enrichment module: ChunkMetadata schema, all four strategies, coordinator class	View on GitHub
settings.yaml	Runtime configuration: enables/disables title extraction, keyword extraction, summarization, type classification	View on GitHub