

Master's thesis 2026

Towards Reliable Retrieval Systems: Design and Evaluation of an Agentic GraphRAG Pipeline

Thesis for the degree of Master of Science in Complex Adaptive
Systems

Adam Spreitz

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden

Master's thesis 2026

Towards Reliable Retrieval Systems: Design and Evaluation of an Agentic GraphRAG Pipeline

Adam Spreitz

Supervisor: Christian Ehrenström Roos

Examiner: Mats Granath

Department: Department of Physics

University: Chalmers University of Technology

Gothenburg, Sweden, 2026

Abstract

Large language models are increasingly used for knowledge-intensive question answering, but their reliability remains limited when answers must be grounded in domain-specific documents. This limitation is particularly important in high-assurance environments, where systems must be controllable, traceable, and deployable without relying on external infrastructure. This thesis investigates the design and evaluation of an agentic GraphRAG system for document-grounded question answering, using scientific literature as a controlled proxy for technical internal documentation.

The implemented system combines document ingestion, section-aware chunking, knowledge graph construction, vector indexing, graph traversal, reranking, and language-model-based answer generation. Two retrieval architectures are compared under shared conditions: a VectorRAG baseline using iterative hybrid search and a GraphRAG system using community-first hierarchical traversal over an explicit knowledge graph. The systems are evaluated across multiple retrieval configurations, generation models, and query sets using automated RAG evaluation metrics, statistical tests, and pairwise LLM-as-judge comparisons.

The results show a clear divergence between retrieval-oriented metrics and answer-level evaluation. VectorRAG achieves stronger automated retrieval scores, particularly on context precision and recall, while GraphRAG is preferred in holistic answer comparisons and manual validation. This suggests that chunk-level retrieval metrics do not always capture the usefulness of structurally retrieved evidence for downstream answer generation. The findings indicate that graph-based retrieval can improve answer quality and interpretability in agentic RAG systems, but also highlight important limitations related to dataset construction, evaluator dependence, corpus scale, graph quality, and agentic control.

Keywords: GraphRAG, retrieval-augmented generation, knowledge graphs, agentic systems, evaluation, Neo4j, LangChain, large language models

Acknowledgements

I would like to thank my supervisors at Saab and Chalmers for their guidance, support, and valuable feedback throughout this thesis. I am also deeply grateful to the team at X Innovation Lab for their warm welcome, encouragement, and generosity throughout the project. It has been a privilege to work in such a professional and inspiring environment.

Finally, this thesis is dedicated to my nephew, Benjamin. You arrived just as this work reached its final pages, a reminder that while one chapter closes, another begins. May your life be filled with curiosity, courage, and wonder.

List of Acronyms

RAG	Retrieval-Augmented Generation
GraphRAG	Graph-based Retrieval-Augmented Generation
LLM	Large Language Model
VLM	Vision-Language Model
KG	Knowledge Graph
GDS	Graph Data Science
QA	Question Answering
BM25	Best Matching 25
RRF	Reciprocal Rank Fusion
API	Application Programming Interface
DOI	Digital Object Identifier
JSON	JavaScript Object Notation
MCP	Model Context Protocol
IR	Information Retrieval

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem Description and Motivation	2
1.3	Aim and Research Questions	3
1.4	Scope Delimitations	4
2	Theory	5
2.1	Information Retrieval	5
2.2	Retrieval-Augmented Generation	6
2.3	Knowledge Graphs	7
2.4	Agentic Systems	8
2.4.1	Agents	8
2.4.2	State and Memory	8
2.4.3	Iterative Agent Loops	9
2.4.4	Context	9
2.4.5	Prompts	9
2.4.6	Tools and Tool Calling	10
2.5	Evaluation of Retrieval-Augmented Systems	10
2.5.1	Retrieval Evaluation	10
2.5.2	Generation Evaluation	11
2.5.3	LLM-as-a-Judge	11
2.5.4	Statistical Significance Testing	11
3	Methodology	13
3.1	Research Design	14
3.1.1	Threats to Validity	17
3.2	Corpus Construction	18
3.2.1	Corpus Choice	18
3.2.2	Corpus Statistics	18
3.2.3	Knowledge Graph Representation	20

3.3	Software Tools and Frameworks	20
3.4	Evaluation Methodology	21
3.4.1	Evaluation Objectives and Hypothesis	21
3.4.2	Evaluation Datasets	22
3.4.3	Evaluation Metrics	25
3.4.4	Evaluation Implementation	27
3.4.5	Final Evaluation Protocol	30
3.4.6	Threats to Validity	33
4	System Design and Implementation	35
4.1	Ingestion and Knowledge Graph Construction	35
4.1.1	Ingestion & Pre-processing	35
4.1.2	Knowledge Graph Construction	37
4.2	Retrieval Design	43
4.2.1	Shared Infrastructure	44
4.2.2	Query Decomposition	44
4.2.3	Vector RAG: Iterative Hybrid Search	45
4.2.4	Graph RAG: Community-First Hierarchical Traversal	47
4.3	Evaluation Implementation	51
4.3.1	Evaluation Framework Architecture	51
4.3.2	RAGAS implementation	52
4.3.3	Comparative Evaluation and Sensitivity Analysis	53
4.3.4	Statistical Test Implementation	53
5	Results	54
5.1	Phase 1: Ablation selection	55
5.1.1	Selection summary	55
5.1.2	GraphRAG + OSS → <code>graph_default</code>	56
5.1.3	GraphRAG + Mistral → <code>graph_thorough</code>	56
5.1.4	VectorRAG + OSS → <code>vector_minimal</code>	59
5.1.5	VectorRAG + Mistral → <code>vector_thorough</code>	59
5.1.6	Phase 1 takeaway	64
5.2	Phase 2: Finalist bout	64
5.3	Phase 3: Manual validation	68
6	Discussion	71
6.1	Main Interpretation of Results	72
6.2	GraphRAG Compared with VectorRAG	73
6.3	Evaluation Design and Threats to Validity	74

6.4	Corpus and Graph Design Implications	75
6.5	System Limitations	76
6.6	Industrial Relevance	77
7	Conclusion	79
7.1	Summary	79
7.2	Answers to the Research Questions	80
7.3	Contributions	81
7.4	Future Work	82
A	Software Configuration and Frameworks	86
A.1	Document Processing	86
A.1.1	GROBID	86
A.1.2	Docling	86
A.2	Graph Database	86
A.2.1	Neo4j	86
A.3	Retrieval and Evaluation	87
A.3.1	LangChain	87
A.3.2	LangFuse	87
A.3.3	RAGAS	87
A.4	Implementation helpers	87
A.4.1	Docker	87
B	Prompts	88
B.1	Edge Case Handling	88
B.1.1	Figure Summary Prompt	88
B.1.2	Table Reconstruction Prompt	88
B.1.3	Table Summary Prompt	89
B.1.4	Community Summary Prompt	89
B.1.5	Curated Dataset Prompt	89
B.1.6	Comparative LLM-judge Prompts	90
C	Graph Schema and Constraints	92
C.1	Knowledge Graph Schema and Index Configuration	92
C.1.1	Node Types	92
C.1.2	Relationship Types	92
C.1.3	Indexes	93
C.1.4	Schema Diagram	94
C.1.5	Visualisations	94

D Additional Results	101
E Algorithms	102
E.0.1 Community Creation	103
E.0.2 Embedding Windows	104
F Json Schemas	105
F.0.1 Ingestion Schema	105

Chapter 1

Introduction

1.1 Background

Large language models (LLMs) have demonstrated strong capabilities in natural language understanding and generation [1], enabling applications such as question answering, summarization, and decision support. In many practical systems, these capabilities are accessed through external API-based services provided by commercial model vendors. While such services offer strong performance and ease of integration, they also introduce dependencies on third-party infrastructure, pricing, availability, and data-handling policies [2].

In high-assurance environments, such dependencies can be problematic. Organizations working with sensitive or restricted information often require systems that can be deployed locally, controlled internally, and inspected throughout the processing pipeline. On-premise deployment of LLM-based systems can address some of these concerns, but large models require substantial computational resources and can be expensive to operate at scale. Smaller locally deployable models are more practical, but often perform worse on complex knowledge-intensive tasks, especially when the answer depends on domain-specific information not contained in the model parameters.

Retrieval-Augmented Generation (RAG) addresses this limitation by providing the language model with external context at inference time [3]. Instead of relying only on knowledge encoded during training, the model receives retrieved document passages that are relevant to the user query. This can improve factual grounding and reduce hallucination by anchoring generation in source material. However, conventional RAG systems often rely primarily on vector similarity search over text chunks. While effective for many single-hop retrieval tasks, flat vector retrieval can struggle when relevant evidence is distributed across multiple documents, sections, or connected concepts. It also provides limited transparency into why particular pieces of evidence were retrieved [4].

Graph-based Retrieval-Augmented Generation (GraphRAG) extends RAG by introducing an explicit graph representation of the document collection. Documents, sections, chunks, authors, references, figures, tables, and other entities can be represented as nodes, while structural and semantic relationships are represented as edges. This makes it possible to combine semantic retrieval with graph traversal, allowing the system to navigate through document hierarchy, citation links, topical communities, and other explicit relationships. In principle, such structure can support multi-hop retrieval, improve context completeness, and make retrieval behaviour more interpretable.

This thesis investigates whether an agentic GraphRAG system can improve document-grounded question answering compared with a vector-based RAG baseline under controlled conditions. The work is motivated by internal-document retrieval in high-assurance environments, but the evaluation is conducted on a corpus of scientific papers used as a reproducible proxy for technical internal documentation. The study focuses on system design, retrieval behaviour, answer quality, and evaluation methodology, with particular attention to how automated retrieval metrics, statistical tests, and pairwise answer-level judging can lead to different conclusions about system performance.

1.2 Problem Description and Motivation

API-based LLM systems can provide strong performance, but their reliance on external providers makes them difficult to use in high-assurance environments such as the defence sector. In such settings, systems are often required to satisfy strict demands on security, reliability, traceability, and operational control. Dependence on third-party infrastructure can conflict with these requirements, since model access, service availability, pricing, and data-handling policies remain outside the organization’s direct control.

A natural alternative is to deploy LLM-based systems locally. However, large language models require substantial computational resources, making local deployment expensive and difficult to scale. Smaller models are more practical for controlled environments, but their reduced capacity can increase the risk of hallucination, incomplete grounding, and unreliable answers in knowledge-intensive tasks. This creates a central challenge: how can locally deployable language-model systems provide reliable answers over domain-specific document collections without depending on external model providers?

Retrieval-Augmented Generation (RAG) partially addresses this challenge by grounding generation in retrieved source material. However, conventional RAG systems commonly rely on flat retrieval over unstructured text chunks. This can be effective when the answer is contained in a single clearly relevant passage, but it may be insufficient when the required evidence is distributed across multiple documents, sections, or related concepts. In such cases, retrieval quality depends not only on semantic similarity

to the query, but also on the system’s ability to follow relationships within the document collection.

GraphRAG provides a promising direction by representing the document collection as an explicit knowledge graph. By encoding papers, sections, chunks, citations, figures, tables, and topical communities as connected entities, retrieval can be guided through both semantic similarity and graph structure. Combined with agentic traversal, this enables the system to make explicit retrieval decisions, inspect intermediate evidence, and collect context through structured paths rather than relying only on corpus-wide vector search.

The motivation of this thesis is therefore to investigate whether an agentic GraphRAG system can improve document-grounded question answering compared with a vector-based RAG baseline under controlled conditions. The system design is motivated by deployment requirements in restricted or high-assurance environments, where local control and traceability are important. However, the empirical evaluation in this thesis is conducted on non-confidential scientific literature and uses external evaluator models only as part of the offline research workflow for dataset construction and assessment. These external models are used only to accelerate the evaluation workflow through parallel evaluation calls. They are not part of the deployed retrieval architecture under study.

1.3 Aim and Research Questions

The aim of this thesis is to design and evaluate an agentic GraphRAG system. The work is guided by the following research questions:

1. How can an agentic GraphRAG system be designed to support document-grounded question answering over technical document collections?
2. How does GraphRAG compare with VectorRAG in terms of retrieval quality, answer grounding, and overall question-answering performance?
3. What failure modes emerge in multi-loop agentic retrieval systems, and how can they be detected and mitigated?
4. How does explicit graph traversal affect the interpretability and traceability of retrieval compared with vector-based retrieval?
5. What practical challenges and limitations arise when evaluating agentic GraphRAG and VectorRAG systems?

1.4 Scope Delimitations

This thesis is limited to the design, implementation, and evaluation of a proof-of-concept retrieval-augmented question-answering system. The focus is on comparing an agentic GraphRAG architecture with a vector-based RAG baseline under controlled experimental conditions. The work does not aim to deliver a production-ready system.

The project was conducted within a fixed 20-week period, including literature review, system design, prototyping, implementation, evaluation, and thesis writing.

The system is motivated by internal-document retrieval in high-assurance environments, but the empirical evaluation is conducted on a controlled corpus of scientific papers. This corpus was selected because it provides a reproducible proxy for technical internal documentation, containing hierarchical structure, dense terminology, citations, figures, tables, and cross-document relations. The results should therefore be interpreted as evidence of system behaviour under controlled conditions, rather than as a direct validation on confidential industrial documentation.

The retrieval and generation pipelines evaluated in this thesis are designed to reflect a locally deployable architecture. Core system components, including document processing, graph storage, retrieval, and answer generation, are implemented to operate within controlled infrastructure. External API-based models are not part of the retrieval or generation architecture under study. However, external models are used in the offline research workflow to accelerate dataset construction and evaluation on non-confidential scientific material, primarily through parallel evaluation calls. This distinction separates the deployment-oriented system design from the practical tools used to carry out the thesis evaluation.

Several delimitations follow from the project scope and available resources. The corpus size is limited, the number of evaluated queries is modest, and only a small number of model and retrieval configurations are tested. The study does not evaluate long-term graph maintenance, access-control mechanisms, continuous document updates, production monitoring, or deployment in a live industrial environment. These aspects are left for future work.

Chapter 2

Theory

2.1 Information Retrieval

Information Retrieval (IR) concerns the task of identifying documents or document fragments that are relevant to a user query. Modern retrieval systems typically combine lexical and semantic retrieval methods to improve both precision and recall.

Dense retrieval and vector embeddings Dense retrieval represents documents and queries as numerical vectors produced by an embedding model. Relevance is estimated through similarity measures such as cosine similarity. This approach captures semantic relationships between words and phrases, allowing retrieval of conceptually related content even when exact keywords do not match.

Sparse retrieval Sparse retrieval methods rely on lexical matching between query terms and document terms. BM25 is one of the most widely used ranking algorithms and estimates relevance based on term frequency, inverse document frequency, and document length normalization. Sparse retrieval often performs well when queries contain specific terminology [5].

Hybrid retrieval Dense and sparse retrieval capture complementary signals. Hybrid retrieval combines semantic similarity and lexical matching to improve robustness across different query types [6].

Reciprocal Rank Fusion Reciprocal Rank Fusion (RRF) is a rank aggregation method that combines multiple ranked retrieval lists. Documents receive a score based on their position in each ranking, allowing the system to benefit from multiple retrieval signals without requiring score normalization [7].

Cross-encoder reranking Initial retrieval typically prioritizes recall over precision. Cross-encoder reranking evaluates query-document pairs jointly using a transformer model, producing more accurate relevance estimates at higher computational cost. Rerankers are therefore commonly applied to a smaller candidate set retrieved during the initial retrieval stage [8].

2.2 Retrieval-Augmented Generation

Large Language Models (LLMs) possess strong language understanding and generation capabilities, but their knowledge is limited to information acquired during training. As a result, they may produce incorrect, outdated, or fabricated information when answering questions about topics that are not sufficiently represented in their training data. This phenomenon, commonly referred to as hallucination, poses a significant challenge in applications where factual correctness and traceability are required.

Retrieval-Augmented Generation (RAG) addresses this limitation by combining information retrieval with language-model-based generation. Rather than relying solely on knowledge stored within model parameters, a RAG system retrieves relevant information from an external knowledge source and provides it to the language model as context during inference. This allows the generated response to be grounded in source material that can be inspected and verified [3].

A typical RAG pipeline consists of four stages. First, the user query is transformed into a representation suitable for retrieval. Second, relevant documents or document fragments are retrieved from a knowledge source such as a vector database, document collection, or knowledge graph. Third, the retrieved information is assembled into a context window and combined with the user query. Finally, the language model generates a response conditioned on the provided context.

By incorporating external knowledge at inference time, RAG systems can improve factual accuracy, reduce hallucination, and provide access to information that was unavailable during model training. The approach also enables knowledge updates without requiring model retraining, since changes can be made directly to the underlying knowledge source.

Despite these advantages, traditional RAG systems have several limitations. Most implementations operate on independent text chunks retrieved through semantic similarity search. While effective for many retrieval tasks, this approach often treats documents as isolated pieces of information and may struggle when relevant evidence is distributed across multiple documents or connected concepts [4]. In such cases, retrieving the most semantically similar chunks does not necessarily provide the most complete context for answering a query.

These limitations become particularly apparent in multi-hop retrieval tasks, where answering a question requires combining information from several sources. Traditional retrieval methods may successfully retrieve individual pieces of evidence while failing to capture the relationships between them. This motivates the exploration of structured retrieval approaches that explicitly model connections within the document collection.

One such approach is Graph-based Retrieval-Augmented Generation (GraphRAG), which extends the traditional RAG paradigm by incorporating knowledge graph structures into the retrieval process. By representing documents and their relationships explicitly, GraphRAG enables retrieval strategies that combine semantic similarity with graph traversal, allowing the system to explore connected information and construct richer retrieval contexts.

2.3 Knowledge Graphs

Knowledge graphs represent information as a collection of nodes and edges, where nodes correspond to entities, concepts, or documents, and edges represent relationships between them. Unlike traditional document collections, which primarily store information as isolated text, knowledge graphs explicitly capture how pieces of information are connected. This enables both structured querying and graph-based reasoning over the stored data [9].

In document-centric applications, nodes may represent documents, sections, text chunks, authors, figures, tables, or references, while edges capture structural and semantic relationships such as authorship, citation, document hierarchy, sequential order, or topical similarity. The resulting graph provides a structured representation of the document collection that can be explored beyond simple keyword or semantic matching.

A key advantage of knowledge graphs is their ability to encode both local and global structure. Local structure captures relationships between nearby entities, such as sections belonging to the same document, while global structure captures broader patterns, such as citation networks or thematic communities spanning multiple documents. This enables retrieval strategies that exploit explicit relationships in addition to semantic similarity.

Graph traversal refers to the process of navigating a graph by iteratively following edges between connected nodes. In retrieval applications, traversal can be guided by a user query, predefined heuristics, or an autonomous agent. Rather than retrieving documents solely based on similarity to the query, the system can progressively expand its search by exploring related nodes and relationships. This allows evidence to be gathered across multiple documents and retrieval steps, supporting tasks that require multi-hop reasoning or synthesis of distributed information.

In this thesis, the knowledge graph models both the hierarchical structure of scientific documents and cross-document relationships. The graph serves as the foundation for GraphRAG retrieval, enabling the system to combine semantic search with structured traversal of document hierarchy, citation links, semantic similarity relationships, and community structure.

2.4 Agentic Systems

Agentic systems extend traditional LLM-based pipelines by introducing autonomous decision-making components that can dynamically control how information is retrieved, processed, and generated. Rather than executing a fixed sequence of operations, agentic systems adapt their behaviour based on intermediate results, system state, and task-specific objectives. This allows the system to iteratively gather information, evaluate progress, and modify its strategy during execution.

In retrieval-augmented systems, agentic behaviour is commonly used to improve retrieval quality by enabling query refinement, tool usage, iterative retrieval, and self-evaluation. Instead of relying on a single retrieval step, the system can progressively collect evidence, identify information gaps, and explore alternative retrieval paths before producing a final response.

2.4.1 Agents

An agent is a modular component responsible for performing a specific task within a larger system. Depending on its purpose, an agent may retrieve information, reformulate queries, evaluate outputs, generate responses, or coordinate other components. Agents typically operate through a defined input-output interface and use a language model to make decisions within their assigned scope [10, 11].

By decomposing complex workflows into specialized agents, systems can become more modular, interpretable, and easier to debug. In this thesis, different agents are responsible for tasks such as query decomposition, graph traversal, retrieval evaluation, and answer generation.

2.4.2 State and Memory

State refers to the information accumulated during system execution. In iterative retrieval pipelines, this may include retrieved documents, intermediate outputs, evaluation scores, previous actions, and traversal history. Maintaining state across iterations allows the system to build upon previous results rather than repeatedly solving the same problem from scratch.

Memory mechanisms enable agents to avoid redundant operations, track progress, and make decisions based on earlier observations. In practice, memory may consist of both conversational state and externally retrieved evidence. Separating working memory from retrieved evidence can help control context growth while preserving access to previously discovered information.

2.4.3 Iterative Agent Loops

Many agentic systems operate through iterative execution loops rather than a single retrieval-generation cycle. After retrieving information and generating an initial response, the system evaluates the quality of the result and determines whether additional evidence is required. If deficiencies are detected, the system can refine the query, retrieve additional information, and repeat the process.

This iterative approach allows retrieval behaviour to adapt to the information discovered during execution. Rather than assuming that the first retrieval attempt is sufficient, the system can progressively improve its evidence base through multiple retrieval and evaluation cycles. Such loops are particularly useful for complex queries requiring synthesis across multiple documents or reasoning over distributed information.

2.4.4 Context

Context refers to the information provided to the language model during generation. In retrieval-augmented systems, context typically consists of evidence retrieved from an external knowledge source. Since language models condition their outputs directly on the provided context, retrieval quality has a substantial impact on answer quality.

Effective context construction involves selecting relevant information, minimizing noise, and organizing evidence in a way that supports downstream reasoning. Even when relevant information exists within the document collection, poor context selection can lead to incomplete or incorrect responses.

2.4.5 Prompts

Prompts define the instructions and inputs provided to a language model. They influence how the model interprets tasks, reasons about information, selects actions, and generates outputs. In agentic systems, prompts are often used to specify behavioural constraints, retrieval strategies, output formats, and tool-usage guidelines.

Prompt design therefore plays an important role in system performance. Well-designed prompts can improve consistency, reduce undesired behaviour, and encourage structured reasoning, while poorly designed prompts may lead to ambiguous or unreliable outputs.

2.4.6 Tools and Tool Calling

Tools are external functions or interfaces that agents can invoke to perform operations beyond text generation. Examples include database queries, search functions, graph traversal operations, and document retrieval mechanisms. Tool usage allows language models to interact with structured data and access information that is not contained within their parameters [12].

Modern agentic systems commonly employ tool-calling mechanisms, where the language model decides when a tool should be invoked and which tool is most appropriate for the current task. Rather than directly generating a response, the model may first retrieve evidence, query a database, or explore a knowledge graph before continuing its reasoning process.

In this thesis, tool calling forms the foundation of the retrieval architecture. Agents use specialized tools to navigate the knowledge graph, retrieve document content, explore semantic communities, and collect evidence required for downstream answer generation. This enables retrieval decisions to be made explicitly and inspected throughout the execution process.

2.5 Evaluation of Retrieval-Augmented Systems

Evaluating Retrieval-Augmented Generation (RAG) systems is challenging because system quality depends on both retrieval performance and generation quality. A system may retrieve highly relevant evidence but fail to use it effectively during generation, while another system may generate useful answers despite retrieving context that differs from a predefined reference. Consequently, RAG evaluation is commonly divided into retrieval-oriented and generation-oriented assessments [13].

2.5.1 Retrieval Evaluation

Retrieval evaluation measures how effectively a system identifies and retrieves information relevant to a user query. Traditional information retrieval metrics often focus on relevance, ranking quality, and coverage of known ground-truth evidence.

In RAG systems, retrieval evaluation typically assesses whether the retrieved context contains the information required to answer the query. Common metrics include context precision, which measures the proportion of retrieved information that is relevant, and context recall, which measures how much of the relevant information was successfully retrieved. Entity-level metrics may also be used to determine whether important concepts, entities, or relationships are present in the retrieved evidence.

Retrieval metrics are useful for diagnosing search behaviour and comparing retrieval strategies. However, they do not directly measure answer quality and may favour systems that retrieve evidence similar to predefined reference contexts.

2.5.2 Generation Evaluation

Generation evaluation assesses the quality of answers produced by the language model. Unlike retrieval metrics, generation metrics focus on whether the generated response is correct, relevant, and supported by the retrieved evidence.

A common objective is to measure faithfulness, which evaluates whether statements in the generated answer can be supported by the retrieved context. Additional metrics may assess answer relevance, completeness, correctness, or groundedness with respect to available evidence.

Generation-oriented evaluation is particularly important because the ultimate objective of most RAG systems is not retrieval itself, but the production of useful and trustworthy answers.

2.5.3 LLM-as-a-Judge

Recent work has increasingly employed large language models as evaluators of retrieval and generation quality. Rather than relying solely on reference-based metrics, an evaluator model is prompted to assess the quality of retrieved evidence or compare generated answers according to predefined criteria.

Pairwise comparison is a common approach in which the evaluator is presented with two candidate answers and asked to determine which is superior. This method can provide a more holistic assessment of answer quality by considering factors such as completeness, clarity, reasoning quality, and overall usefulness [14].

Despite their effectiveness, LLM-based judges are not without limitations. Evaluation outcomes may be influenced by prompt design, model bias, answer ordering, and evaluator model choice [15]. As a result, LLM-as-a-judge methods are often used alongside automated metrics rather than as a complete replacement for them.

2.5.4 Statistical Significance Testing

Observed differences between systems may arise due to random variation rather than genuine performance differences. Statistical significance testing is therefore commonly used to determine whether observed improvements are likely to be meaningful [16].

For retrieval and generation evaluation, non-parametric tests are often preferred because metric distributions may not satisfy the assumptions required by parametric

methods. Statistical testing can help distinguish consistent performance improvements from differences that occur due to chance alone.

When many pairwise comparisons are performed simultaneously, multiple-comparison corrections may be applied to reduce the probability of incorrectly identifying insignificant differences as significant. Such methods increase the reliability of conclusions drawn from comparative evaluation [17]

Chapter 3

Methodology

This chapter presents the methodology used to design, implement, and evaluate the proposed retrieval systems. It is structured around four main areas: research design, corpus construction, software tools and frameworks, and evaluation methodology.

The research design section introduces the experimental setup, including the systems under comparison, the controlled and independent variables, and the dependent outcomes used to assess system performance. The corpus construction section describes the document collection used in the study, motivates the choice of scientific literature as a controlled proxy for technical internal documentation, and summarizes the structural properties of the processed corpus.

The chapter then describes the tools, frameworks, and evaluation procedures used in the study. These sections explain how the experimental environment was constructed, how system outputs were evaluated, and how comparisons between GraphRAG and VectorRAG were made in a controlled and reproducible manner.

3.1 Research Design

The objective of this study is twofold. First, it investigates how a graph-based retrieval architecture compares with a vector-based retrieval architecture for document-grounded question answering. Second, it examines how different retrieval configurations influence the behaviour and performance of agentic retrieval systems.

Evaluating Retrieval-Augmented Generation (RAG) systems requires more than assessing the final generated answer. System performance depends on multiple interacting components, including the ability of the retrieval system to identify relevant evidence, the ability of the language model to use that evidence faithfully, and the quality of the final response itself [13]. Consequently, the evaluation considers both retrieval-oriented and generation-oriented outcomes.

To isolate the effect of retrieval strategy, the retrieval component is treated as the principal independent variable. The ingestion pipeline, document corpus, database infrastructure, generation models, and evaluation framework are kept constant wherever possible. This experimental design allows observed differences in performance to be attributed primarily to differences in retrieval behaviour rather than other system components.

The overall experimental workflow is illustrated in Figure 3.1. Documents are first processed through a shared ingestion pipeline and stored within a common database infrastructure. Separate VectorRAG and GraphRAG retrieval systems then operate over the same underlying corpus. Retrieved evidence is subsequently passed to the same generation models and evaluated using a common evaluation framework.

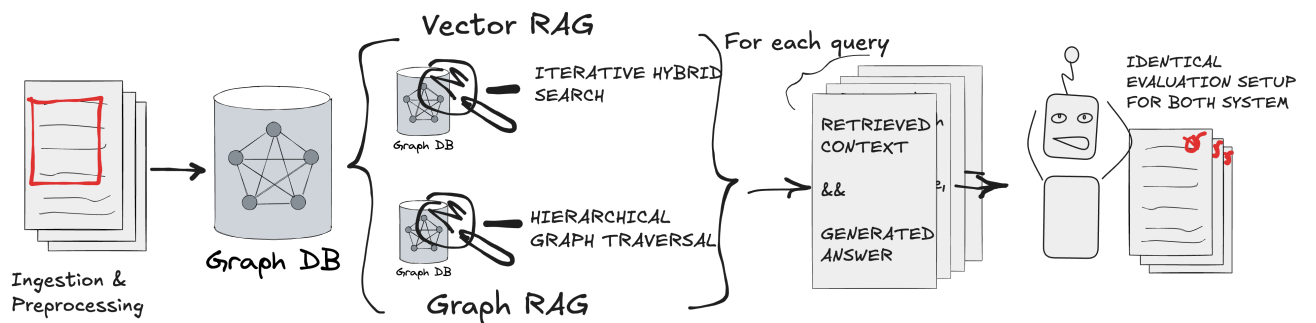


Figure 3.1: Overview of the experimental workflow. The ingestion pipeline, database infrastructure, generation models, and evaluation framework are held constant, while the retrieval strategy is varied to study its effect on retrieval behaviour and answer quality.

The experimental design is summarized in Table 3.1. Controlled variables ensure that both systems are evaluated under comparable conditions. The retrieval architecture constitutes the primary independent variable, while retrieval quality, answer quality, comparative judging outcomes, and statistical relationships are treated as dependent

variables.

The evaluation follows a comparative design in which GraphRAG and VectorRAG are assessed under the same experimental conditions. Retrieval quality is measured using automated retrieval metrics, while answer quality is evaluated through generation metrics, pairwise LLM-as-a-judge comparisons, and manual validation. Statistical significance testing is used to determine whether observed performance differences are likely to reflect genuine behavioural differences rather than random variation across queries.

Table 3.1: Overview of controlled, independent, and dependent variables in the experimental setup.

Variable type		Component	Description
Controlled variable		Ingestion and pre-processing	Both systems operate on the same corpus of scientific papers. Text, tables, figures, and equations are extracted, chunked, and embedded using the same ingestion pipeline and embedding strategy.
Controlled variable		Database infrastructure	Both systems use the same database instance. The vector-based system uses vector indexing and cosine similarity search, while the graph-based system additionally uses the graph schema and traversal functionality.
Controlled variable		Evaluation procedure	Both systems produce outputs in the same format, including retrieved chunks, retrieval metadata, and final generated answers. Outputs are evaluated using the same framework and metric configuration.
Independent variable	vari-	Vector-based RAG	An agentic retrieval pipeline using hybrid retrieval, query rewriting, reranking, and iterative retrieval loops. Retrieval is based on vector similarity and lexical search without exploiting graph structure.
Independent variable	vari-	Graph-based RAG	An agentic retrieval pipeline using the explicit graph structure for layered retrieval and graph traversal. Retrieval decisions are guided by tool-calling agents operating over graph relationships.
Dependent variable		Retrieval and generation performance	Retrieval and generation quality are evaluated using the metrics described in Section 3.4.3. The same metric configuration is applied to both systems.
Dependent variable		Direct comparison	A comparative LLM-as-a-judge evaluation compares generated answers and retrieved contexts from both systems using the same prompt and judging criteria.
Dependent variable		Statistical relationships	Statistical significance testing is used to assess whether observed differences between Vector RAG and GraphRAG configurations are meaningful across queries.

3.1.1 Threats to Validity

Several factors may influence the validity of the conclusions drawn from this study.

Construct Validity Evaluating RAG systems is inherently challenging because no single metric fully captures system quality. Retrieval-oriented metrics measure the relevance and coverage of retrieved evidence, while generation-oriented metrics assess the quality of the final answer. Additionally, LLM-as-a-judge evaluations rely on the judgement of another language model and may be influenced by evaluator bias, prompt design, or answer ordering. To mitigate these risks, the study combines multiple evaluation methods, including automated metrics, pairwise judging, statistical testing, and manual validation.

Internal Validity The primary objective is to isolate the effect of retrieval strategy on system performance. To reduce confounding factors, both GraphRAG and VectorRAG operate on the same document corpus, use the same ingestion pipeline, share the same generation models, and are evaluated using the same procedures. However, agentic retrieval systems contain non-deterministic components, including language-model reasoning and tool-calling behaviour, which may introduce variation between runs.

External Validity The evaluation is conducted on a corpus of scientific papers rather than confidential industrial documentation. Scientific literature was selected because it provides a reproducible corpus containing hierarchical structure, citations, figures, tables, and cross-document relationships. Nevertheless, the behaviour of the proposed systems may differ when deployed on real-world internal document collections with different characteristics, update patterns, and domain-specific terminology.

Conclusion Validity The study is constrained by project duration, computational resources, and evaluation cost. As a result, the number of evaluated queries, retrieval configurations, and generation models is limited. Statistical significance testing is therefore used to assess whether observed differences are likely to represent genuine performance differences rather than random variation. However, the findings should be interpreted as evidence within the scope of the evaluated corpus and experimental setup rather than universal conclusions about GraphRAG systems.

3.2 Corpus Construction

This section presents the rationale behind the selection of the document corpus and describes its characteristics. The corpus serves both as the knowledge source for the retrieval systems and as the foundation for the evaluation datasets used throughout the study.

3.2.1 Corpus Choice

Scientific literature was selected as the primary corpus because it provides a challenging and naturally structured information environment that is well suited for evaluating Retrieval-Augmented Generation (RAG) systems. Academic papers contain explicit hierarchical organization through sections and subsections, as well as rich relational information through citations, references, figures, tables, and equations. These structural properties create meaningful connections between pieces of information and facilitate retrieval tasks that require evidence to be gathered across multiple sources.

The corpus was also chosen because it shares several characteristics with technical internal documentation, which motivates the broader application domain of this work. Both scientific papers and technical documentation contain domain-specific terminology, hierarchical organization, references to related information, and dense technical content. Although scientific literature cannot fully replicate the characteristics of confidential industrial document collections, it provides a reproducible and publicly accessible proxy that enables controlled experimentation.

Prior work suggests that graph-structured retrieval can be particularly effective for multi-hop question answering because explicit relationships may guide retrieval across connected pieces of evidence [18]. At the same time, the corpus remains suitable for conventional vector-based retrieval, as the majority of information is represented as unstructured text. This combination of rich structure and dense textual content makes the corpus well suited for comparing vector-based and graph-based retrieval strategies under controlled conditions.

Scientific literature was therefore selected as a balance between structural richness, technical complexity, and experimental reproducibility. The resulting corpus provides an environment in which both retrieval paradigms can be evaluated on realistic information-seeking tasks while maintaining full control over the underlying data.

3.2.2 Corpus Statistics

The domains represented in the corpus were selected primarily based on the research areas accessible within the surrounding academic and industrial environment during

the project period. As a result, the corpus consists predominantly of literature related to radar systems, agentic LLM systems, Retrieval-Augmented Generation, Bayesian optimization, knowledge graphs, and world models.

There is substantial thematic overlap between several of the largest domains, particularly RAG systems, knowledge graphs, agentic systems, and world models. This overlap is expected, as these topics are closely related to the central focus of the thesis. At the same time, the inclusion of additional technical domains introduces variation in terminology, writing style, and document structure, providing a more diverse retrieval environment.

The final corpus consists of 103 scientific papers distributed across the selected technical domains, as shown in Figure 3.2.

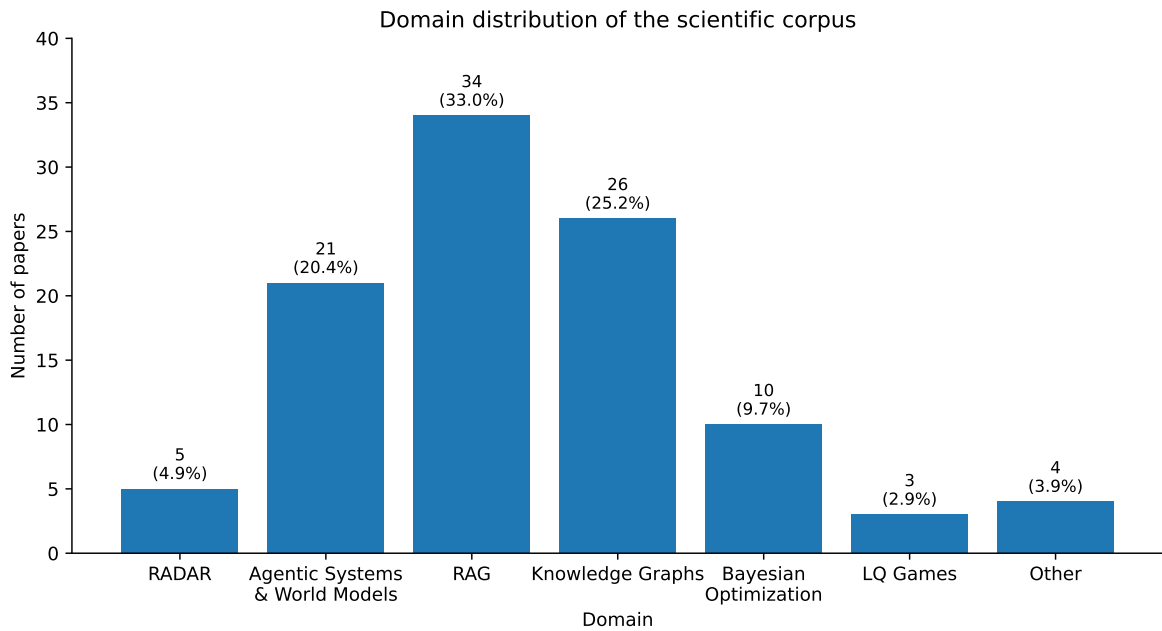


Figure 3.2: Domain distribution of the scientific corpus. The corpus consists of 103 papers distributed across seven primary domain categories.

Table 3.2: Summary statistics of the processed corpus.

Statistic	Value
Number of papers	103
Total chunks	4186
Total sections	3295
Total figures	664
Total tables	444
Total equations	1934
Mean chunks per paper	40.6

The corpus contains a substantial amount of structured content beyond plain text, including figures, tables, and equations. This is particularly relevant for the proposed GraphRAG architecture, as these artifacts are explicitly extracted and represented within the knowledge graph rather than being discarded during preprocessing. The resulting graph therefore captures not only textual content, but also structural relationships present within and between documents.

3.2.3 Knowledge Graph Representation

The processed corpus is transformed into a knowledge graph that serves as the foundation for retrieval. Each paper is represented through a hierarchy of connected nodes, including papers, sections, chunks, figures, tables, and equations. Relationships between nodes capture both document structure and semantic connections.

Hierarchical relationships represent the organization of each document, linking papers to sections and sections to their corresponding content elements. Additional relationships capture cross-document information such as citations, semantic similarity, and community structure. Together, these relationships provide multiple pathways through which information can be retrieved and explored.

This representation enables the retrieval system to operate not only on textual similarity, but also on explicit structural relationships contained within the corpus. As a result, the graph can support retrieval strategies that traverse document hierarchy, follow citation links, explore semantically related content, and aggregate evidence across multiple sources. The construction of this graph representation is described in detail in the following sections.

3.3 Software Tools and Frameworks

The implementation relies on several tools selected for their suitability for scientific document processing, graph-based storage, retrieval orchestration, observability, evaluation, and reproducible local deployment. This section provides a concise overview of the primary tools used in the system. A summary is provided in Table 3.3, while more detailed descriptions are included in Appendix A.

The core retrieval and generation pipelines were implemented to run within controlled infrastructure, using locally deployed components where required. This constraint influenced model selection, system design, and evaluation feasibility. External API-based models were not part of the retrieval or generation architecture under study. However, external models were used in the research workflow to accelerate dataset construction and evaluation on non-confidential scientific material, primarily through parallel evaluation calls.

Table 3.3: Primary software tools and their role in the system.

Tool	Role	Used for
Docling	Document parsing	Text, sections, tables, figures
GROBID	Metadata extraction	Authors, abstracts, references, DOI
Neo4j	Graph database	Knowledge graph, traversal, vector indexes
LangChain	Agent orchestration	Tool calling, prompts, state propagation
Langfuse	Observability	Traces, tokens, tool calls, latency
RAGAS	Evaluation	Retrieval and generation metrics
Docker	Deployment	Reproducible local infrastructure
Python	Implementation	Pipeline and system logic
AI assistants	Development support	Ideation, debugging, and code suggestions

3.4 Evaluation Methodology

The objective of the evaluation is to compare VectorRAG and GraphRAG under different retrieval configurations, in order to investigate their respective strengths and weaknesses. The systems are evaluated on both single-hop and multi-hop questions, with particular focus on how retrieval configuration affects downstream performance. The evaluation framework assesses both retrieval quality and generation quality.

3.4.1 Evaluation Objectives and Hypothesis

The evaluation is designed to answer the following questions:

- How does vector-based RAG compare to graph-based RAG across the constructed evaluation datasets?
- How do the systems perform on single-hop compared to multi-hop questions?
- How do the systems differ in retrieval quality and generated answer quality?
- How do token limits, prompt size, retrieval loop budgets, and graph traversal depth affect system performance?
- How sensitive are the systems to architectural choices such as context assembly, graph guidance, and tool-calling constraints?

According to literature graph RAG performs better on multi-hop questions and reduces hallucinations for larger corpora [4]. Given that it provides structure and direct multi-hop capabilities for the agent. As such, the hypothesis were the following;

H1: GraphRAG will outperform VectorRAG on multi-hop queries due to explicit relational retrieval.

H2: VectorRAG will perform competitively on single-hop questions.

H3: System performance is sensitive to retrieval configuration and retrieval depth.

H4: Chunk-level evaluation datasets may favour retrieval approaches operating directly on chunks.

3.4.2 Evaluation Datasets

Three datasets were constructed to evaluate the systems fairly. As shown in Table 3.4, the datasets represent increasing levels of human involvement, ranging from fully automated generation to human-driven multi-hop evaluation.

Table 3.4: Overview of the evaluation datasets.

Dataset	Construction	Human involvement
RAGAS-generated	Automated	Low
Bootstrapped	Semi-automatic	Medium
Manual multi-hop	Manually curated	High

Dataset 1: RAGAS-generated Dataset The initial evaluation dataset was generated using the RAGAS framework [19]. RAGAS constructs evaluation records through a multi-stage LLM-driven generation process. Where content is first selected from a provided corpus and subsequently transformed into questions, reference answers, and reference contexts.

Chunks are either treated independently (single-hop synthesis) or combined into groups of related contexts (multi-hop synthesis). An LLM then generates a query conditioned on these contexts together with a reference answer and the supporting evidence required to answer the query. Each generated evaluation record therefore contains:

- Query
- Reference answer
- Reference context
- Query metadata (hop type, query category)

Single-hop synthesis generates questions answerable from a single context unit, while multi-hop synthesis combines information across multiple contexts to generate questions requiring synthesis and relational reasoning.

The generated dataset additionally included personas and four query categories:

- Factual retrieval
- Comparison
- Summarization
- Logical reasoning

The initial implementation operated on chunk-level document units to preserve retrieval granularity and enable precise ground-truth attribution.

Dataset 2: Bootstrapped Dataset To reduce the dependency on chunk-level generation and obtain a more retrieval-neutral evaluation set, a second dataset was constructed using a bootstrapping approach.

For each generation iteration, three to four papers within the same domain were selected and provided as input to a stronger external language model (Claude Opus 4). The model was instructed to generate evaluation questions spanning multiple query styles and reasoning patterns while maintaining the same structural categories established during the RAGAS phase.

The generated records contained:

- Query
- Reference answer
- Reference section or section fragment

Unlike the RAGAS dataset, the bootstrapped dataset generated questions from larger semantic units, typically entire papers, rather than isolated chunks. This reduced dependence on predefined chunk boundaries and enabled more coherent multi-hop reasoning patterns. The main trade-off was reduced retrieval granularity: exact ground-truth chunks were difficult to identify, and evidence could instead only be associated at the section or partial-section level.

A further trade-off is increased context complexity. Since generation is conditioned on several full papers rather than isolated chunks, the model is exposed to substantially larger amounts of information. While this enables richer multi-hop reasoning, it may also introduce context dilution effects, where relevant evidence becomes diluted with distracting information. This could increase the complexity of downstream generation.

This dataset represents a compromise between fully automated generation and manual construction, offering greater control while remaining feasible within the time constraints.

Dataset 3: Curated Dataset Manual evaluation remains the gold standard for evaluation dataset construction, particularly in specialized domains where expert judgement is required. However, manual construction is expensive, time-consuming and requires a domain expert, especially when multi-hop questions require careful validation across multiple sources.

Due to project constraints, we could not afford a domain expert. Instead we used the latest and most powerful LLM models as assistants in creating the dataset. And validated each query reference manually.

With the instructional prompt seen in Appendix B.1.5, three to five papers were given to the LLM along with a thematic or comparative question asked by the author. The author had only read the abstract of the papers before asking the question and thus the structural bias (favoring graph rag) was eliminated. The model then responded with an answer, and from where it got the answer. The provenance of the answer and reference was then validated by the author. If the answer was not found explicitly in the text or could not be understood by the author from the given reference. That question was removed from the dataset.

Each manually curated record was reviewed according to the following criteria:

- **Answerability:** the question must be answerable from the selected source sections.
- **Multi-hop requirement:** the answer must require synthesis across at least two papers’ sections rather than retrieval of a single isolated fact.
- **Ground-truth traceability:** the supporting sections must be explicitly identifiable so that retrieval performance can be evaluated against known evidence.

Final Dataset Composition The final evaluation setup consists of three complementary datasets:

Dataset	Size	Construction	Primary Purpose
RAGAS-generated	~100	Automated	Large-scale benchmarking
Bootstrapped	~80	Semi-automatic	Comparative evaluation
Manual	16	Human curated	Qualitative validation

Table 3.5: Final evaluation dataset composition.

The three datasets represent different trade-offs between automation, control, and human involvement.

The three datasets provide complementary evaluation perspectives. The RAGAS-generated dataset enables scalable evaluation over a larger number of queries, but

its chunk-level construction risks favouring retrieval methods that operate directly on chunks. The bootstrapped dataset reduces this dependency by generating questions from larger semantic units, thereby decreasing coupling to predefined chunk boundaries while remaining computationally feasible. The manually curated dataset provides the strongest validation signal, since each query and evidence chain is manually checked, but is limited in size due to the higher construction cost.

Together, these datasets reduce the risk that the evaluation conclusions are driven by artifacts of a single dataset construction method.

Together, these datasets provide complementary evaluation perspectives and reduce dependence on any single construction methodology.

3.4.3 Evaluation Metrics

Evaluating retrieval-augmented generation systems requires measuring both the quality of the retrieved context and the quality of the generated answer. The RAGAS evaluation framework [13], provides a suite of metrics designed specifically for RAG evaluation. Metrics were selected to cover both reference-based and reference-free evaluation, since the two categories measure fundamentally different properties and, as our results show, can lead to different conclusions about system performance.

Retrieval Metrics The following retrieval metrics aims to assess the capability of the system to retrieve correct, and sufficient information to answer user queries.

- *Context Recall* measures the fraction of claims in the reference answer that can be attributed to the retrieved context. An LLM judge decomposes the reference answer into individual statements and checks whether each is supported by the retrieved chunks. This is a reference-based metric; it requires a ground truth answer and measures how well the retrieved context covers it.
- *Context Precision* measures whether the relevant chunks are ranked higher than irrelevant ones in the retrieved set. It computes average precision over binary relevance judgments made by an LLM judge, rewarding systems that place useful content early in the context. This metric is sensitive to retrieval ordering, making it more informative for systems that perform explicit reranking.
- *Context Relevance* is a reference-free retrieval metric in our evaluation. It measures the proportion of retrieved content that is relevant to the query, without comparing against a ground truth answer. An LLM judge extracts the sentences from the retrieved context that are relevant to answering the query, and the score is the ratio of relevant sentences to total sentences. Because it does not depend on a

reference, this metric provides a fair comparison between structurally different retrieval systems.

- *Context Entity Recall* measures the fraction of entities present in the ground truth context that also appear in the retrieved context. Unlike the other metrics, this is computed through entity extraction rather than LLM judgment, providing a complementary lexical signal. It is particularly useful for fact-based queries where specific named entities must be present in the retrieval.
- *Context precision without reference*: Measures how relevant the retrieved context is to the generated answer without relying on predefined ground-truth contexts. Instead of comparing retrieval results against a reference set, a LLM-judge evaluates whether the retrieved information contributes meaningfully to the produced answer. High scores indicate that the retrieval contains mostly useful and answer-supporting information, while lower scores suggest the presence of irrelevant or noisy context.

Generation Metrics Generation metrics assess whether the system produced a correct, grounded, and relevant answer from the retrieved context

- *Faithfulness* measures whether the generated answer is grounded in the retrieved context. The LLM judge decomposes the answer into atomic claims and verifies each against the retrieved chunks. A high faithfulness score indicates that the system does not hallucinate, each claim in the answer is supported by the retrieved evidence.
- *Answer Correctness* measures the factual overlap between the generated answer and the reference answer, combining semantic similarity with factual comparison. This is a reference-based metric that captures whether the system arrived at an answer close to the reference answer, regardless of how the answer is phrased.
- *Response Groundedness* assesses whether the claims in the generated response can be traced back to the retrieved context. While similar to faithfulness, groundedness evaluates the connection between response and context at a coarser level, providing a complementary signal about the generator’s tendency to stay within the evidence.
- *Answer Relevancy* measures whether the generated answer actually addresses the query. An answer may be faithful to the context and factually correct but fail to address the specific question asked. This metric captures that distinction by evaluating the semantic alignment between the query and the generated response.

3.4.4 Evaluation Implementation

All evaluations were executed using deterministic decoding with temperature set to 0 in order to reduce stochastic variation and improve reproducibility. However, complete determinism cannot be guaranteed, as minor variability may still arise from failed/skipped tool calls affecting retrieval ordering.

Due to the computational and time cost of the evaluation pipeline, the full evaluation was not repeated systematically across many independent runs. Instead, a limited number of preliminary runs were conducted for validation and debugging, followed by one final evaluation run used for reporting the results. This limits the extent to which statistical variation can be estimated. However, the use of multiple complementary datasets, covering different construction procedures and query types, was intended to partially mitigate dependence on a single evaluation setting.

RAGAS-based Evaluation RAGAS was used as the primary evaluation framework for computing retrieval and generation metrics [19]. The framework was applied post-hoc to stored system outputs, where each evaluated entry consisted of the original query, retrieved contexts, generated answer, and reference answer where available. This allowed both retrieval quality and answer quality to be evaluated without re-running the retrieval systems. Retrieval and Generation performance was aggregated using a weighted score over the selected metrics. The weights can be seen in Tables 3.6 and 3.7.

Table 3.6: Weights used for aggregating retrieval metrics.

Metric	Weight
Context recall	0.15
Context relevance	0.35
Context precision	0.15
Context entity recall	0.15
Context precision without reference	0.20

Table 3.7: Weights used for aggregating generation metrics.

Metric	Weight
Faithfulness	0.30
Answer correctness	0.15
Answer relevancy	0.25
Response groundedness	0.30

Higher weight was assigned to reference-free metrics, particularly context relevance and context precision without reference. Reference-based metrics were still included to preserve alignment with known supporting evidence, but were given lower individual

weights to reduce overdependence on predefined ground-truth contexts. The generation metric weights follow the same rationale of balancing reference-based and reference-free evaluation. Answer correctness provides a reference-based estimate of factual alignment with the expected answer, while faithfulness, answer relevancy, and groundedness assess whether the generated response is supported by the retrieved context and addresses the query.

Generation and retrieval performance were aggregated into two separate composite scores: one for retrieval quality and one for generation quality. The retrieval composite score was computed according to Equation 3.1, while the generation composite score was computed according to Equation 3.2.

$$\text{Composite}_{\text{retrieval}} = \frac{\sum_{i=1}^n r_i \cdot w_i^r}{\sum_{i=1}^n w_i^r} \quad (3.1)$$

$$\text{Composite}_{\text{generation}} = \frac{\sum_{i=1}^n g_i \cdot w_i^g}{\sum_{i=1}^n w_i^g} \quad (3.2)$$

Comparative Evaluation In addition to the RAGAS-based evaluation, a comparative LLM-as-a-judge evaluation was implemented. In this setup, an evaluator LLM compares the output of two different RAG systems or ablations relative to the original query. The model receives the generated answer and retrieved context from both systems as input, and outputs a winner and a loser, and a short justification. The comparison prompt is provided in Appendix B.1.6.

The motivation for this evaluation is that standard retrieval and generation metrics assign isolated scores to each system, but may not fully capture relative strengths and weaknesses between retrieval approaches. The comparative evaluation therefore provides an additional pairwise signal that complements the metric-based evaluation.

Sensitivity Analysis With Ablations Ablation studies were conducted to evaluate how architectural choices and control parameters influence retrieval behaviour and overall system performance. Since retrieval quality constitutes the primary independent variable in this study, the ablation experiments focus on retrieval configuration rather than generation parameters. The generation model and prompt were held constant across all configurations to isolate the effect of retrieval changes.

Three configuration presets were defined for each retrieval system, designed to test sensitivity along a minimal-to-thorough axis:

- **Minimal.** Optimised for speed and low computational cost. Reduces retrieval depth (fewer hops or loops), lowers token budgets for agent reasoning, and relaxes

stopping criteria. For the graph path, the finish guard is disabled, allowing the agent to stop after retrieving from a single paper. For the vector path, the loop count and top- k retrieval candidates are reduced. This configuration tests the lower bound of each system’s performance: how is quality affected when retrieval resources are constrained.

- **Default.** The standard configuration used in the main evaluation, representing a balanced trade-off between retrieval thoroughness and computational cost. All behavioural controls (community search guards, finish guards) are active at their standard settings.
- **Thorough.** Maximises retrieval depth and coverage. For the graph path, the minimum paper threshold is raised to three, the coverage check tool is enabled, and the context token budget is increased. For the vector path, the loop count, retrieval candidates, and confidence threshold are all increased. This configuration tests the upper bound: whether additional retrieval effort translates to improved answer quality, or whether diminishing returns or context dilution degrade performance.

The specific parameters varied for each system are summarised in Table 3.8. Parameters not listed were held constant across all configurations to ensure that observed differences are attributable to the varied parameters alone.

Table 3.8: Ablation configuration parameters for both retrieval systems. Parameters not listed are held constant across configurations.

System	Parameter	Minimal	Default	Thorough
GraphRAG	Max hops	10	15	20
	Agent token budget	512	1024	2048
	Min papers (finish)	0	2	3
	Context token budget	2000	5000	8000
VectorRAG	Max loops	2	5	8
	Top- k vector	10	15	20
	Top- k BM25	10	15	20
	Top- k rerank	3	5	10
	Confidence threshold	0.70	0.85	0.90
	Stagnant threshold	1	3	4
	Judge draft tokens	128	256	512
	Judge faith tokens	256	512	1024
	Judge score tokens	512	1024	2048
	Reason tokens	512	1024	2048
	Query rewrite tokens	512	1024	1024

These experiments provide insight into the sensitivity of each system to retrieval depth, context volume, and control logic. By comparing across configurations within each system, we can identify diminishing returns, context dilution effects, and the

contribution of behavioural controls such as finish guards and coverage checks. By comparing the best configuration of each system against the other, we obtain a fairer cross-system comparison than a single fixed configuration would allow.

Statistical Analysis To determine whether observed performance differences between VectorRAG and GraphRAG are statistically meaningful rather than artifacts of sampling variation, three complementary statistical methods were applied.

- *Wilcoxon Signed-Rank Test*: For continuous evaluation metrics, such as context relevance, faithfulness, and answer relevancy, the Wilcoxon signed-rank test was used to compare paired observations between systems. Since both systems were evaluated on the same queries, the observations are naturally paired, making the Wilcoxon test an appropriate non-parametric alternative to the paired t-test.

The Wilcoxon test was preferred because several evaluation metrics are bounded to the interval $[0, 1]$ and cannot be assumed to follow a normal distribution. Statistical significance was assessed using a threshold of $p < 0.05$.

- *Binomial Test*: For the LLM-as-judge pairwise evaluation, each query produced a discrete outcome: GraphRAG win, VectorRAG win, or tie. Ties were excluded and a binomial test was applied to the remaining win counts. The null hypothesis assumes no systematic advantage between systems, corresponding to a win probability of $p = 0.5$. This evaluates whether the observed win distribution deviates significantly from chance.
- *Effect Size*: Statistical significance alone was considered insufficient due to the limited evaluation size of 83 queries, as small but practically irrelevant differences may still reach significance. Therefore, effect sizes were reported alongside p-values using rank-biserial correlation for Wilcoxon comparisons. This provides an estimate of the magnitude of observed differences independent of statistical significance.

3.4.5 Final Evaluation Protocol

The final evaluation uses the bootstrapped dataset as the primary benchmark and the manually curated dataset as a validation set. The RAGAS-generated dataset is excluded from the final system comparison due to its chunk-level construction process, which may introduce bias when comparing retrieval architectures that operate differently over document structure. The bootstrapped dataset reduces this dependency by generating questions from larger semantic units, while the manually curated dataset provides a smaller set of verified multi-hop questions for qualitative validation.

The evaluation combines automated metrics, statistical tests, and pairwise LLM-as-a-judge comparisons as complementary evidence lenses. Automated metrics are primarily interpreted as diagnostic indicators of retrieval behaviour, evidence coverage, ranking quality, and reference alignment. Since the ultimate objective of the systems is document-grounded question answering, final conclusions are based primarily on answer quality, groundedness, and completeness rather than retrieval overlap alone. This distinction is important because structurally different retrieval systems may retrieve different evidence while still producing equally valid answers.

Query Sets and Systems

Two query sets are used in the final evaluation:

- **Bootstrapped dataset:** 83 queries (43 single-hop and 40 multi-hop), used for the primary comparative evaluation.
- **Manual dataset:** 16 manually curated multi-hop queries, used for final validation.

Both GraphRAG and VectorRAG operate on the same document corpus, use the same ingestion pipeline, and are evaluated using the same procedures. The systems differ only in retrieval architecture and retrieval configuration.

Three-Phase Evaluation Protocol

A full factorial comparison of all retrieval configurations using expensive pairwise judging would be computationally prohibitive. The evaluation therefore follows a three-phase protocol:

1. **Phase 1 – Ablation Selection.** Within each retrieval architecture and model combination, the minimal, default, and thorough configurations are compared using automated metrics, statistical testing, and pairwise LLM-as-a-judge evaluation. The strongest configuration from each family is selected as a finalist.
2. **Phase 2 – Cross-System Comparison.** The selected finalists are evaluated against one another on the full bootstrapped dataset. This phase establishes the primary comparison between GraphRAG and VectorRAG under their strongest observed configurations.
3. **Phase 3 – Manual Validation.** The finalists are re-evaluated on the manually curated dataset using a stronger evaluator model. This phase provides an additional validation signal that is independent of the bootstrapped reference answers.

The evaluation workflow is illustrated in Figure 3.3.

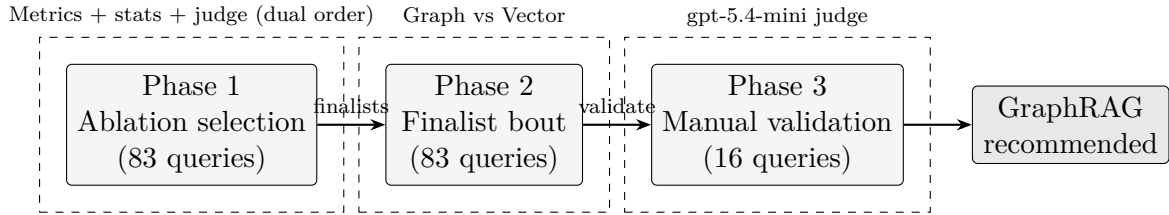


Figure 3.3: Three-phase evaluation funnel: ablation selection, finalist comparison, and manual validation.

Evidence Lenses

The evaluation triangulates three complementary sources of evidence:

Automated Metrics Retrieval and generation metrics are computed using the RAGAS framework. These metrics provide scalable and reproducible measurements of retrieval quality, answer quality, and grounding behaviour.

Statistical Testing Paired Wilcoxon signed-rank tests are applied to metric distributions to determine whether observed differences between systems are statistically meaningful.

LLM-as-a-Judge Pairwise comparisons are used to assess the overall quality of retrieved evidence and generated answers. All reported win rates are averaged across both presentation orders to reduce position bias. Phases 1 and 2 use `gpt-4o-mini` as evaluator, while Phase 3 uses `gpt-5.4-mini`.

Decision Criteria

The final conclusions are derived from all three evidence lenses rather than any individual metric. Retrieval metrics are primarily used to analyse retrieval behaviour and diagnose system strengths and weaknesses. Pairwise judging and manual validation are treated as the primary indicators of practical question-answering performance, since the objective of the systems is to generate useful and well-supported answers rather than reproduce predefined reference chunks.

Consequently, a system is not considered superior solely because it achieves higher retrieval scores. Preference is instead given to systems that consistently demonstrate stronger answer quality, groundedness, and completeness across multiple evaluation methods.

3.4.6 Threats to Validity

Several factors may influence the validity of the conclusions drawn from this study.

Construct Validity Evaluating Retrieval-Augmented Generation systems is inherently challenging because no single metric fully captures system quality. Retrieval metrics assess evidence coverage and ranking quality, while generation metrics assess answer quality and grounding. Furthermore, LLM-as-a-judge evaluations depend on the behaviour of another language model and may be influenced by prompt design, evaluator choice, and presentation order. To reduce dependence on any single evaluation method, the study combines automated metrics, pairwise judging, statistical testing, and manual validation.

Internal Validity The primary objective is to isolate the effect of retrieval architecture on system performance. To minimize confounding factors, both GraphRAG and VectorRAG operate on the same corpus, use the same ingestion pipeline, share the same generation models, and are evaluated using identical procedures. However, agentic retrieval systems contain non-deterministic components, including language-model reasoning and tool-calling behaviour, which may introduce variation between runs.

External Validity The evaluation is conducted on a corpus of scientific papers rather than confidential industrial documentation. Scientific literature was selected because it provides a reproducible corpus containing hierarchical structure, citations, figures, tables, and cross-document relationships. Nevertheless, the behaviour of the proposed systems may differ when applied to real-world document collections with different characteristics, update patterns, and domain-specific terminology.

Conclusion Validity The study is constrained by project duration, computational resources, and evaluation cost. As a result, the number of evaluated queries, retrieval configurations, and generation models is limited. Statistical significance testing is therefore used to distinguish meaningful performance differences from random variation. However, the findings should be interpreted as evidence within the scope of the evaluated corpus and experimental setup rather than universal conclusions about GraphRAG systems.

Dataset Construction Bias A particular threat identified during the study concerns the construction of evaluation datasets. The initial RAGAS-generated dataset was created from chunk-level document units, which may favour retrieval approaches that operate directly on chunks. To mitigate this risk, the final evaluation relies primarily on

a bootstrapped dataset generated from larger semantic units and a manually curated validation set. This reduces dependence on a single dataset construction methodology and provides multiple perspectives on system performance.

Chapter 4

System Design and Implementation

This chapter presents the design and implementation of the proposed retrieval systems. The chapter is divided into three main parts. First, the document ingestion pipeline and knowledge graph construction process are described. Second, the retrieval architectures of VectorRAG and GraphRAG are presented and compared. Finally, the implementation of the evaluation framework is outlined.

Particular emphasis is placed on maintaining a controlled experimental setup. Both retrieval systems operate on the same document corpus, ingestion pipeline, embedding model, database infrastructure, generation models, and evaluation framework. Consequently, retrieval strategy constitutes the primary experimental variable, enabling a focused comparison between graph-based and vector-based retrieval approaches.

4.1 Ingestion and Knowledge Graph Construction

This section describes the methods used to transform raw PDF documents into a structured knowledge graph with semantic embeddings.

4.1.1 Ingestion & Pre-processing

Parsing with GROBID, Docling and VLM Edge Case Handling

Transforming raw PDF documents into a structured knowledge graph is a non-trivial part of the system construction. Scientific PDFs contain heterogeneous content types, including metadata, body text, section hierarchies, references, tables, figures, and equations. No single parser was found to reliably extract all of these elements and constructing one from scratch was deemed out of scope for this thesis. The ingestion pipeline therefore combines three complementary extraction components: GROBID for bibliographic metadata, Docling for layout-aware text and structure extraction, and a vision-language model (VLM) for visually complex edge cases.

GROBID is used in the first stage of the pipeline to extract document-level metadata. In particular, it extracts the title, authors, abstract, journal, publication date, DOI, and bibliography. Rather than using GROBID for full-document parsing, only the lightweight header and reference endpoints are used. This choice was made because GROBID was reliable and efficient for bibliographic metadata, but less suitable for extracting full text and fine-grained document structure. The extracted metadata is later used to create *Paper*, *Author* and *Reference* nodes in the knowledge graph. Along with paper-level metadata such as DOI and publication date for each paper node.

Docling is used in the second stage to extract the structural and textual content of each document. The implementation traverses Docling’s structured document object directly using the `iterate_items()` interface. This makes it possible to identify typed document elements such as section headers, text blocks, lists, tables, figures, and equations. The approach also preserves page-level provenance for figures, tables, and equations, which is important for traceability and later visual processing.

Textual content is accumulated within section boundaries and then split into section-aware chunks. Chunking is necessary because retrieval systems operate over discrete retrievable units, and the size of these units affects the balance between retrieval precision and contextual completeness [20]. In this study, chunks are created with a target size of approximately 800 tokens. Which is selected to preserve sufficient local context while keeping chunks specific enough for later retrieval and embedding. Chunks do not cross section boundaries, since preserving section structure is important for both graph construction and retrieval. Chunks, figures and tables are assigned individual IDs and graph provenance, discussed further in 4.1.2

Tables extracted by Docling are validated before being inserted into the graph. A table is considered valid if it contains a recognizable table structure, at least two rows and two columns, and less than 30% empty cells. Valid tables are stored directly as table nodes. Tables that do not satisfy these criteria are marked as malformed and routed to the edge case handler.

Figures are detected with the help of Docling’s figure elements, and then sent to the edge case handler for processing. Equations are detected and represented as equation nodes with section and page provenance. Docling provides functionality for full equation inclusion, but due to compute and time constraints the actual equation contents were ignored and entity placeholders were used instead. Equation processing was computationally expensive, and individual equations were not expected to embed reliably as natural language evidence. Instead, equation nodes are preserved mainly as structural and provenance markers in the graph.

Edge Case Handler The third stage of the ingestion pipeline handles edge cases using a vision-language model. Docling often provides useful provenance information even when it cannot reliably extract the content itself. This is particularly valuable for malformed tables, visually complex figures, or scanned and missed pages. For such elements, the relevant PDF page is rendered as an image and sent to a locally hosted vision language model (VLM), the exact model name is in Table 4.5. Malformed tables are reconstructed, while figures are converted into structured natural language descriptions. Tables are also summarized into natural language to produce an embeddable representation, since raw tables are poorly aligned with natural language queries. The prompts used for figure description, malformed table reconstruction, and table summarisation are provided in Appendix B. All requests to the VLM consist of two components:

- A rendered image of the relevant PDF page.
- A structured prompt defining the extraction task.

Pages are rendered using PyMuPDF at a controlled resolution (150 DPI for figures and 100 DPI for tables) before being encoded as base64 images and transmitted to the VLM.

After extraction and edge case handling, an assembler merges the outputs and assigns every node a provenance tag, indicating whether it originated from GROBID, Docling, or the VLM. This distinction is important because directly extracted content and model-generated content have different reliability profiles. The final representation consists of explicit nodes and edges and will be covered in more detail in the next section.

4.1.2 Knowledge Graph Construction

After the parsing and edge-case handling stages, the extracted document representation is transformed into a Neo4j knowledge graph. The purpose of the knowledge graph is twofold. First, it provides a structured representation of the corpus by preserving document hierarchy, citation relationships, and links between textual and visual elements. Second, it also serves as a graphical inspection tool, allowing the ingested entities and their relationships to be inspected and evaluated in an intuitive way. This was useful during development to verify that papers, sections, chunks, figures, tables, and citation links had been correctly created and connected. Figure 4.1 shows an expanded view of a single paper in the Neo4j database.

Graph Schema

The graph schema is designed to represent scientific documents at multiple levels of granularity. The main node types are **Paper**, **Author**, **Reference**, **Section**, **Chunk**,

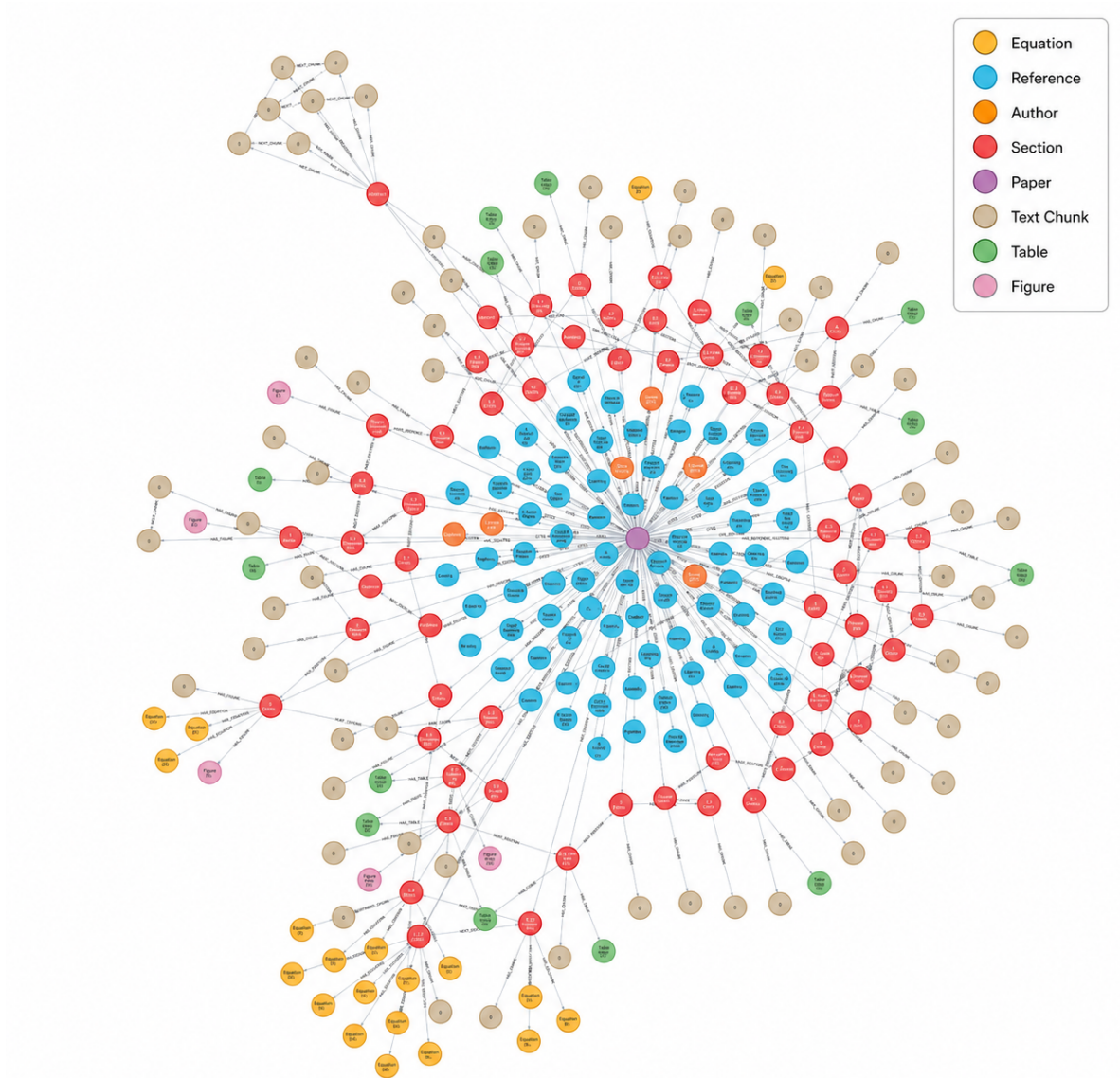


Figure 4.1: Example of an expanded paper visualized in Neo4j. Each paper in the corpus is ingested into the same fine-grained graph structure, consisting of interconnected nodes and relationships representing document elements such as sections, chunks, figures, tables, equations, authors, and references. This structure enables the graph-based retrieval agent to traverse the document in downstream retrieval processes.

Table, **Figure**, **Equation**, and **Community**. Paper nodes represent individual scientific papers and contain metadata such as title, abstract, DOI, journal, and publication date. Author and Reference nodes represent bibliographic information extracted from the documents. Section and Chunk nodes represent the internal structure of each paper, while Table, Figure, and Equation nodes preserve non-standard content types and their location in the document. The graph uses typed relationships to encode document structure, cross-document relations, sequential order, and links between textual and non-textual document elements. The relationship types used in the knowledge graph are summarized in Table 4.1.

Table 4.1: Relationship types used in the knowledge graph.

Relationship	Direction	Purpose
HAS_AUTHOR	Paper $\rightarrow$ Author	Connects papers to authors
CITES	Paper $\rightarrow$ Paper/Reference	Represents citation relations
HAS_SECTION	Paper $\rightarrow$ Section	Encodes document hierarchy
HAS_CHUNK	Section $\rightarrow$ Chunk	Connects sections to text chunks
NEXT_SECTION	Section $i \rightarrow$ Section $i+1$	Preserves section reading order
NEXT_CHUNK	Chunk $i \rightarrow$ Chunk $i+1$	Preserves chunk reading order
HAS_TABLE	Section $\rightarrow$ Table	Links sections to tables
HAS_FIGURE	Section $\rightarrow$ Figure	Links sections to figures
HAS_EQUATION	Section $\rightarrow$ Equation	Links sections to equations
DESCRIBED_IN	Figure/Table $\rightarrow$ Chunk	Links visual elements to referencing text
SIMILAR	Paper $\leftrightarrow$ Paper	Captures semantic similarity between papers
BELONGS_TO	Paper $\rightarrow$ Community	Links papers to Leiden communities

This schema allows the system to support both fine-grained retrieval over text chunks and broader graph traversal across papers, sections, citations, and communities. Visual representations of the relationship edges can be found in Appendix C.1.5

Graph Loading and Idempotency

As mentioned, the outputs from GROBID, Docling, and the VLM are assembled into a JSON representation containing explicit nodes and edges, an example of this format can be found in Appendix F. This intermediate step separates extraction from database insertion, making the pipeline easier to inspect, debug, and reproduce.

The assembled graph is then loaded into Neo4j using Cypher **MERGE** statements, eliminating the risk of duplicate node and relationship creation. Additionally, unique constraints are created on the `id` property of each node type before insertion. These implementations makes the loading process idempotent, meaning that the same document can be reprocessed and reloaded without creating duplicate nodes. Nodes and edges are loaded in batches to reduce transaction overhead and make the pipeline more stable for

larger corpora.

Reference Resolution

Bibliographic references are initially stored as **Reference** nodes, since many cited papers are not part of the corpus. After all papers are ingested, references are matched against existing **Paper** nodes using normalized title matching and fuzzy token overlap. Successful matches are converted into direct **Paper**–to–**Paper** CITES edges seen in Figure C.4, while unresolved references remain as standalone bibliographic nodes.

Embedding Strategy

In order to support semantic retrieval we create dense vector embeddings within the graph database. Embeddings are generated for chunk text, figure descriptions, table summaries, and paper abstracts. All embeddings are stored directly as properties on their corresponding Neo4j nodes and indexed using Neo4j vector indexes with cosine similarity. The same embedding model is used throughout the system to ensure a consistent semantic embedding space across all entity types. All models used are listed in Table 4.5.

For textual content, the embedding strategy decouples storage chunks from embedding chunks. As previously described, the ingestion pipeline employs section-aware chunking to preserve the natural hierarchical structure of scientific papers. However, this introduced a limitation with hard cut-offs and no overlap between chunks. It was also noted that due to imperfect parsing some very small subsections were parsed as chunks. Naively embedding each stored graph chunk independently may produce suboptimal vector representations, particularly for short or context-dependent chunks. Prior work on late chunking recognizes this, and argues that independently encoded chunks can lose contextual information from surrounding text, which can negatively affect retrieval quality [20].

This addresses a trade-off in retrieval system construction:

- Smaller chunks improve retrieval precision but often lack sufficient semantic context.
- Larger chunks provide richer representations but reduce granularity and increase noise.
- The graph agent requires section-level structure to perform granular traversal over the document graph.
- No overlap between chunks could reduce semantic relevance for cosine search

To address this, we use a greedy bin-packing strategy for constructing embedding windows. The approach is inspired by late chunking [20], but adapted to the requirements of graph-based retrieval. Instead of embedding each stored graph chunk independently, consecutive chunks are grouped into larger embedding windows while preserving section boundaries.

Section chunks are processed sequentially within each section and concatenated until the embedding window approaches an approximate maximum size of 800 tokens. Chunks exceeding the maximum token limit are embedded individually, while trailing windows below the minimum threshold are merged backward into the previous window. All chunks belonging to the same embedding window share a common `embedding_id` and embedding vector. The procedure is summarized in Algorithm 2.

Figures are embedded using the natural language descriptions generated by the vision-language model. Tables are embedded using generated natural language summaries. Paper abstracts are embedded separately to support document-level semantic similarity and downstream community construction. All embeddings are stored directly on the corresponding entity node as metadata, illustrated in Table 4.2.

Table 4.2: Example Paper node properties in the Neo4j knowledge graph.

Property	Value
<code>id</code>	<code>paper_A_performance_evaluation_of...</code>
<code>title</code>	A performance evaluation of open source graph databases
<code>doi</code>	10.1145/2567634.2567638
<code>publication_date</code>	2014-02-16
<code>source</code>	<code>grobid_extracted</code>
<code>isolated</code>	<code>true</code>
<code>abstract</code>	With the proliferation of large, irregular, and sparse [...]
<code>abstract_embedding</code>	<code>[-0.0066, 0.0573, -0.0304, ...]</code> (768-dimensional vector)

Community Construction

One advantage of representing the corpus as a graph is the ability to construct multi-level structure in the data. In this work the structure takes the form of communities, defined as clusters of semantically similar and co-cited papers identified through community detection on the paper graph. Community detection is performed using the Leiden algorithm which is a graph partitioning algorithm designed to identify densely connected groups of nodes, while maximizing community quality through modularity optimization. Compared to earlier methods such as Louvain, Leiden guarantees better connected communities and reduces the risk of producing disconnected clusters [21]. Rather than

searching directly over all chunk embeddings for every query, the graph-based pipeline can first identify relevant communities and then traverse into selected papers, sections, and chunks. This reduces the number of candidate documents considered during later retrieval stages and provides a structured alternative to flat similarity-search [4]. Since high-dimensional similarity-search can become less discriminative as dimensionality increases, particularly when many weakly relevant candidates are present. Reducing the candidate space through graph-based filtering could also mitigate some practical effects of high-dimensional vector retrieval [22]. The Leiden algorithm operates on weighted graph relationships. In this work, the edge weights are constructed from a combination of explicit citation relationships and semantic similarity relationships between papers. Citation relationships **CITES** provide deterministic structural information and were therefore assigned a higher relative importance than semantic similarity relationships **SIMILAR**, which are computed from cosine similarity between paper abstract embeddings. The semantic similarity edges complement the citation structure, since citation relationships alone were observed to leave some papers weakly connected or entirely isolated, particularly in smaller corpora. The resulting graph used for community detection therefore consists of a weighted combination of **CITES** and **SIMILAR** relationships. For each valid community, an LLM generates a short technical summary based on the titles and abstracts of the papers within the cluster. These summaries are subsequently embedded and stored as **Community** nodes in the graph. Each paper is connected to its corresponding community through a **BELONGS_TO** relationship. In addition, vector indexes are created over community embeddings, enabling the graph-based retrieval pipeline to efficiently identify relevant topical regions of the corpus during retrieval. The hyperparameter we used when running this algorithm can be found in Table 4.3 and the resulting distribution of communities are presented in Table 4.4.

Table 4.3: Leiden community detection hyperparameters.

Parameter	Value	Description
maxLevels	10	Maximum refinement passes
γ (gamma)	1.0	Resolution — controls community granularity
θ (theta)	0.01	Low Randomness for reproducibility
WeightProperty	weight	Edge property used as weight
CITES default weight	0.7	Citation edges drive community formation
SIMILAR default weight	0.3	Semantic edges add relations

Table 4.4: Community distribution after Leiden clustering

Community	Number of papers	Share of corpus
World Models and Agentic Systems	21	20.4%
Knowledge Graphs and RAG	10	9.7%
Graph Databases and Graph Theory	7	6.8%
Knowledge Graph Reasoning and Completion	20	19.4%
LLM and RAG surveys	28	27%
Bayesian Optimization	9	8.7%
3D Object Detection in Autonomous Vehicles	4	3.9%
Papers not in any community	4	3.9%

Resulting Graph Representation

The final knowledge graph combines document structure, citation relationships, semantic similarity, visual content, and community-level abstractions. This representation supports two retrieval modes. The vector-based system uses the graph primarily as a vector store by querying chunk embeddings through cosine similarity. The graph-based system uses the same database but additionally exploits explicit relationships such as citations, sections, sequential chunk links, figure/table references, and community membership.

This shared representation ensures that both systems operate on the same underlying corpus and processed content, while differing only in how they retrieve information from it.

4.2 Retrieval Design

This section describes the design of both retrieval systems evaluated in this thesis. Although VectorRAG and GraphRAG share several infrastructure components they employ fundamentally different retrieval strategies. VectorRAG performs flat semantic search over chunk embeddings with iterative refinement, while GraphRAG navigates the knowledge graph’s hierarchical structure through LLM-driven tool calling. This architectural separation is intentional: the goal is to isolate the effect of retrieval strategy on answer quality, with all other variables held constant.

We first describe the components common to both paths, then detail each retrieval strategy individually. Design trade-offs and their implications for answer quality are discussed further in chapter 6.

4.2.1 Shared Infrastructure

As described in Section 4.1.2, both retrieval paths operate over the same knowledge graph, use the same embedding and generation models, and produce output in the same format for evaluation. This subsection focuses on describing the components that are identical across both systems.

4.2.2 Query Decomposition

Both retrieval systems use query decomposition for complex questions. Query decomposition refers to splitting the original user query into a set of more focused sub-queries, where each sub-query targets a specific part of the information need.

For example, a query such as:

What is the difference in results between Paper A and Paper B?

can be decomposed into:

- What are the results reported in Paper A?
- What are the results reported in Paper B?

This is useful because the original query may require evidence from multiple papers, sections, or semantic regions of the corpus. Without explicit decomposition, a single retrieval query may only cover part of the required information. Decomposition therefore increases the likelihood that each relevant aspect of the query is represented in the retrieval process.

In this system, each sub-query is executed as an independent retrieval run. The retrieved evidence from all sub-queries is then merged, deduplicated, and passed to the shared context assembly and generation pipeline.

Similarity Search

Both systems use embedding-based similarity search, but at different granularities and for different purposes. The Vector RAG path performs hybrid search directly over chunk embeddings. The GraphRAG path uses cosine similarity over community summary and paper abstract embeddings for navigation decisions, identifying which communities and papers to explore, but does not perform any chunk-level similarity search.

Answer Generation

Both systems use the same language model and prompt template for final answer generation. The generator receives the assembled context and the original query, and produces a cited, synthesized answer. The generation model, prompt, and token budget are identical across conditions, ensuring that any differences in answer quality are attributable to differences in the retrieved context rather than differences in generation.

Models

To facilitate fair comparison, both retrieval systems use the same models for shared pipeline components. Table 4.5 summarizes the main models and their roles in the system.

Table 4.5: Models used in the retrieval and generation pipeline.

Component	Model
Embedding	sentence-transformers/all-mpnet-base-v2
Reranking	ms-marco-MiniLM-L-6-v2
Generation and agent reasoning	Mistral-7B-Instruct / GPT-OSS-120B
Visual processing	GLM-4.1V-9B-Thinking
Evaluation	gpt-4o-mini / gpt-5.4-mini

Observability and Tracing

All retrieval runs were instrumented with Langfuse, an open-source LLM observability platform. Each agent call, tool invocation, and LLM response was recorded as a trace, enabling post-hoc inspection of the full retrieval path for any query. This was critical during development for diagnosing agent behavior, identifying repeated tool calls, premature finish decisions, and empty LLM responses that would not have been visible from evaluation metrics alone. Langfuse traces also provided timing data for each pipeline stage, informing decisions about hop budgets and model selection.

4.2.3 Vector RAG: Iterative Hybrid Search

The Vector RAG path implements a multi-loop retrieval architecture in which a sequence of specialized agents collaborate to iteratively search, evaluate, and refine the retrieval query until the retrieved context is deemed sufficient or a stopping criterion is met.

Architecture Overview

Each retrieval loop executes the following agent sequence:

1. **QueryAgent** prepares the search query. On the first loop, the original user query is passed through unchanged. On subsequent loops, the agent rewrites the query based on diagnostic output from the ReasonAgent, focusing the search on information identified as missing from previous iterations.
2. **SearchAgent** executes a hybrid search combining dense vector retrieval and sparse keyword matching. Results from both search modalities are merged using Reciprocal Rank Fusion (RRF) and reranked with a cross-encoder model.
3. **JudgeAgent** assembles the accumulated context, generates a draft answer, and scores the retrieval quality using four metrics: faithfulness (LLM-judged), context relevancy (embedding similarity), answer relevancy (embedding similarity), and a holistic sufficiency score (LLM-judged). These are combined into a weighted aggregate score.
4. **Exit check.** If the aggregate score exceeds the confidence threshold, the pipeline exits with a “sufficient” verdict. If the score has not improved for a configurable number of consecutive loops, the pipeline exits as “stuck” and reverts to the best context seen so far. If the maximum loop count is reached, the pipeline exits with “max_loops”.
5. **ReasonAgent** (if continuing) diagnoses why the retrieval was insufficient. It identifies the weakest scoring metric, hypothesises what information is missing, and produces a query rewrite hint and search strategy for the next loop.

The pipeline tracks the best-scoring context across all loops and reverts to it upon exit, ensuring that the generator always receives the highest-quality context observed during the run, even if later loops degraded the score.

Hybrid Search

This work implements a hybrid retrieval setup to ensure competitive results. The setup provides a robust baseline by combining semantic similarity, lexical matching, and relevance scoring through hybrid retrieval and rank fusion methods [23, 24, 7].

Vector search. Dense retrieval over chunk embeddings using cosine similarity. The query is embedded using the shared embedding model and compared against all chunk embeddings in the Neo4j vector index. This captures semantic similarity, retrieving chunks that are conceptually related to the query even when they share no keywords.

BM25 keyword search. Sparse retrieval using Neo4j’s full-text index over chunk text. This captures exact terminology and named entities that embedding models may not represent precisely, such as specific model names, dataset identifiers, or mathematical notation.

Reciprocal Rank Fusion. Results from both modalities are merged using a weighted RRF formulation. Each candidate receives a score that combines its rank position and raw score from each modality, with configurable weights (default: 0.6 vector, 0.4 BM25). This produces a single ranked list that benefits from both semantic and lexical matching.

Cross-encoder reranking. The merged candidates are reranked using a cross-encoder modelss that jointly encodes the query and each candidate passage, producing a more accurate relevance score than the bi-encoder embeddings used in the initial retrieval. The top- k candidates after reranking are retained for context assembly.

Iterative Refinement

The iterative loop architecture is motivated by the observation that a single retrieval pass may not capture all information needed to answer complex queries. The ReasonAgent diagnoses retrieval failures at the metric level, for example, identifying that context relevancy is low (the retrieved chunks are off-topic) versus answer relevancy being low (the chunks are relevant but insufficient to answer the specific question). This diagnosis guides the QueryAgent’s rewrite, which tries to target the identified gap rather than simply rephrasing the original query.

On loop 2+, the SearchAgent excludes previously seen chunks using their embedding identifiers, ensuring that each loop retrieves novel content. When the ReasonAgent determines that the query should be decomposed into sub-questions, the SearchAgent executes parallel searches for each sub-query and combines the results.

4.2.4 Graph RAG: Community-First Hierarchical Traversal

The Graph-RAG path replaces the iterative hybrid search with a fundamentally different retrieval strategy: an LLM-driven agent navigates the knowledge graph’s hierarchical structure from communities down to individual sections and chunks, using tool calling to interact with the graph at each level.

Architecture Overview

The GraphRAG retrieval path consists of two main components:

1. **GraphAgent:** An LLM-based tool-calling agent that navigates the knowledge graph hierarchy. The agent receives a query and a set of tools for interacting with the graph, then autonomously decides which communities, papers, and sections to explore.
2. **GraphSearchOrchestrator:** A wrapper that manages the agent’s execution, collects the retrieved evidence, and applies the shared context assembly pipeline before passing the result to the generator.

Unlike the VectorRAG pipeline, the GraphRAG configuration does not use iterative query rewriting, a judge agent, or a reason agent. Instead, retrieval control is delegated to the graph traversal process itself, where the LLM selects relevant communities, papers, sections, and chunks through tool-calling decisions. The graph structure thereby acts as an implicit retrieval constraint and refinement mechanism.

Hierarchical Traversal

The GraphAgent follows a top-down traversal of the knowledge graph hierarchy:

Query Decomposition. The first step in the GraphRAG pipeline is query decomposition. The input query is decomposed into a configurable number of smaller sub-queries, each targeting a distinct aspect of the original information need. Each sub-query is then processed independently through the retrieval pipeline with its own retrieval state and context. This allows broad or multi-hop questions to be transformed into more focused retrieval tasks before the retrieved evidence is combined in the final answer generation step.

Community selection. The agent then searches for relevant communities in the graph by comparing the query embedding against community summary embeddings using cosine similarity. The agent receives the top-ranked communities with their summaries and paper counts, and selects 2–3 communities to explore further. This step provides topic-level routing that narrows the search space from the full corpus to a relevant subset.

Paper selection. For each selected community, the agent retrieves the papers belonging to that community, ranked by the cosine similarity between their abstract embeddings and the query embedding. The agent reads paper titles, abstracts, and relevance scores, then selects the most relevant papers for deeper exploration. This step leverages the graph’s community structure to present papers in a topically coherent context, rather than ranking them against the full corpus.

Section navigation. For each selected paper, the agent retrieves all section headings, along with chunk count and semantic similarity score between the query embedding and the first chunk of each section. Sections are returned pre-ranked by similarity score, with low-value sections such as appendices and reference lists explicitly flagged. The agent then selects which sections to retrieve content from, the step where its reasoning ability is most critical. Section headings such as 3.1 Method or 4.2 Results provide structural cues that the agent must interpret in the context of the query, combining the similarity signal with domain knowledge about academic paper structure.

Content retrieval. For each selected section, the agent retrieves top-n chunks by similarity score. If those chunks reference figures or tables, the agent searches and retrieves the connected entities. This is the only step that produces actual evidence for the final answer. The retrieved section text, along with any connected figure and table summaries, is stored in an evidence buffer.

Coverage Check and Cross-Paper Retrieval. At this stage the agent reviews its findings. If it finds information that implies that another referenced paper could help answer the query. It looks through the references in that paper, if there's a match in the database the agent can traverse directly to that paper. And start searching through its sections for relevant information.

The Agent Toolbox

The GraphAgent interacts with the knowledge graph through a set of tools, each classified by its role in the retrieval process:

- **Navigation tools** (`search_communities`, `get_community_papers`, `get_paper_sections`, `get_citations`) provide information for decision-making but do not produce evidence. Their outputs enter the agent's working memory only, informing subsequent tool selections without contributing to the final context.
- **Evidence tools** (`get_section_content`) retrieve actual text content that becomes part of the final context passed to the generator. This is the only tool that populates the evidence buffer.
- **Supplementary tools** (`get_section_figures_tables`) attach figure and table summaries to previously retrieved sections, enriching the evidence without adding standalone chunks.
- **Control tools** (`finish`, `check_coverage`) manage the agent's execution flow. The coverage check tool provides the agent with a summary of what it has collected so far, enabling it to assess whether further exploration is needed before terminating.

The distinction between navigation and evidence tools implements a working memory versus evidence split: the agent accumulates navigational context (community summaries, paper abstracts, section headings) to inform its decisions, but only the content explicitly retrieved via evidence tools enters the final context. This prevents the agent from polluting the generator’s context with navigational metadata.

Behavioural Controls

Several mechanisms constrain the agent’s behavior to ensure efficient and non-redundant retrieval, following the observations of Yao et al. [12].

- **Duplicate call detection.** Each tool call is recorded as a key combining the tool name and its arguments. Repeated calls with identical arguments are intercepted and return a message directing the agent to proceed to the next step, preventing the agent from wasting hops on redundant operations.
- **Community search guard.** The community search tool can only be called n -times per retrieval. Subsequent calls are blocked and the agent is redirected to use already-retrieved community results.
- **Finish guards.** The agent is prevented from calling `finish` until it has retrieved evidence using the `get_section` tool call. Additionally, if the agent attempts to finish with fewer than n papers, it receives a message prompting further exploration. This mitigates the observed tendency to stop after finding a single relevant source, which is insufficient for multi-hop queries requiring synthesis across papers.
- **Hop budget.** The agent has a configurable maximum number of tool calls (default: 20). If the budget is exhausted, the orchestrator proceeds with whatever evidence has been collected.

These controls were developed iteratively in response to observed agent behavior during development. Without them, the agent frequently repeated community searches, stopped after retrieving a single section, or spent the majority of its hop budget generating reasoning text rather than calling tools.

Summary

The retrieval architectures represent two fundamentally different approaches to information retrieval over the same knowledge base. Vector RAG relies on iterative refinement over a flat embedding space using hybrid search and iterative LLM-driven refinement, whereas GraphRAG exploits the explicit structure of the knowledge graph through hierarchical traversal and tool-guided evidence collection. Despite these differences, both

systems share the same ingestion pipeline, database, context assembly, and generation stages, enabling a controlled comparison where retrieval strategy is the primary experimental variable. Table 4.6 summarises the key architectural differences between the two retrieval paths.

Table 4.6: Comparison of retrieval strategies.

Aspect	VectorRAG	GraphRAG
Search strategy	Corpus-wide hybrid search	Hierarchical graph traversal
Agent framework	Multi-agent loop	Single tool-calling agent
Iteration mechanism	Judge $\rightarrow$ reason $\rightarrow$ rewrite	Decomposed pass $\rightarrow$ Parallel trav. $\rightarrow$ Coverage Check
Structural awareness	Seen metadata	Communities, citations, and hierarchy
Figure/table access	Through description embeddings	Via graph relationships
Ranking	RRF + cross-encoder	Similarity + Reasoning

4.3 Evaluation Implementation

This section describes the practical implementation of the evaluation framework, including the RAGAS-based evaluation pipeline, the comparative evaluation setup, and the applied statistical analysis methods.

4.3.1 Evaluation Framework Architecture

To support large-scale experimentation, a dedicated evaluation framework was implemented. The framework automates metric computation, comparative judging, statistical testing, and result aggregation across all retrieval configurations.

For each evaluated query, the framework stores the retrieved context, generated answer, retrieval metadata, and execution traces. These outputs are subsequently processed by automated evaluation pipelines, including RAGAS-based retrieval and generation metrics, pairwise LLM-as-a-judge comparisons, and statistical significance testing.

The evaluation framework was designed to support reproducibility and robustness during long-running experiments. Results are stored incrementally, intermediate outputs are preserved, and incomplete evaluations can be resumed without re-running the entire experiment. This enables efficient evaluation of large query sets while maintaining consistent experimental conditions across all system configurations.

4.3.2 RAGAS implementation

RAGAS Pipeline

The evaluation framework is built on the RAGAS library and operates as a post-hoc process, fully decoupled from the retrieval pipeline. The retrieval pipeline outputs structured JSON files containing metadata and ground truth references and answers carried over from the evaluation dataset. These JSON files serve as the input to the evaluation pipeline, enabling repeated evaluation runs without re-executing the retrieval system.

Each evaluation run loads a results file, iterates over all query records, and computes retrieval and generation metrics for each entry. To enable parallelization, the metric functions are implemented as asynchronous co-routines wrapping the corresponding RAGAS metric classes documented in Section 3.4.3. Each metric is instantiated with the evaluation LLM and called via `metric.ascore()`, which internally decomposes the evaluation task into one or more LLM calls depending on the metric type.

Results are accumulated into a list of per-query dictionaries containing all metric scores, verdict classifications (pass, borderline, fail), and metadata. The final output is written as a JSON file for downstream statistical analysis.

Execution Protocol

All evaluation LLM calls use temperature 0 to ensure deterministic decoding and reproducible scores across runs. The evaluation LLM's maximum output token limit is set to 100,000 tokens to prevent truncation of the model's reasoning chain during claim-by-claim verification, which was observed to cause incomplete evaluations at lower limits.

The retrieval and generation metric groups are executed in parallel using Python's `asyncio.gather` with `return_exceptions=True`, so that a failure in one metric does not prevent the remaining metrics from completing. Concurrency is controlled via `asyncio.Semaphore` set to match the number of available inference endpoints, preventing server overload from simultaneous requests. When a metric call fails, the exception is logged and the corresponding score is recorded as `None` rather than halting the evaluation.

Evaluation of the graph-enabled and vector-only conditions is run as two separate batch processes, each producing an independent output file. Both conditions evaluate the same set of queries to ensure paired statistical comparison.

Models

A deliberate design choice was made to separate the models used for pipeline execution from those used for evaluation. Due to the large number of LLM calls required by the RAGAS evaluation protocol, the evaluation was performed using an API-hosted model. GPT-4o mini was selected as a cost-efficient evaluator model.

This separation not only dramatically reduced the evaluation runtime but also reduces the risk of circular evaluation. If the same model is used both to generate answers and evaluate them, the evaluator may share the generator’s blind spots and systematically overlook errors that the generator is prone to making.

Embedding-based metrics use the `all-mpnet-base-v2` sentence transformer model, consistent with the embeddings used in the retrieval pipeline itself.

4.3.3 Comparative Evaluation and Sensitivity Analysis

Comparative Evaluation The LLM-evaluator was instructed to assess the systems relative to the original query, retrieved evidence, factual grounding, completeness, and answer quality rather than stylistic preferences.

The evaluator produced a structured JSON output containing the winning system, the losing system, and a short textual justification. In cases where both systems were judged equivalent, the result was recorded as a tie. Queries producing malformed outputs or incomplete responses were flagged and excluded from the comparative analysis.

To reduce position bias, each system comparison was evaluated in both presentation orders, following the dual-order protocol proposed by Zheng et al. [14]. The resulting judgments were later averaged into win/loss statistics and used as input for the binomial significance test described in the statistical analysis section of the methodology 3.4.4.

4.3.4 Statistical Test Implementation

All statistical analyses were conducted using SciPy and NumPy. Tests were applied independently for each evaluation metric. No correction for multiple comparisons was applied due to the exploratory nature of the analysis and the limited evaluation size. Consequently, individual p-values should be interpreted cautiously and together with their corresponding effect sizes.

Chapter 5

Results

This chapter presents a three-phase comparison of GraphRAG and VectorRAG across three retrieval ablations: *minimal*, *default*, and *thorough*. The evaluation combines automated RAGAS-style metrics [13], pairwise LLM-as-judge comparisons and statistical significance tests.

5.1 Phase 1: Ablation selection

Phase 1 selects one ablation per (system, model) cell from twelve configurations (2 systems $\times$ 2 models $\times$ 3 ablations). Each cell is evaluated on the 83-query bootstrapped set using the three evidence lenses introduced in Section 3.4.5. When evidence lenses disagreed during the ablation stage, greater weight was assigned to the composite metric scores, since the objective of Phase 1 was configuration screening rather than final system selection. Holistic answer quality was evaluated more thoroughly in the later comparison phases.

5.1.1 Selection summary

Table 5.1 lists the recommended finalist per cell. Table 5.2 reports automated composites and Holm-significant metric win counts for all ablations; Table 5.3 summarises order-averaged LLM judge outcomes for each pairwise comparison.

Table 5.1: Phase 1 recommended ablations per system and generation model.

System	Model	Ablation	Confidence	Mean chunks
GraphRAG	OSS	default	high	7.1
GraphRAG	Mistral	thorough	high	12.8
VectorRAG	OSS	minimal	medium	4.2
VectorRAG	Mistral	thorough	high	20.1

Table 5.2: Phase 1 automated metric composites and Holm-significant win counts (83 queries). Bold marks the selected finalist per cell.

Cell	Ablation	Retr. composite	Gen. composite	Stats sig. wins
Graph + OSS	minimal	0.700	0.657	1
	default	0.755	0.693	3
	thorough	0.733	0.665	2
Graph + Mistral	minimal	0.598	0.651	2
	default	0.573	0.621	1
	thorough	0.745	0.692	9
Vector + OSS	minimal	0.834	0.674	1
	default	0.835	0.685	0
	thorough	0.831	0.733	3
Vector + Mistral	minimal	0.826	0.703	2
	default	0.828	0.750	2
	thorough	0.828	0.732	3

Table 5.3: Phase 1 order-averaged LLM judge answer wins per pairwise ablation comparison (`gpt-4o-mini`, dual order).

Cell	Pair	Wins A	Wins B	Ties	Answer winner
Graph + OSS	<code>minimal_vs_default</code>	33	48.5	1.5	<code>default</code>
	<code>minimal_vs_thorough</code>	25.5	55.5	2	<code>thorough</code>
	<code>default_vs_thorough</code>	28.5	50	4.5	<code>thorough</code>
Graph + Mistral	<code>minimal_vs_default</code>	36.5	46.5	0	<code>default</code>
	<code>minimal_vs_thorough</code>	0.5	82.5	0	<code>thorough</code>
	<code>default_vs_thorough</code>	0.5	82	0	<code>thorough</code>
Vector + OSS	<code>minimal_vs_default</code>	35	43.5	4.5	<code>default</code>
	<code>minimal_vs_thorough</code>	21.5	60.5	1	<code>thorough</code>
	<code>default_vs_thorough</code>	25.5	52.5	5	<code>thorough</code>
Vector + Mistral	<code>minimal_vs_default</code>	22	60	1	<code>default</code>
	<code>minimal_vs_thorough</code>	17.5	65.5	0	<code>thorough</code>
	<code>default_vs_thorough</code>	23.5	59	0.5	<code>thorough</code>

The four subsections below walk through each cell with supporting figures: metric heatmaps, judge win distributions, Holm-significance heatmaps, and cost–quality trade-offs.

5.1.2 GraphRAG + OSS $\rightarrow$ `graph_default`

`graph_default` achieves the highest retrieval composite (0.755) and generation composite (0.693) and leads on Holm-significant metric wins (3 vs. 1–2; Table 5.2, Figure 5.2b). The metric heatmap (Figure 5.1) shows `default` leading across most per-metric means, while `thorough` improves context recall and faithfulness at the cost of answer relevancy (0.711 vs. 0.754) and generation composite.

Order-averaged judge comparisons (Table 5.3, Figure 5.2a) favour `default` over `minimal` (48.5 vs. 33.0 answer wins) but favour `thorough` over `default` in direct comparison (50.0 vs. 28.5). We nevertheless select `default` because it dominates automated generation quality and statistics, beats `minimal` clearly on judge answers, and avoids the retrieval-cost penalty of `thorough` (≈ 7.1 vs. ≈ 12.9 mean chunks; Figure 5.3). More retrieval depth does not improve end-to-end quality for OSS GraphRAG.

5.1.3 GraphRAG + Mistral $\rightarrow$ `graph_thorough`

This cell shows the clearest ablation result in the study: all three lenses unanimously favour `graph_thorough`. `graph_default` retrieval composite (0.573) is inadequate under Mistral; `thorough` leads on both composites (0.745 / 0.692) with 9 Holm-significant metric wins (Table 5.2, Figure 5.5b). The judge wins all three pairwise comparisons with

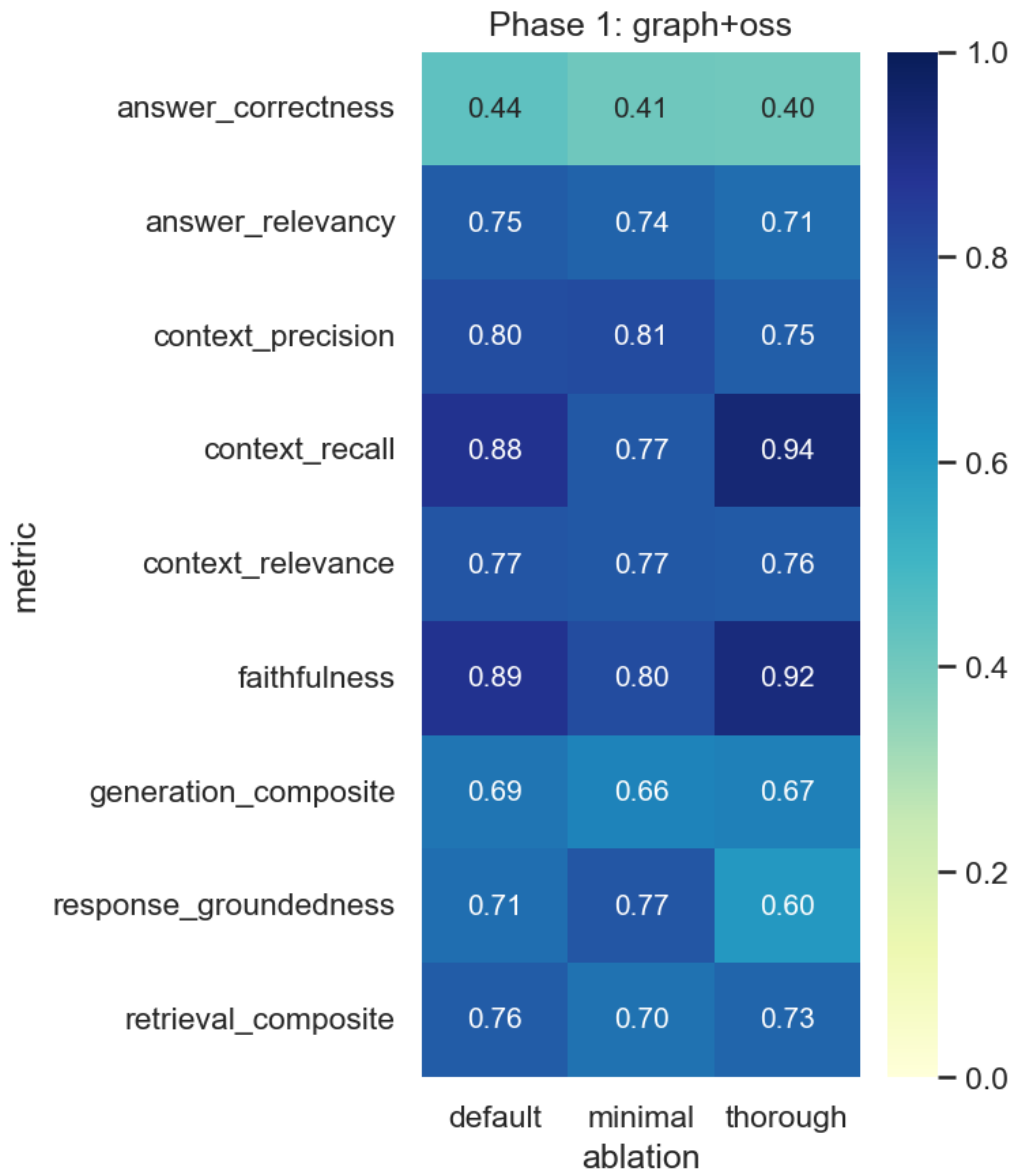
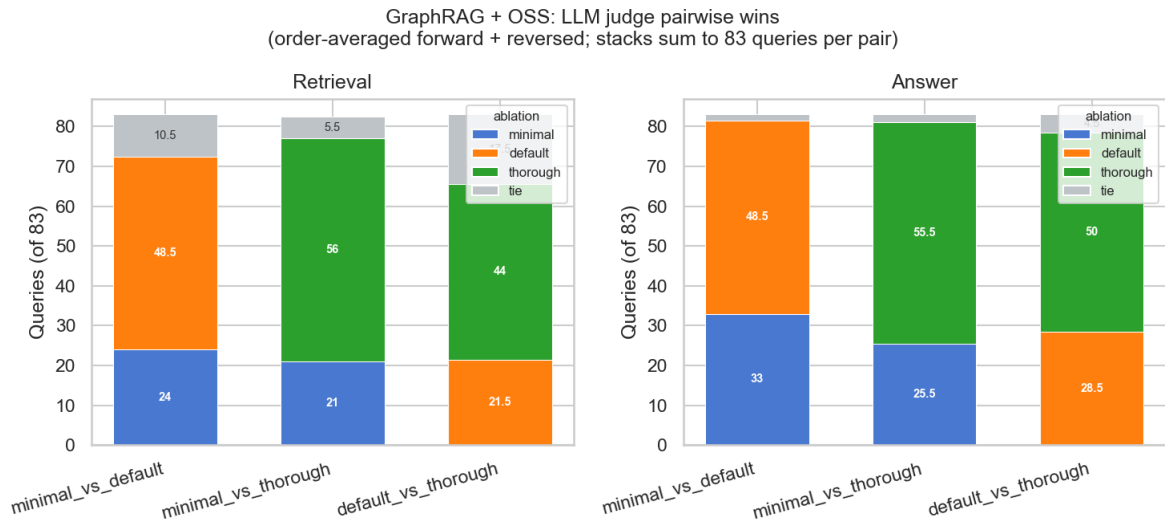
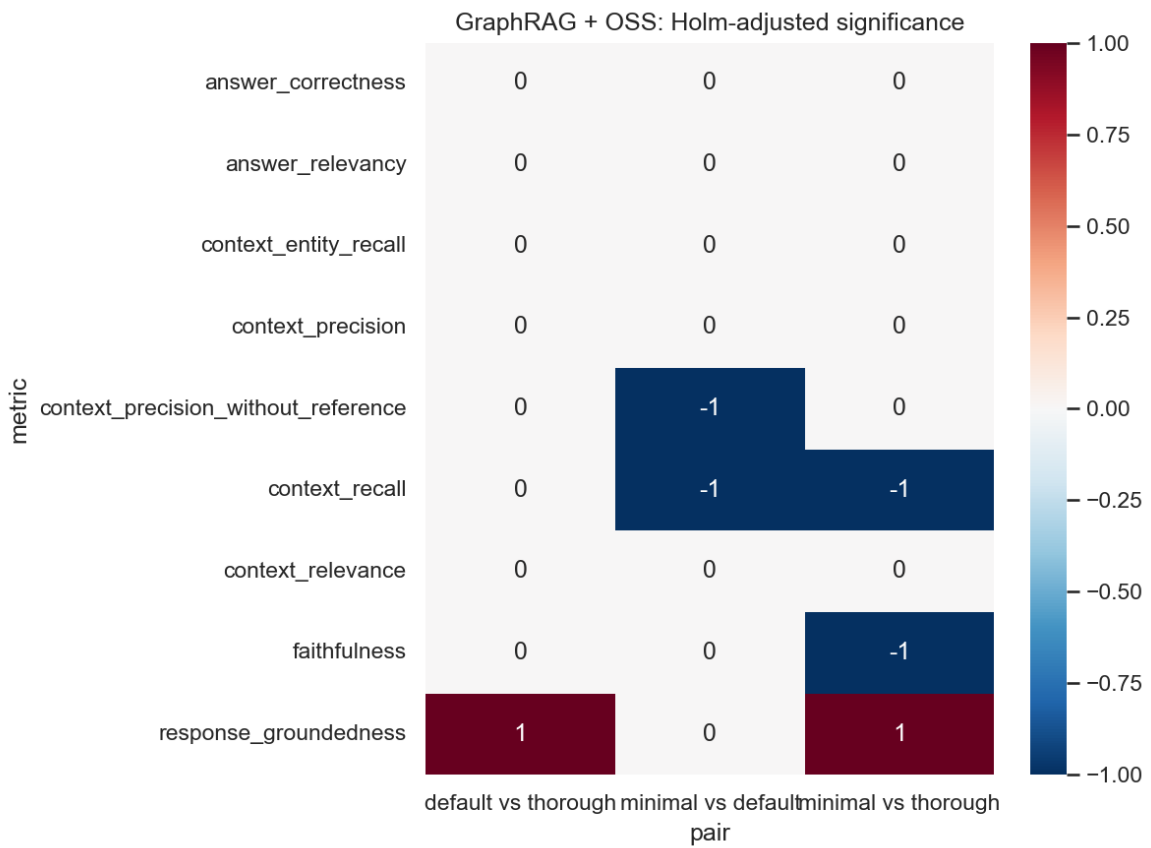


Figure 5.1: GraphRAG + OSS: Phase 1 per-metric means across ablations.



(a) Order-averaged judge wins



(b) Holm-significant metric wins

Figure 5.2: GraphRAG + OSS: LLM judge and statistical evidence for ablation selection.

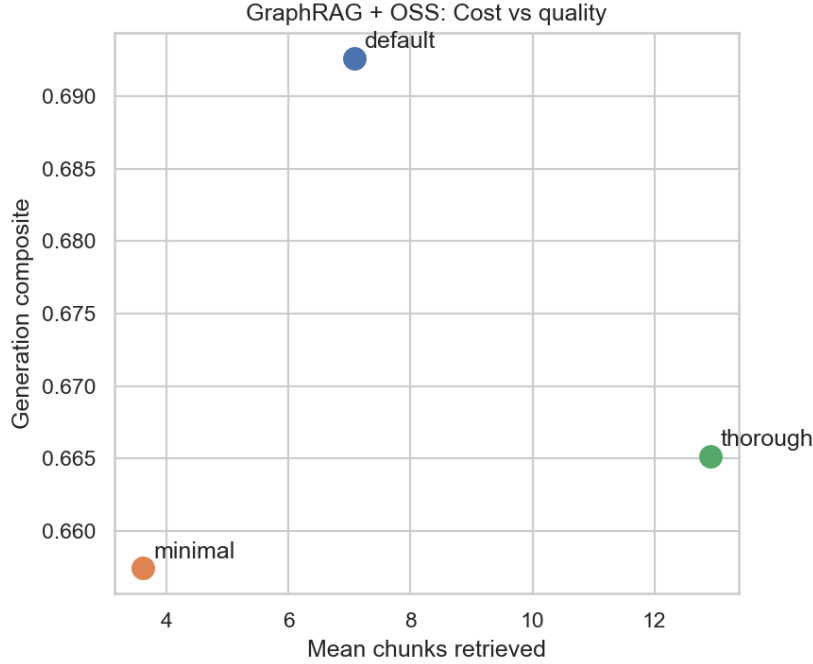


Figure 5.3: GraphRAG + OSS: retrieval cost (mean chunks) vs. generation composite by ablation.

≈ 80 – 82 answer wins vs. ≤ 1 for competitors (Table 5.3, Figure 5.5a). Under Mistral, GraphRAG requires deep graph traversal; shallow ablations fail on both automated metrics and holistic judging (Figure 5.4).

5.1.4 VectorRAG + OSS $\rightarrow$ **vector_minimal**

Vector + OSS is the most contested cell. Retrieval composites are nearly tied across ablations (span < 0.004 ; Table 5.2, Figure 5.6): **minimal** (0.834), **default** (0.835), **thorough** (0.831). **thorough** leads generation composite (0.733) and Holm-significant wins (3), and the order-averaged judge prefers **thorough** in every pairwise comparison (Table 5.3, Figure 5.7a).

We select **vector_minimal** with **medium confidence** because, at essentially equal retrieval quality, it uses ≈ 4.2 mean chunks vs. $\approx 10+$ for other ablations ($\approx 60\%$ retrieval-cost reduction; Figure 5.8). Forward-order judge strongly preferred **minimal** over **default** (61–16 answers). **vector_thorough** remains the alternative if generation composite is prioritised over efficiency.

5.1.5 VectorRAG + Mistral $\rightarrow$ **vector_thorough**

vector_thorough leads Holm-significant metric wins (3 vs. 2) and wins all pairwise judge comparisons (≈ 59 – 65 vs. 17 – 23 answer wins; Table 5.3, Figure 5.10).

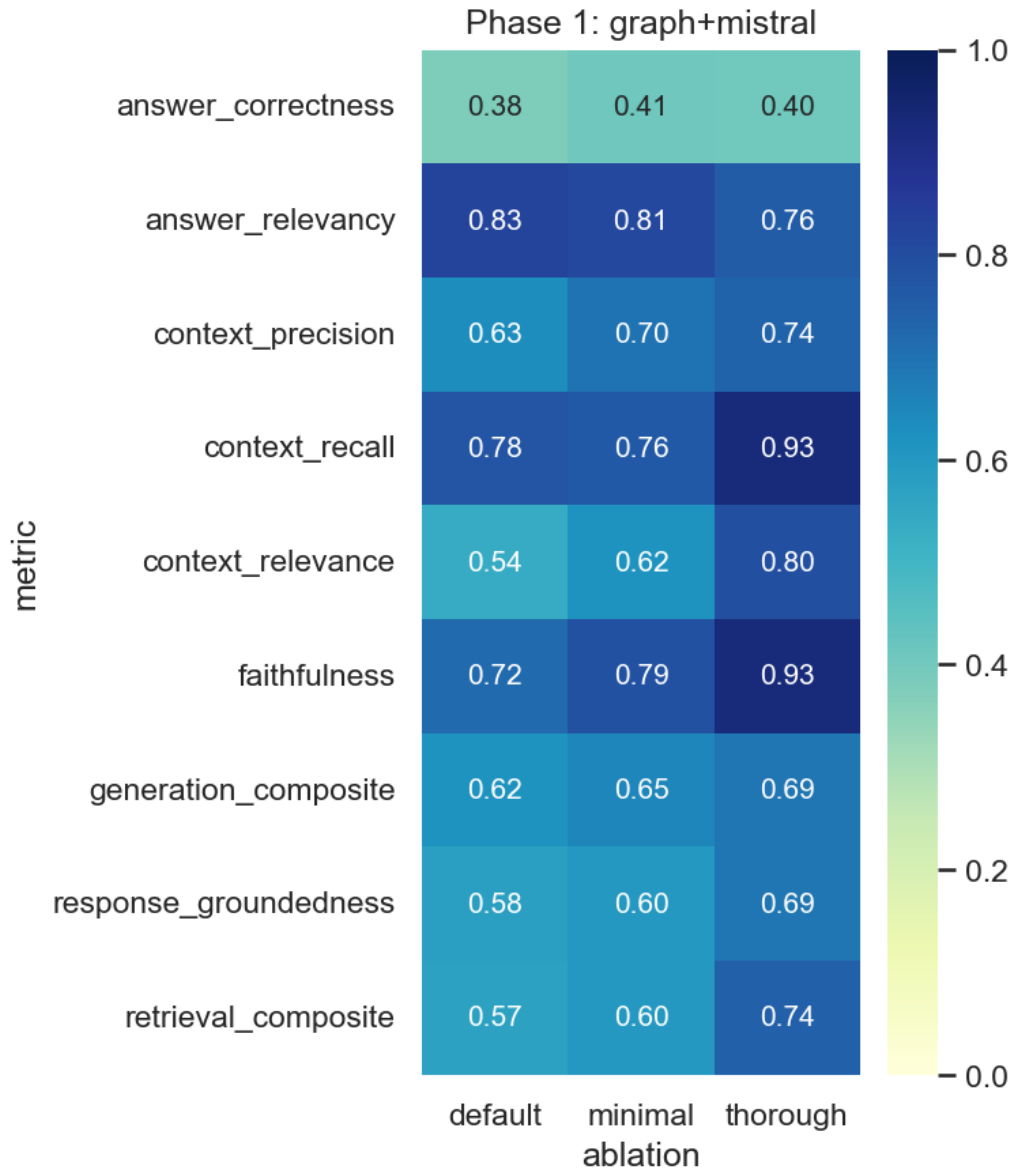
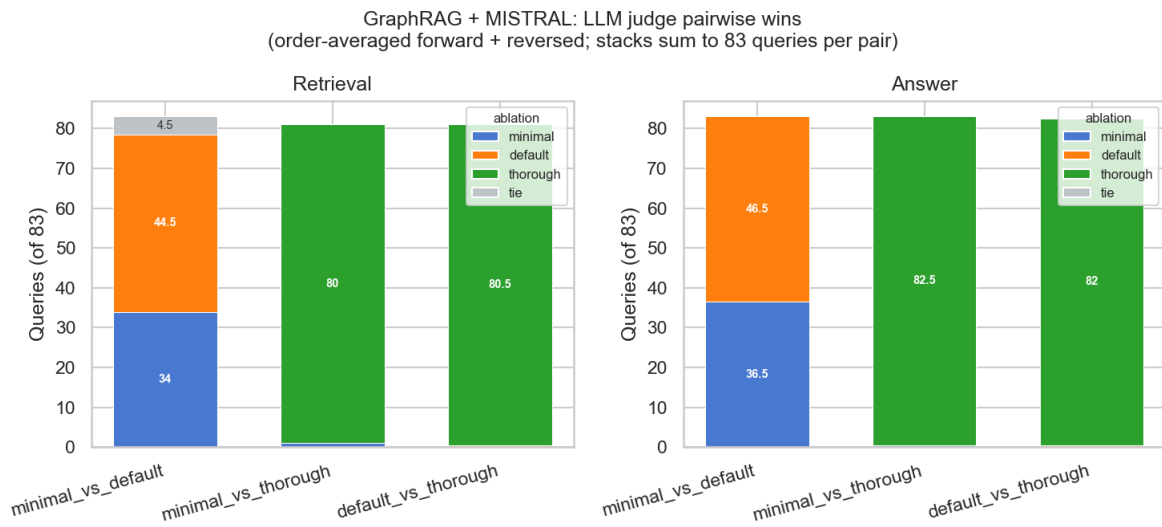
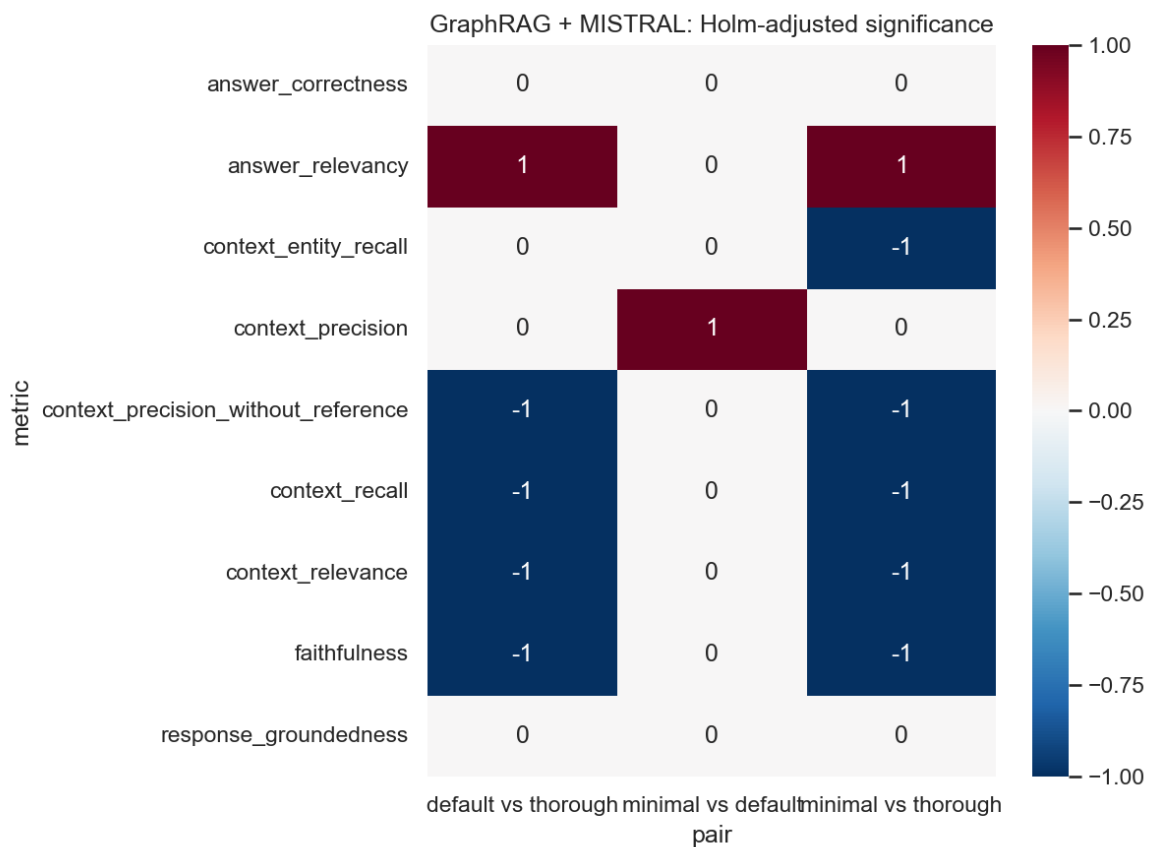


Figure 5.4: GraphRAG + Mistral: Phase 1 per-metric means across ablations.



(a) Order-averaged judge wins



(b) Holm-significant metric wins

Figure 5.5: GraphRAG + Mistral: unanimous evidence for `graph_thorough`.

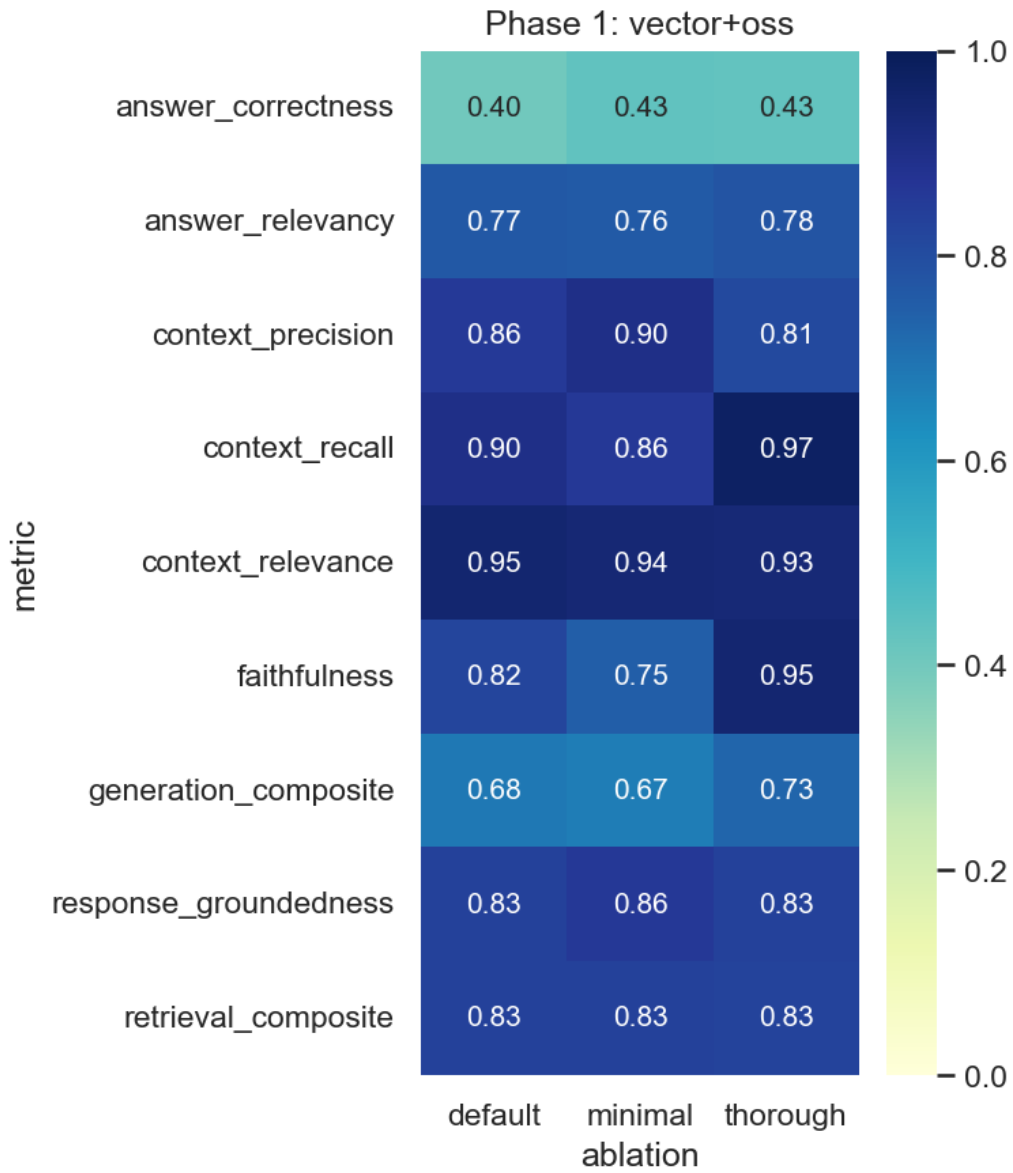
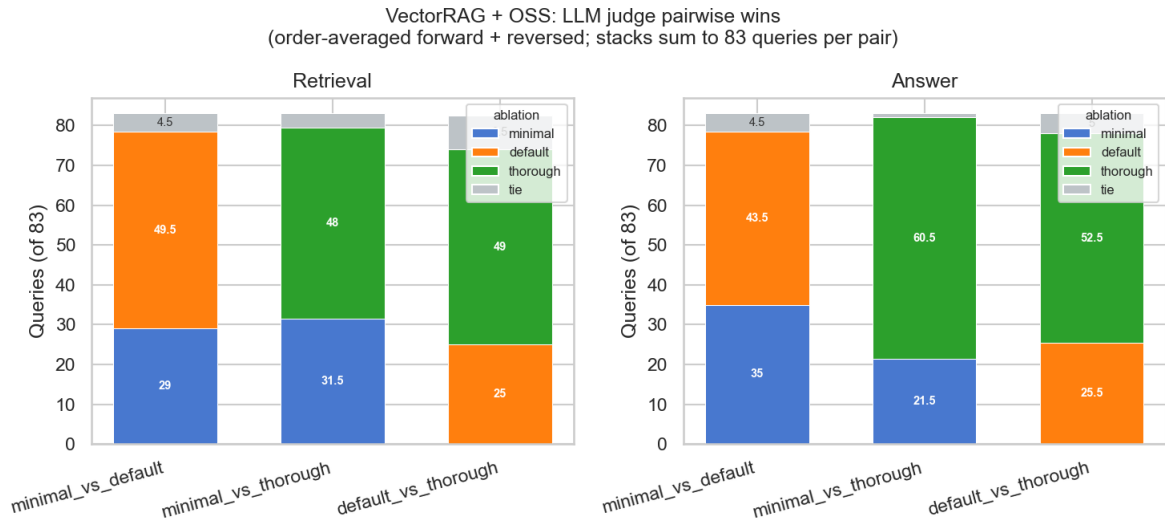
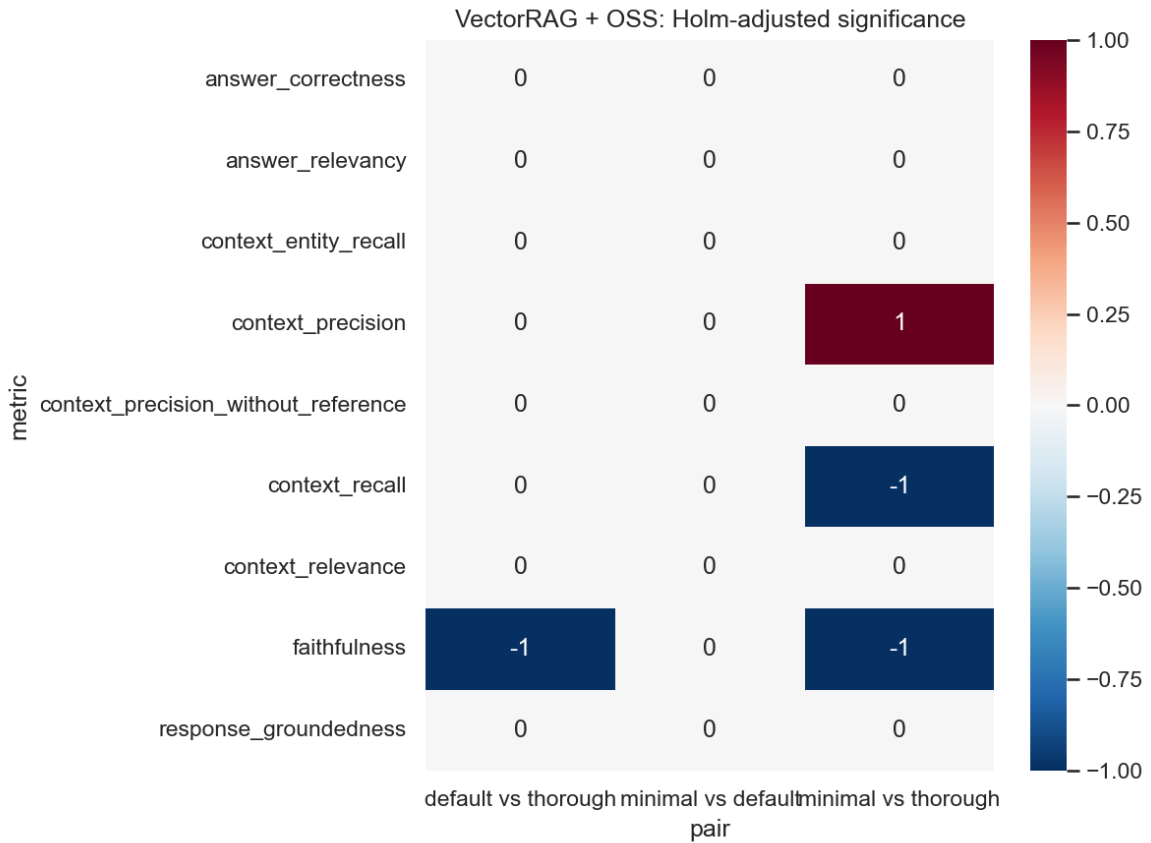


Figure 5.6: VectorRAG + OSS: Phase 1 per-metric means across ablations (retrieval composites differ by < 0.004).



(a) Order-averaged judge wins



(b) Holm-significant metric wins

Figure 5.7: VectorRAG + OSS: order-averaged LLM judge outcomes and Holm-adjusted significance results for pairwise ablation comparisons. The judge generally favours **thorough**, especially for answer quality. The statistical results show that **thorough** significantly improves context recall and faithfulness, while **minimal** significantly improves context precision. Most other metric differences are not statistically significant.

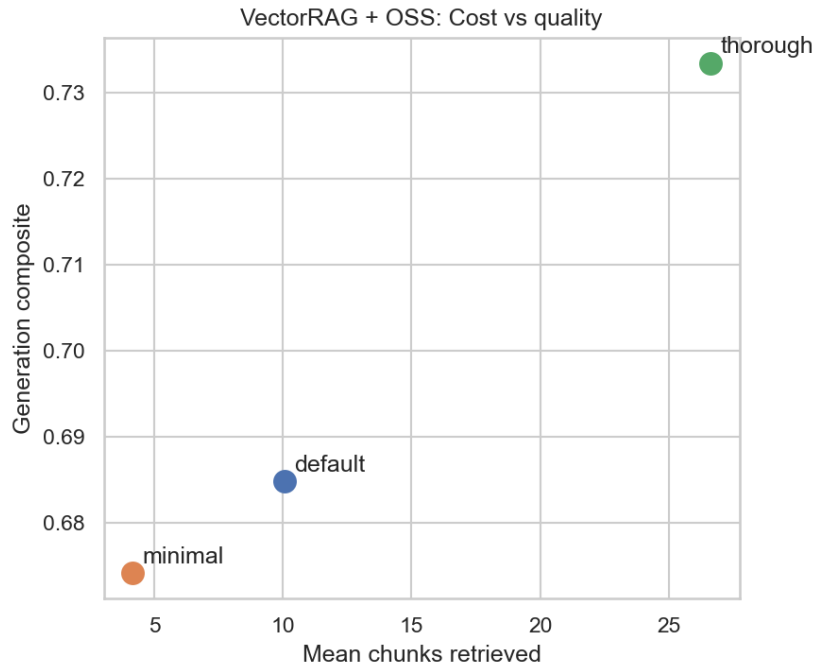


Figure 5.8: VectorRAG + OSS: retrieval cost vs. generation composite (minimal offers lowest cost at near-peak generation).

`vector_default` has a slight generation-composite edge (0.750 vs. 0.732), visible in Figure 5.9, but judge and statistics favour `thorough` decisively (Figure 5.11). The cost is high: `vector_thorough` retrieves ≈ 20.1 chunks on average—the highest of all twelve configurations (Figure 5.12).

5.1.6 Phase 1 takeaway

Phase 1 finalists fed into Phase 2 are: `graph_default` vs. `vector_minimal` (OSS) and `graph_thorough` vs. `vector_thorough` (Mistral). Two patterns emerge from the figures above. First, **generation model matters**: Mistral GraphRAG requires `thorough` retrieval, whereas OSS GraphRAG peaks at `default`. Second, **evidence lenses can diverge**—most sharply for Vector + OSS, where judge and generation composite favour `thorough` but retrieval efficiency favours `minimal`. These disagreements foreshadow the Phase 2 finding that retrieval metrics and answer quality need not align.

5.2 Phase 2: Finalist bout

Phase 2 compares `graph_default` vs. `vector_minimal` (OSS lane) and `graph_thorough` vs. `vector_thorough` (Mistral lane) on all 83 bootstrapped queries.

Table 5.4 summarises headline automated scores; Table 5.5 and Figure 5.13 show how evidence lenses diverge.

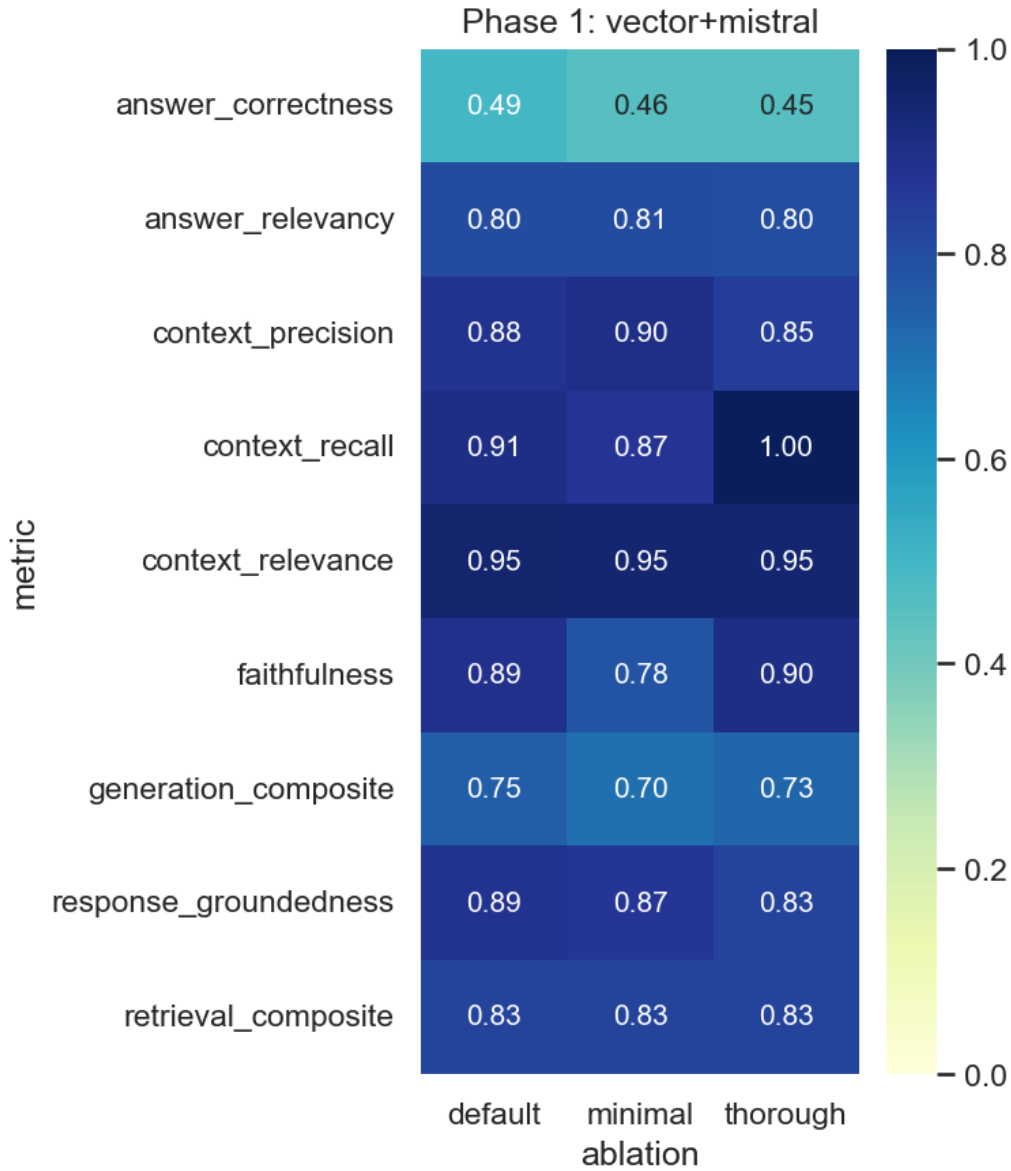


Figure 5.9: VectorRAG + Mistral: Phase 1 per-metric means across ablations.

Table 5.4: Phase 2 finalist automated metrics (selected composites). Bold indicates higher value within each row.

Bout	Metric	GraphRAG	VectorRAG
OSS	Retrieval composite	0.755	0.834
	Generation composite	0.693	0.674
	Faithfulness	0.886	0.749
	Answer relevancy	0.754	0.759
Mistral	Retrieval composite	0.745	0.828
	Generation composite	0.692	0.732
	Faithfulness	0.928	0.904
	Answer relevancy	0.757	0.800

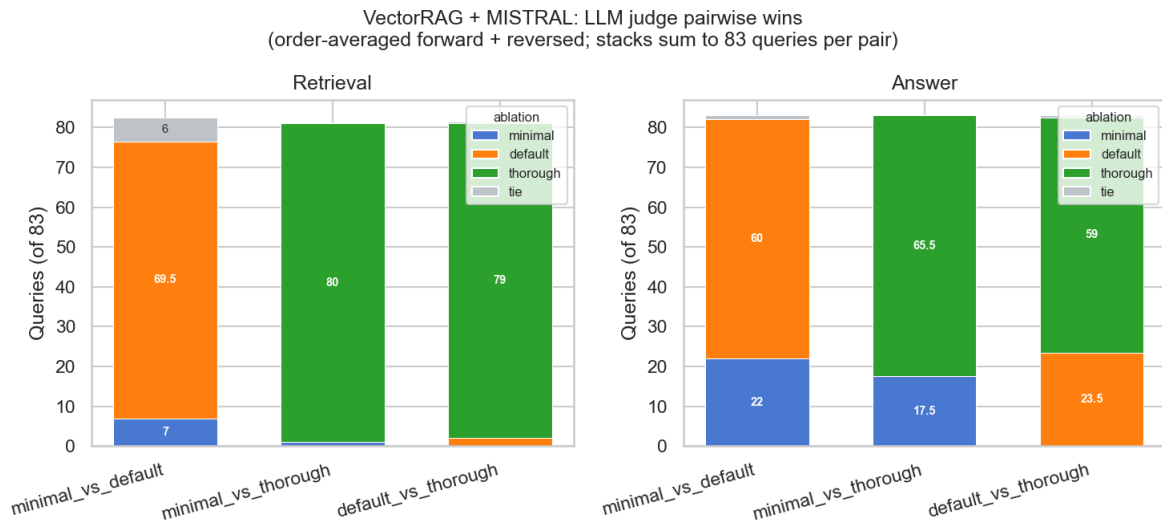


Figure 5.10: VectorRAG + Mistral: order-averaged judge wins.

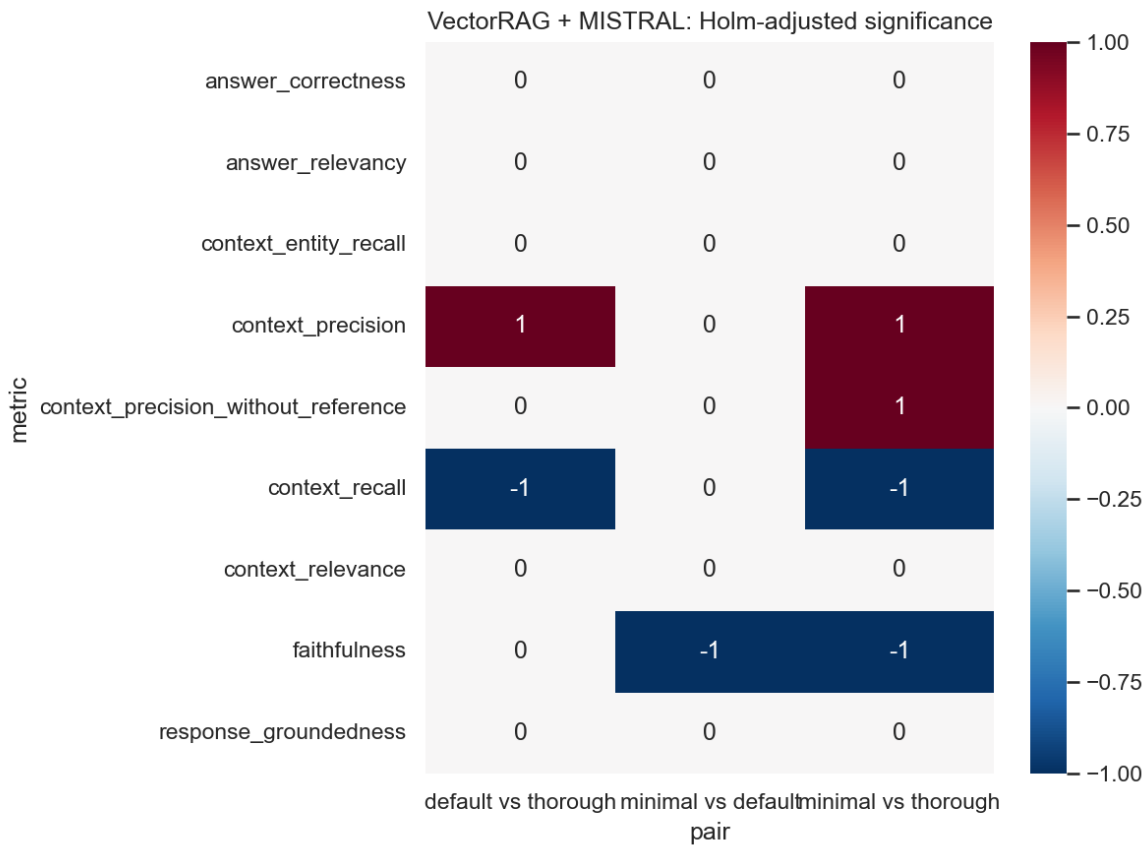


Figure 5.11: VectorRAG + Mistral: Holm-significant metric wins.

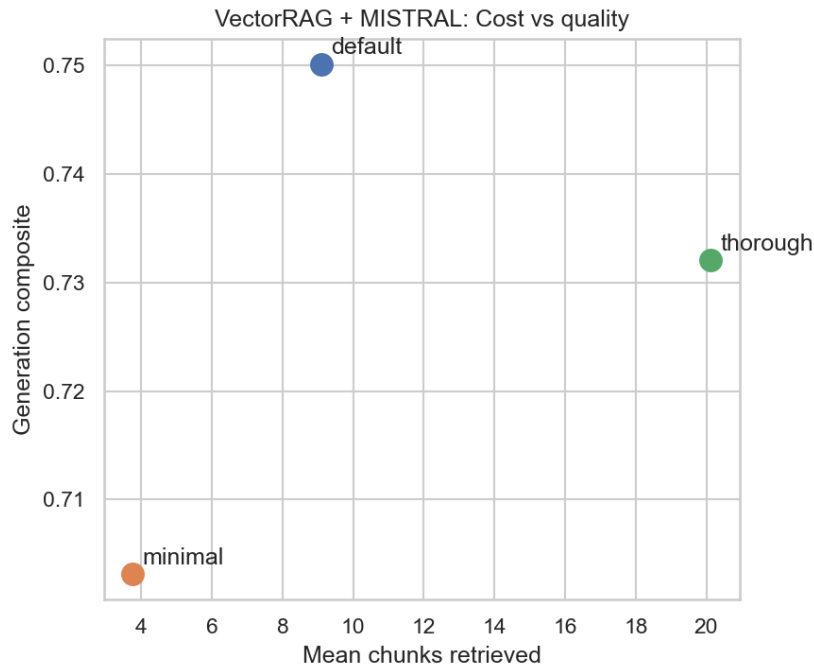


Figure 5.12: VectorRAG + Mistral: **thorough** achieves best judge/statistical profile at highest retrieval cost.

Table 5.5: Triangulation across automated metrics, Holm-significant statistics, and LLM judge (answer wins).

Bout	Evidence lens	Graph (%)	Vector (%)	Graph wins	Vector wins
OSS Phase 2	Automated metrics	33.3	44.4	3	4
Mistral Phase 2	Automated metrics	11.1	88.9	1	8
OSS Phase 2	Statistics (Holm sig.)	40.0	60.0	2	3
Mistral Phase 2	Statistics (Holm sig.)	0.0	100.0	0	4
OSS Phase 2	LLM judge (answers)	63.3	36.7	52.5	30.5
Mistral Phase 2	LLM judge (answers)	96.4	3.6	80	3

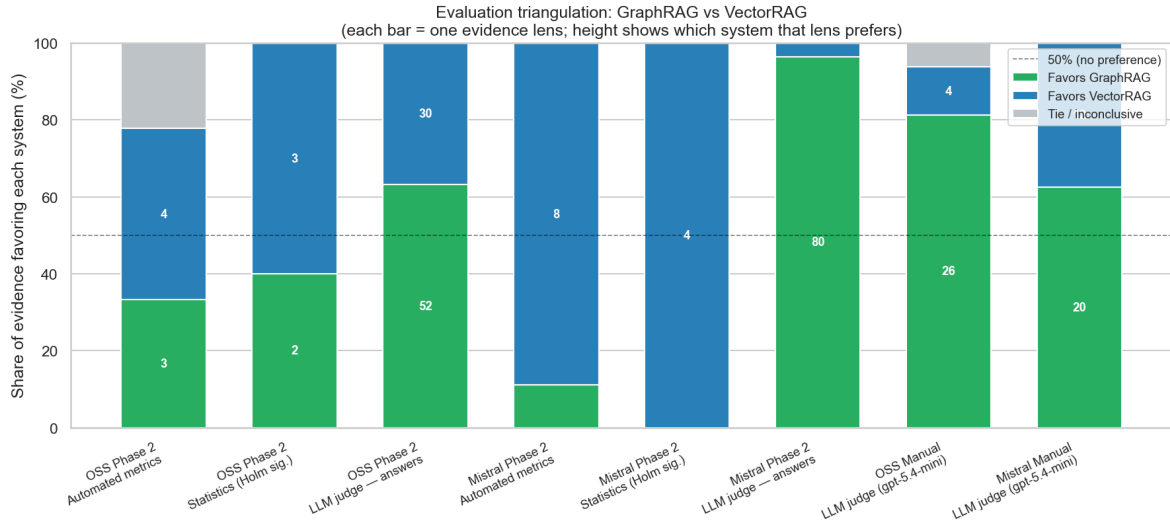


Figure 5.13: Stacked-bar summary of Graph vs Vector win share by evidence lens and bout.

On automated metrics and statistics, **VectorRAG leads consistently**, especially for retrieval composites and context precision/recall. Conversely, the LLM judge prefers **GraphRAG answers** in both model lanes: 63% vs. 37% (OSS, order-averaged) and 96% vs. 4% (Mistral).

Figures 5.14a and 5.14b break down judge outcomes by query type where available.

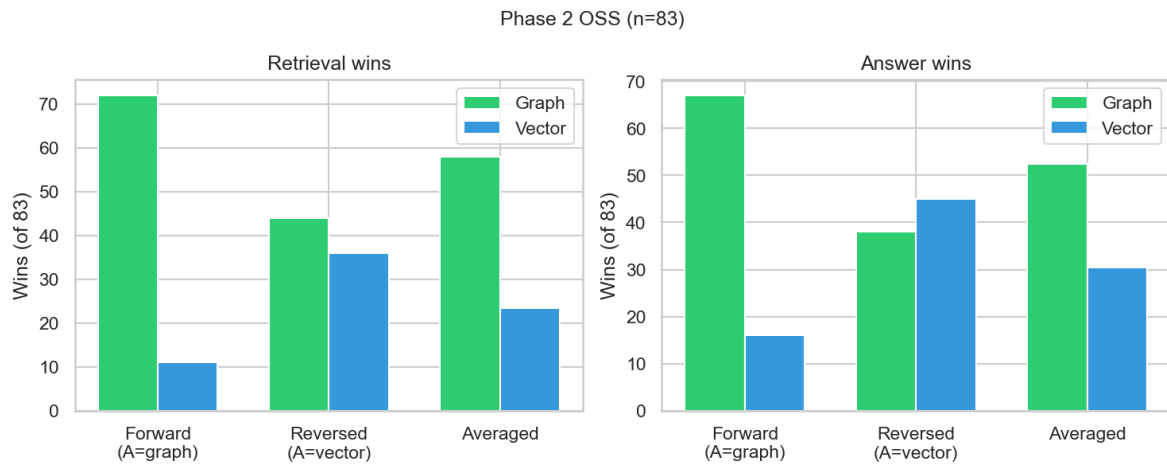
Interpretation. Retrieval-oriented proxies favour VectorRAG, yet downstream answer quality favours GraphRAG under holistic judging. This dissociation motivates Phase 3 manual validation on queries without bootstrapped references.

5.3 Phase 3: Manual validation

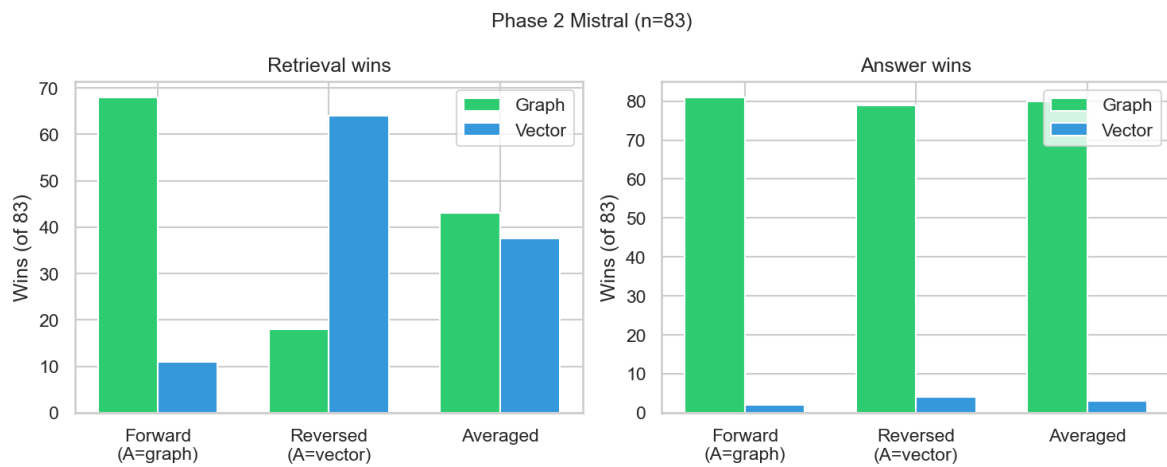
Phase 3 re-runs the Phase 2 finalist pairs on 16 manually curated queries. Each query is judged twice (Graph-first and Vector-first); we report the total answer wins for both orders combined.

Table 5.6: Cross-phase verdict: LLM judge answer wins (Graph vs Vector).

Phase	Comparison	Winner	Graph ans. wins	Vector ans. wins
2	OSS (83 queries)	GraphRAG	52.5	30.5
2	Mistral (83 queries)	GraphRAG	80	3
3	OSS manual (16 queries)	GraphRAG	26	4
3	Mistral manual (16 queries)	GraphRAG	20	12

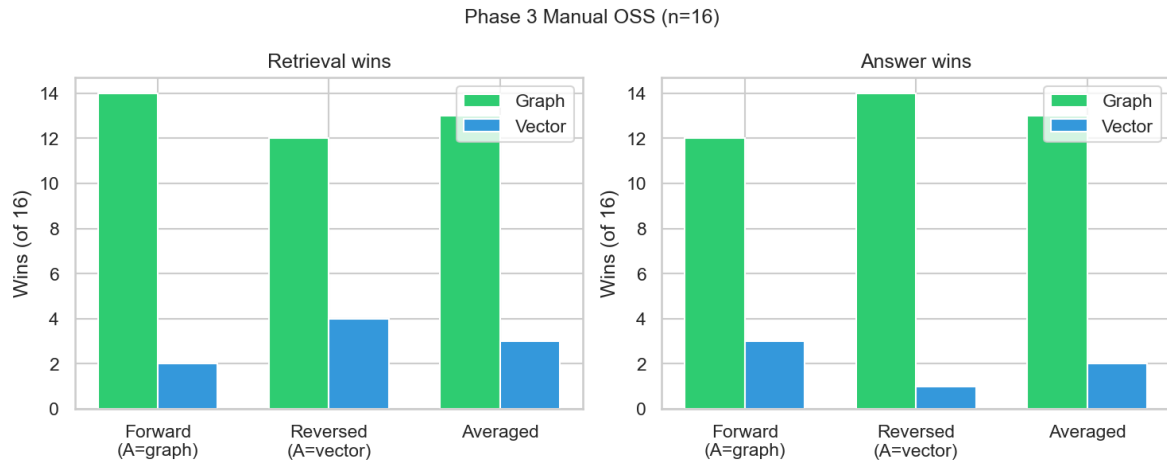


(a) OSS lane

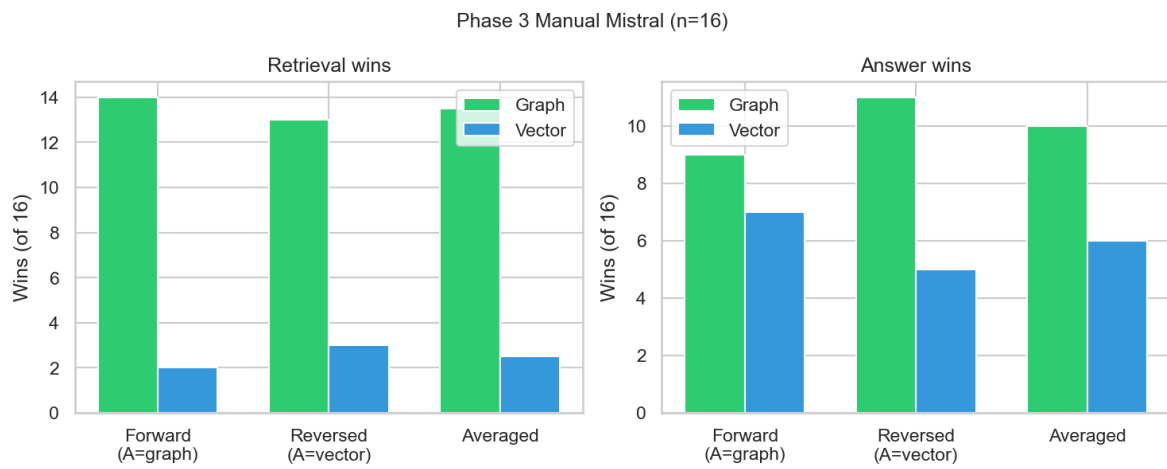


(b) Mistral lane

Figure 5.14: Phase 2 order-averaged LLM judge wins (Graph vs Vector finalists).



(a) OSS manual set



(b) Mistral manual set

Figure 5.15: Phase 3 manual validation: per-query order-averaged judge outcomes.

Chapter 6

Discussion

6.1 Main Interpretation of Results

The central finding of this thesis is that retrieval-oriented metrics and answer-level quality do not necessarily lead to the same system preference. Across the final evaluation phases, VectorRAG achieved stronger automated retrieval scores, particularly on context precision, context recall, and retrieval composite metrics. However, GraphRAG was preferred in pairwise answer-level comparisons and was further supported by the manually curated validation set.

This result suggests that chunk-level retrieval quality is not always a reliable proxy for downstream question-answering quality. VectorRAG is effective at retrieving chunks that are semantically or lexically similar to the query and aligned with predefined reference contexts. This explains its strong performance on automated retrieval metrics. However, GraphRAG retrieves evidence through a structured traversal process, moving from communities to papers, sections, and connected document elements. This can produce more coherent and task-relevant context, even when the retrieved chunks do not overlap as strongly with the reference evidence used by automated metrics.

The results therefore do not show that GraphRAG is universally superior to VectorRAG. Instead, they show that the two systems optimize for different retrieval behaviours. VectorRAG is stronger according to metrics that reward direct chunk-level overlap and ranking quality, while GraphRAG appears stronger when the final answer is evaluated holistically for usefulness, grounding, and completeness. This distinction is important because the intended task is not retrieval in isolation, but document-grounded question answering. A likely explanation for this divergence is that the two evaluation views reward different notions of retrieval success. The automated retrieval metrics primarily reward overlap with predefined reference evidence and penalize retrieved context that does not match the expected chunks or entities. This benefits VectorRAG, since its retrieval process is explicitly optimized for chunk-level semantic and lexical matching. GraphRAG, however, retrieves evidence through a structured path over communities, papers, sections, and connected document elements. As a result, it may retrieve different evidence than the reference context while still collecting information that is coherent, complementary, and sufficient for answering the query.

This distinction is especially important for multi-hop and comparative questions. In these cases, answer quality depends not only on whether individual chunks are relevant, but also on whether the retrieved evidence forms a useful explanatory context. GraphRAG can provide this by preserving document hierarchy and cross-document relations during retrieval. The judge evaluations may therefore prefer GraphRAG because its answers are better synthesized from structurally related evidence, even when the retrieved chunks achieve lower reference-overlap scores. In contrast, VectorRAG may

retrieve highly relevant isolated chunks but provide less coherent support for synthesis across sources.

6.2 GraphRAG Compared with VectorRAG

The comparison between GraphRAG and VectorRAG highlights a trade-off between retrieval precision, structural context, and answer quality. VectorRAG benefits from mature and efficient retrieval mechanisms: dense vector search, BM25, reciprocal rank fusion, reranking, and iterative query refinement. These components make it a strong baseline, especially for queries where the required evidence is located in a small number of clearly relevant chunks.

GraphRAG, by contrast, relies on explicit document structure. Its retrieval process is less dependent on corpus-wide similarity search and instead uses graph traversal to collect evidence from connected papers, sections, figures, tables, and communities. This makes retrieval more interpretable and can support synthesis across multiple sources. The graph structure also gives the system a stronger notion of document hierarchy and cross-document relationships, which is useful for multi-hop and comparative questions.

The evaluation suggests that this structural advantage is most visible at the answer level. GraphRAG does not always retrieve the same chunks as the reference context, and this can reduce its automated retrieval scores. However, the retrieved evidence may still be sufficient, coherent, and useful for answering the query. This explains why answer-level judges can prefer GraphRAG even when retrieval metrics favour VectorRAG.

This should not be interpreted as a failure of the automated metrics, but rather as a limitation of what they measure. Retrieval metrics are valuable for assessing whether a system finds expected evidence, but they are less able to evaluate whether the retrieved evidence is organized in a way that supports reasoning and synthesis. GraphRAG's advantage appears to come from context construction rather than simple evidence matching: it retrieves information along meaningful document and citation structures, which can make the final context easier for the generator to use.

The preferred GraphRAG configuration also depended on the generation model. For the OSS model lane, `graph_default` provided the best trade-off between retrieval depth and answer quality. For the Mistral lane, `graph_thorough` was more effective, suggesting that some model and retrieval configurations require deeper evidence collection to produce strong answers. This indicates that retrieval depth should not be treated as a fixed universal parameter. Instead, it should be tuned jointly with the generation model and the expected query complexity.

For cost-sensitive OSS deployments, `graph_default` provides a balanced configuration, since it achieves strong answer quality while retrieving fewer chunks than the

Table 6.1: Recommended GraphRAG configuration by generation model.

Model lane	System	Configuration	Confidence
OSS	GraphRAG	<code>graph_default</code>	High
Mistral	GraphRAG	<code>graph_thorough</code>	High

thorough configuration. For Mistral, `graph_thorough` is justified by stronger agreement across the evaluation phases, although this comes at higher retrieval cost. The practical recommendation is therefore not simply to maximize retrieval depth, but to select retrieval configuration based on the trade-off between answer quality, latency, token usage, and model behaviour.

6.3 Evaluation Design and Threats to Validity

A major finding of this thesis is that evaluation design strongly affects the conclusions drawn from RAG experiments. The initial RAGAS-generated dataset was useful for scalable benchmarking, but its chunk-level construction introduced a risk of favouring systems that retrieve evidence at the same granularity as the dataset generation process. This was one reason why the final comparison relied primarily on the bootstrapped dataset and used the manually curated dataset as an additional validation set.

The use of multiple datasets was therefore important. The RAGAS-generated dataset enabled large-scale automated evaluation, the bootstrapped dataset reduced dependence on chunk-level construction, and the manually curated dataset provided a smaller but more carefully verified validation signal. These datasets represent different trade-offs between scale, control, and human involvement. Together, they reduce the risk that the conclusions are caused by artifacts of a single dataset construction method.

The disagreement between automated metrics and LLM-as-judge comparisons also highlights the difficulty of evaluating RAG systems. Reference-based metrics such as context recall, context precision, entity recall, and answer correctness measure alignment with predefined reference evidence. These metrics are valuable for controlled benchmarking and regression testing, but they may penalize systems that retrieve alternative evidence that is still sufficient to answer the query.

Reference-free metrics and LLM-as-judge comparisons provide a more holistic signal, since they evaluate the relationship between the query, retrieved context, and generated response. However, they are also limited. LLM judges depend on evaluator model choice, prompt design, presentation order, and model bias. The dual-order evaluation protocol mitigates position bias, but does not eliminate all evaluator uncertainty. The manually curated validation set strengthens the evaluation, but its small size means that it should be interpreted as supporting evidence rather than definitive proof.

The statistical analysis is similarly constrained by the number of evaluated queries and the cost of repeated evaluation. Paired non-parametric tests are appropriate because both systems are evaluated on the same queries, but the study does not include a large number of independent repeated runs. As a result, the analysis captures variation across queries more clearly than variation across repeated executions of the same query. This is a practical limitation of evaluating agentic RAG systems, where each run can involve multiple retrieval steps, tool calls, model calls, and judge calls.

Overall, the evaluation results should be interpreted as evidence under the specific experimental conditions of this thesis. They support the conclusion that GraphRAG can improve judged answer quality in this setting, but they do not establish a universal ranking between GraphRAG and VectorRAG.

6.4 Corpus and Graph Design Implications

Scientific literature was selected as the evaluation corpus because it provides a reproducible and structurally rich proxy for technical internal documentation. Research papers contain hierarchical sections, citations, dense terminology, figures, tables, equations, and cross-document relationships. These properties make them suitable for evaluating both vector-based and graph-based retrieval.

At the same time, the corpus is not identical to the confidential internal documentation that motivated the project. Internal documents may contain different structures, naming conventions, access restrictions, update patterns, and implicit organizational context. The results should therefore be interpreted as evidence from a controlled technical corpus rather than direct proof of performance in an industrial deployment.

Corpus scale is another important limitation. The evaluation corpus consists of 103 papers and is relatively thematically concentrated. Several of the largest domains, including RAG, knowledge graphs, agentic systems, and world models, overlap substantially. This limits the extent to which the study can evaluate how graph-based retrieval behaves under larger and more heterogeneous collections.

The Leiden clustering analysis also showed that the corpus had relatively homogeneous community structure. Although several communities were identified, the overlap between research areas reduced the separation between clusters. This may have limited the potential advantage of community-aware retrieval. In a larger and more diverse corpus, community structure, citation networks, and cross-document relations may become more informative. Under such conditions, GraphRAG may gain more from explicit graph structure. However, this hypothesis was not directly tested and should be investigated in future work.

The graph construction process itself also introduces design trade-offs. The schema

captures paper hierarchy, citations, section order, figures, tables, equations, semantic similarity, and communities. This provides a useful structure for traversal, but it is still an approximation of the underlying knowledge. More fine-grained graph representations could include entities, claims, methods, results, contradictions, and task-specific relations. Such extensions may improve retrieval quality, but would also increase construction complexity and introduce new sources of extraction error.

6.5 System Limitations

The implemented systems should be understood as research prototypes rather than production-ready systems. Several limitations affect the interpretation and practical applicability of the results.

First, GraphRAG depends strongly on graph quality. Errors introduced during parsing, chunking, metadata extraction, citation resolution, figure processing, table summarization, or community detection can directly affect traversal behaviour. Although the ingestion pipeline combines GROBID, Docling, and VLM-based edge-case handling, no parser is fully reliable across all scientific PDFs. Citation resolution based on title normalization and fuzzy matching may miss valid links or introduce incorrect ones, and similarity-based community construction depends on embedding quality and chosen thresholds.

Second, both systems depend on agentic control logic. The GraphRAG agent sometimes repeated tool calls, stopped after retrieving from too few papers, selected structurally plausible but weakly relevant sections, or exhausted its hop budget before collecting sufficient evidence. These behaviours were mitigated through duplicate-call detection, community-search guards, finish guards, coverage checks, and hop budgets. However, these controls do not guarantee optimal traversal. Similarly, the VectorRAG pipeline depends on query rewriting, judging, and reasoning agents, which can misdiagnose missing evidence or confuse topical similarity with evidential sufficiency.

Third, token usage and context management remain limited. The system does not include learned trajectory compression, systematic context pruning, or an independent verification controller. In the most severe observed cases, the thorough GraphRAG ablation retrieved approximately 27,000 tokens. This remained within the context limits of the evaluated models, but it would not be sufficient as a production-level strategy. Future systems would require stronger mechanisms for evidence pruning, summarization, adaptive stopping, and confidence calibration.

Fourth, multimodal and equation handling remain incomplete. Figures and malformed tables were processed using a vision-language model and converted into natural-language descriptions or summaries. This allowed visual and tabular content to par-

ticipate in retrieval, but the generated descriptions were not systematically validated by domain experts. Equations were preserved as graph nodes with provenance, but their mathematical content was not fully extracted or used as retrievable evidence. The system is therefore less suitable for questions requiring precise mathematical reasoning or symbolic comparison.

Finally, the results are model-specific. The evaluated behaviour depends on the selected generation models, evaluator models, prompts, context limits, and retrieval configurations. The fact that the best GraphRAG configuration differed between the OSS and Mistral lanes indicates that retrieval configuration and model capability interact. Future work should therefore evaluate a broader range of models and retrieval settings before drawing stronger conclusions about general performance.

6.6 Industrial Relevance

The system developed in this thesis is relevant for internal document search in high-assurance and restricted environments, where cloud-based language-model services may be unsuitable due to confidentiality, security, and operational-control requirements. In such settings, organizations need systems that can operate within controlled infrastructure while still providing useful and grounded answers over domain-specific documents.

The proposed GraphRAG architecture supports this by separating the language model from the knowledge source. Documents, metadata, sections, citations, figures, tables, and semantic communities are stored in an explicit knowledge graph, allowing the knowledge base to be inspected, updated, and controlled without retraining the model. This is important for industrial document collections that change over time, such as technical reports, manuals, requirements, and project documentation.

The graph-based approach also improves traceability. Rather than retrieving only isolated text chunks through similarity search, the system can navigate through explicit relationships between communities, papers, sections, and supporting evidence. This makes retrieval behaviour easier to inspect and supports source verification, which is particularly important in environments where answers must be auditable and grounded.

The results also highlight an important practical lesson: retrieval metrics alone may not be sufficient for evaluating industrial RAG systems. Although VectorRAG achieved stronger automated retrieval scores, GraphRAG was preferred in answer-level comparisons. This suggests that industrial evaluation should consider not only retrieval overlap, but also answer usefulness, grounding, completeness, and traceability.

Overall, the thesis indicates that agentic GraphRAG is a promising direction for internal document question answering in controlled environments. However, the imple-

mented system remains a research prototype. Before deployment, further work would be needed on access control, monitoring, confidence calibration, graph maintenance, human validation, and production robustness.

Chapter 7

Conclusion

7.1 Summary

This thesis has presented the design, implementation, and evaluation of an agentic GraphRAG system for document-grounded question answering over technical document collections. The work was motivated by the need for retrieval-augmented language-model systems that can operate in controlled or high-assurance environments, where traceability, local control, and grounded answers are important requirements. Scientific literature was used as a reproducible proxy for internal technical documentation, since it contains hierarchical structure, dense terminology, citations, figures, tables, and cross-document relations.

The implemented system combines document ingestion, metadata extraction, section-aware chunking, knowledge graph construction, vector indexing, graph traversal, reranking, agentic retrieval, and answer generation. Two retrieval architectures were compared under shared conditions: a VectorRAG baseline based on iterative hybrid search, and a GraphRAG system based on community-first hierarchical traversal over an explicit knowledge graph. Both systems used the same corpus, ingestion pipeline, database infrastructure, generation models, and evaluation framework, allowing the retrieval strategy to be treated as the main experimental variable.

The evaluation showed a clear divergence between retrieval-oriented metrics and answer-level evaluation. VectorRAG achieved stronger automated retrieval scores, particularly on context precision, context recall, and retrieval composite metrics. However, GraphRAG was preferred in pairwise LLM-as-judge comparisons and was further supported by the manually curated validation set. This suggests that chunk-level retrieval metrics do not always capture whether retrieved evidence is useful for downstream answer generation, especially when structurally different retrieval systems retrieve different but still valid evidence.

Overall, the results indicate that graph-based retrieval can improve judged answer

quality and retrieval traceability in agentic RAG systems, but also that such systems introduce additional complexity. GraphRAG performance depends on graph quality, traversal control, corpus structure, model behaviour, and evaluation design. The findings should therefore be interpreted as evidence under the evaluated experimental conditions rather than as a general claim that GraphRAG is universally superior to VectorRAG.

7.2 Answers to the Research Questions

The first research question asked how an agentic GraphRAG system can be designed to support document-grounded question answering over technical document collections. The thesis shows that such a system can be constructed by combining structured document ingestion, a document-level knowledge graph, semantic embeddings, community detection, and tool-based graph traversal. The resulting architecture allows an agent to move from broad topical communities to papers, sections, chunks, figures, and tables, while keeping retrieved evidence separate from navigational working memory. This design supports controlled evidence collection and makes retrieval decisions more inspectable than flat vector search.

The second research question asked how GraphRAG compares with VectorRAG in terms of retrieval quality, answer grounding, and overall question-answering performance. The results show that VectorRAG performs better according to several automated retrieval metrics, while GraphRAG performs better according to holistic answer-level judging. This indicates that VectorRAG is stronger at retrieving chunks that align with predefined reference contexts, whereas GraphRAG may retrieve more structurally coherent evidence that supports better final answers. The strongest conclusion is therefore not that one system dominates the other across all dimensions, but that the two retrieval paradigms optimize for different strengths.

The third research question concerned failure modes in multi-loop agentic retrieval systems. Several practical failure modes were observed. The graph agent sometimes repeated tool calls, stopped after retrieving evidence from too few papers, selected structurally plausible but weakly relevant sections, or exhausted its hop budget before collecting sufficient evidence. The vector-based agentic pipeline could also struggle when query rewriting or sufficiency judging failed to identify the missing evidence. These issues were mitigated through duplicate-call detection, finish guards, community-search guards, coverage checks, hop budgets, and a separation between working memory and final evidence. However, the results also show that agentic retrieval requires careful control logic and cannot be assumed to improve retrieval automatically.

The fourth research question asked how explicit graph traversal affects interpretability and traceability compared with vector-based retrieval. The graph-based system provides

stronger traceability because retrieval proceeds through explicit graph objects and relationships, such as communities, papers, sections, citations, figures, tables, and chunks. This makes it possible to inspect why a particular source was reached and how the agent navigated through the corpus. In contrast, VectorRAG primarily retrieves ranked chunks based on semantic and lexical similarity, which is effective but less explanatory. However, graph-based traceability depends on the correctness of the graph schema, parser outputs, citation links, and traversal logs. A poorly constructed graph would reduce both retrieval quality and interpretability.

The fifth research question asked what practical challenges arise when evaluating agentic GraphRAG and VectorRAG systems. The main challenge is that different evaluation methods reward different behaviours. Reference-based retrieval metrics can favour systems that retrieve chunks similar to the dataset construction context, while answer-level judges may reward completeness, synthesis, and usefulness even when retrieved evidence differs from the reference context. LLM-as-a-judge evaluation provides a valuable comparative signal, but remains dependent on evaluator choice, prompt design, presentation order, and model bias. The use of multiple datasets, statistical tests, dual-order judging, and manual validation helped reduce dependence on any single evaluation lens, but the evaluation remains constrained by dataset size, corpus choice, and the absence of expert human assessment.

7.3 Contributions

This thesis makes five main contributions.

First, it presents an end-to-end implementation of an agentic GraphRAG system for document-grounded question answering. The system integrates document parsing, metadata extraction, section-aware chunking, knowledge graph construction, semantic indexing, graph traversal, tool calling, context assembly, and answer generation.

Second, it provides a controlled comparison between GraphRAG and VectorRAG under shared experimental conditions. Both systems operate on the same corpus and use the same generation and evaluation pipeline, making it possible to isolate retrieval strategy as the primary difference between the systems.

Third, the thesis develops a multi-stage evaluation protocol for comparing agentic retrieval systems. The protocol combines automated retrieval and generation metrics, ablation studies, statistical significance testing, pairwise LLM-as-a-judge comparisons, and manual validation.

Fourth, the thesis identifies a central evaluation finding: automated retrieval metrics and answer-level quality assessments can lead to different conclusions. In the evaluated setting, VectorRAG performs better on several retrieval-oriented metrics, while

GraphRAG is preferred in holistic answer comparisons. This highlights the importance of evaluating RAG systems not only by retrieval overlap, but also by downstream answer quality.

Fifth, the thesis documents practical failure modes and control mechanisms for agentic retrieval. These include premature stopping, redundant tool calls, insufficient evidence coverage, context growth, and evaluator dependence. The implemented mitigations provide practical insight into how agentic GraphRAG systems can be made more reliable and inspectable.

7.4 Future Work

Several directions remain for future work. The most important next step is to evaluate the system on larger and more diverse corpora. The present corpus is relatively small and thematically concentrated, which limits conclusions about scalability. Larger corpora with clearer community structure, richer citation networks, and more heterogeneous document types would make it possible to evaluate whether graph-based retrieval becomes more advantageous as corpus complexity increases.

Future work should also include expert human evaluation. The manually curated dataset used in this thesis provides a useful validation signal, but it is small and does not replace domain-expert assessment. Expert evaluation would make it possible to judge factual correctness, completeness, and grounding more reliably, especially for technical questions where LLM judges may miss subtle errors.

Another important direction is improved graph construction. The current graph captures document hierarchy, citations, figures, tables, equations, semantic similarity, and communities. Future systems could extend this representation with finer-grained entities, claim-level relations, method-result links, contradiction relations, and more robust citation resolution. Such additions could make graph traversal more semantically meaningful and improve both retrieval quality and interpretability.

The agentic control logic should also be strengthened. Future systems should include better token management, adaptive stopping criteria, evidence pruning, trajectory summarization, and calibrated confidence estimation. These mechanisms are especially important for production settings, where uncontrolled tool use and context growth can increase cost, latency, and failure risk.

Another important direction for future work would be to compare system behaviour across different generation models, such as OSS and Mistral. In the current experimental design, such a comparison would not be fully fair, since the final answers are generated by models with different capabilities. As a result, differences in answer quality may reflect model strength rather than retrieval-system behaviour. A more controlled setup

would use the same language model for final answer generation across all retrieval conditions, while varying only the retrieval architecture or retrieval configuration. This would make it possible to isolate how different models interact with the retrieved context, and whether GraphRAG and VectorRAG benefit differently from stronger or weaker generators.

Finally, future work should investigate production hardening in controlled industrial environments. This includes access control, monitoring of graph updates over time, regression testing, observability, human-in-the-loop validation, and robustness to continuously changing document collections. The system developed in this thesis should therefore be understood as a research prototype that demonstrates feasibility and comparative behaviour, rather than as a complete production-ready solution.

Bibliography

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [2] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, “On the opportunities and risks of foundation models,” *arXiv preprint arXiv:2108.07258*, 2021.
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.
- [4] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, “From local to global: A graph rag approach to query-focused summarization,” *arXiv preprint arXiv:2404.16130*, 2024.
- [5] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, “Okapi at trec-3,” in *Text REtrieval Conference (TREC-3)*, 1995.
- [6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” *arXiv preprint arXiv:2312.10997*, 2023.
- [7] G. V. Cormack, C. L. A. Clarke, and S. Büttcher, “Reciprocal rank fusion outperforms condorcet and individual rank learning methods,” in *SIGIR*, 2009.
- [8] R. Nogueira and K. Cho, “Passage re-ranking with bert,” in *arXiv preprint arXiv:1901.04085*, 2019.
- [9] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, *et al.*, “Knowledge graphs,” *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–37, 2021.

- [10] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, C. Wang, Z. Liu, *et al.*, “Autogen: Enabling next-gen llm applications via multi-agent conversation,” *arXiv preprint arXiv:2308.08155*, 2023.
- [11] L. Weng, “Llm powered autonomous agents,” *Lil’Log*, 2023.
- [12] S. Yao *et al.*, “React: Synergizing reasoning and acting in language models,” *ICLR*, 2023.
- [13] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, “Ragas: Automated evaluation of retrieval augmented generation,” *EACL 2024*, 2024.
- [14] L. Zheng *et al.*, “Judging LLM-as-a-judge with MT-Bench and chatbot arena,” 2023.
- [15] X. Wang *et al.*, “Large language models are not fair evaluators,” *arXiv preprint arXiv:2305.17926*, 2024.
- [16] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
- [17] S. Holm, “A simple sequentially rejective multiple test procedure,” *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.
- [18] H. Han, L. Ma, Y. Wang, H. Shomer, Y. Lei, Z. Qi, K. Guo, Z. Hua, B. Long, H. Liu, C. C. Aggarwal, and J. Tang, “Rag vs. graphrag: A systematic evaluation and key insights,” *arXiv preprint arXiv:2502.11371*, 2025.
- [19] RAGAS, “Ragas documentation,” 2026. Accessed: 2026-05-26.
- [20] M. Günther, I. Mohr, D. J. Williams, B. Wang, and H. Xiao, “Late chunking: Contextual chunk embeddings using long-context embedding models,” *arXiv preprint arXiv:2409.04701*, 2024.
- [21] V. A. Traag, L. Waltman, and N. J. van Eck, “From louvain to leiden: guaranteeing well-connected communities,” *Scientific Reports*, vol. 9, no. 1, p. 5233, 2019.
- [22] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is “nearest neighbor” meaningful?,” in *Database Theory—ICDT’99*, pp. 217–235, Springer, 1999.
- [23] MongoDB, “Better rag results with reciprocal rank fusion (rrf) and hybrid search,” 2025.
- [24] Weaviate, “Hybrid search explained,” 2025.

Chapter A

Software Configuration and Frameworks

A.1 Document Processing

A.1.1 GROBID

GeneRation Of Bibliographic Data (GROBID) was used for bibliographic metadata extraction, including titles, authors, abstracts, DOI identifiers, and references. The system used the `processHeaderDocument` and `processReferences` endpoints with external consolidation disabled.

Official documentation: <https://grobid.readthedocs.io/>

A.1.2 Docling

Docling was used for layout-aware extraction of body text, section headers, tables, figures, and equations from PDF documents.

Official documentation: <https://docling-project.github.io/docling/>

A.2 Graph Database

A.2.1 Neo4j

Neo4j was used to store the knowledge graph and provide vector indexes for semantic retrieval. Neo4j also provided the graph data science library for clustering algorithms.

Official documentation: <https://neo4j.com/docs/>

A.3 Retrieval and Evaluation

A.3.1 LangChain

LangChain was used for agent orchestration, prompt management, and tool calling.

Official documentation: <https://python.langchain.com/>

A.3.2 LangFuse

Langfuse was used for observability and tracing of agentic LLM workflows. Recording tool calls, prompts, model responses, retrieval traces, and latency information, enabling post-hoc inspection and debugging of retrieval behaviour.

Official documentation: <https://langfuse.com/docs>

A.3.3 RAGAS

RAGAS provides metrics for retrieval quality and answer generation quality, including both reference-based and reference-free measures. Additionally RAGAS provides methods for generating evaluation dataset from a pre-defined corpora.

Official documentation: <https://docs.ragas.io/>

A.4 Implementation helpers

A.4.1 Docker

is used to deploy infrastructure components in isolated and reproducible environments. Containerization reduces dependency conflicts and supports consistent deployment across local and cluster-hosted infrastructure.

Official documentation: <https://docs.docker.com/>

Chapter B

Prompts

B.1 Edge Case Handling

B.1.1 Figure Summary Prompt

""You are analyzing a page from an academic paper. Describe ONLY the figure, chart, diagram, or image on this page.

If a figure caption is visible on the page, include it verbatim at the start of your response under "caption".

Respond in plain text only. Do not use HTML, markdown, or any other markup language.

Provide a structured response with these fields: - caption: The exact caption if visible, otherwise "not found".

- type: (e.g. bar chart, line graph, schematic diagram, microscopy image, flow chart)
- content: What the figure shows - key variables, trends, comparisons, or structures.
- axes_labels: If applicable, what are the x and y axes (or equivalent labels)?
- key_finding: The single most important thing this figure communicates.

Be concise and precise. Do not speculate beyond what is visible.""

B.1.2 Table Reconstruction Prompt

""You are analyzing a page from an academic paper. Extract ONLY the table visible on this page as a markdown table.

If a table caption is visible on the page, include it in verbatim at the start of your response under "caption"

Preserve all column headers and cell values exactly as they appear. Return ONLY the markdown table and the table caption, nothing else.""

B.1.3 Table Summary Prompt

""You are summarizing a table from an academic paper for use in a semantic search index. Write a 2-3 sentence plain English summary of what this table contains, what it measures or compares, and any notable values or findings. Do not reproduce the table. Do not use bullet points or markdown formatting. Return only the summary text.

Table: markdown""

B.1.4 Community Summary Prompt

""You are summarizing a cluster of academic papers that cite each other. Given the titles and abstracts below, write a 2-3 sentence summary describing: - The shared research theme or topic - The key methods or approaches used - What makes this cluster coherent

Be specific and technical. Do not use vague phrases like "various topics".

Papers: papers

Summary: ""

B.1.5 Curated Dataset Prompt

"" You are a scientific research assistant answering questions using only the provided scientific material [material]

Your task is to answer the question while determining whether it requires multi-hop reasoning across papers.

Instructions:

- Use ONLY the provided material. Do not use external knowledge.
- First determine whether the question requires synthesis across multiple papers.
- If the question can be fully answered from a single paper, return: "single_source": true
- Otherwise, answer using evidence synthesized across papers.
- The answer must combine information from multiple sources and cover relevant aspects when available (methods, comparisons, results, limitations, applications, etc.).
- Do not invent information or infer unsupported claims.

- If the provided material does not contain sufficient evidence to answer across papers, return:

Return JSON only in the following format:

"query": "<original question>", "answer": "<evidence-grounded answer synthesized across papers>", "papers_used": ["<paper₁> ", "<paper₂> ", "<paper₃> "]

B.1.6 Comparative LLM-judge Prompts

Retrieval Comparison Prompt

You are an expert evaluator comparing two retrieval systems for academic question answering.

You have NO access to ground truth. Evaluate based solely on the query and retrieved contexts.

QUERY:
<query>

SYSTEM A
Retrieved <n> context chunks from these papers/sections:
<context summary>

Full retrieved context:
<context>

SYSTEM B
Retrieved <n> context chunks from these papers/sections:
<context summary>

Full retrieved context:
<context>

Evaluate the two retrieval results according to the following criteria:

1. Relevance
2. Coverage
3. Specificity
4. Faithfulness support
5. Noise

Important instructions:

- Do not prefer a system just because it retrieved more text.
- Do not penalize a system for being concise if it contains the necessary evidence.

- If both systems retrieve useful evidence, prefer the one with better coverage and less noise.
- If neither system retrieves useful evidence, choose "tie".
- Ignore formatting differences unless they affect interpretability.
- Base your judgment only on the query and retrieved context.

Respond in JSON only with:

```
{
  "retrieval_winner": "A" | "B" | "tie",
  "relevance_a": 1-5,
  "relevance_b": 1-5,
  ...
}
```

Answer Comparison Prompt

You are an expert evaluator comparing two answers to an academic question.

You have NO access to ground truth. Evaluate based solely on the query and answers.

QUERY:

<query>

--- SYSTEM A Answer ---

<answer>

--- SYSTEM B Answer ---

<answer>

Compare the two answers:

1. Which more directly and completely addresses the query?
2. Which is better supported with specific details and citations?
3. Which contains less irrelevant content?

Respond in JSON only:

```
{
  "answer_winner": "A" | "B" | "tie",
  "A_answer_score": <float 0-1>,
  "B_answer_score": <float 0-1>,
  "reasoning": "<3-4 sentences>"
}
```

Chapter C

Graph Schema and Constraints

C.1 Knowledge Graph Schema and Index Configuration

This appendix describes the Neo4j knowledge graph schema used in the GraphRAG pipeline, including node types, relationship types, uniqueness constraints, and vector and full-text indexes.

C.1.1 Node Types

The graph contains nine node types. Each node type has a unique `id` property enforced by a node property uniqueness constraint.

Table C.1: Node types and their indexed properties

Node Label	Description	Indexed Properties
Paper	Academic paper	<code>id</code> (unique), <code>abstract_embedding</code> (vector)
Section	Logical section within a paper	<code>id</code> (unique)
Chunk	Text window within a section	<code>id</code> (unique), <code>embedding</code> (vector), <code>text</code> (fulltext)
Community	Leiden community cluster	<code>embedding</code> (vector)
Author	Paper author	<code>id</code> (unique)
Figure	Figure within a paper	<code>id</code> (unique), <code>embedding</code> (vector)
Table	Table within a paper	<code>id</code> (unique), <code>embedding</code> (vector)
Equation	Equation within a paper	<code>id</code> (unique)
Reference	Bibliographic reference	<code>id</code> (unique)

C.1.2 Relationship Types

The graph contains twelve relationship types representing structural, semantic, and citation connections between nodes.

Table C.2: Relationship types and their meaning

Relationship	Source	Target
HAS_SECTION	Paper	Section
HAS_CHUNK	Section	Chunk
HAS_AUTHOR	Paper	Author
HAS_FIGURE	Paper	Figure
HAS_TABLE	Paper	Table
HAS_EQUATION	Paper	Equation
BELONGS_TO	Paper	Community
CITES	Paper	Paper
SIMILAR	Chunk	Chunk
NEXT_CHUNK	Chunk	Chunk
NEXT_SECTION	Section	Section

C.1.3 Indexes

The graph uses three categories of indexes: vector indexes for semantic similarity search, a full-text index for keyword search, and range indexes enforcing uniqueness constraints.

Vector Indexes

Vector indexes are built using Neo4j’s native vector index (provider `vector-3.0`) over `sentence-transformers/all-mpnet-base-v2` embeddings. Five vector indexes are defined:

Table C.3: Vector indexes

Index Name	Node Label	Property
<code>chunk_embeddings</code>	Chunk	<code>embedding</code>
<code>community_embeddings</code>	Community	<code>embedding</code>
<code>figure_embeddings</code>	Figure	<code>embedding</code>
<code>table_embeddings</code>	Table	<code>embedding</code>
<code>paper_abstract_embeddings</code>	Paper	<code>abstract_embedding</code>

The `chunk_embeddings` index is the primary retrieval index, queried via `db.index.vector.queryNodes` during hybrid search. The `community_embeddings` index is queried by the graph traversal agent to identify relevant Leiden communities.

Full-Text Index

A single full-text index supports BM25 keyword search over chunk text:

This index is queried via `db.index.fulltext.queryNodes` and combined with vector search results using Reciprocal Rank Fusion (RRF) during hybrid retrieval.

Table C.4: Full-text index

Index Name	Node Label	Property
chunk_text_index	Chunk	text

Uniqueness Constraints and Range Indexes

Each node type has a uniqueness constraint on its `id` property, backed by a range index. These ensure referential integrity during graph construction and enable fast point lookups by node identifier.

Table C.5: Uniqueness constraints and their backing range indexes

Node Label	Constraint
Paper	Paper.id unique
Section	Section.id unique
Chunk	Chunk.id unique
Author	Author.id unique
Figure	Figure.id unique
Table	Table.id unique
Equation	Equation.id unique
Reference	Reference.id unique

C.1.4 Schema Diagram

The core traversal path used by the retrieval pipeline is:

```
(:Paper)-[:BELONGS_TO]->(:Community)
(:Paper)-[:HAS_SECTION]->(:Section)-[:HAS_CHUNK]->(:Chunk)
(:Paper)-[:CITES]->(:Paper)
(:Chunk)-[:NEXT_CHUNK]->(:Chunk)
(:Section)-[:NEXT_SECTION]->(:Section)
```

The graph traversal agent navigates this structure to locate relevant papers via community membership, then retrieves section-level content using the `HAS_SECTION` and `HAS_CHUNK` traversal paths.

C.1.5 Visualisations

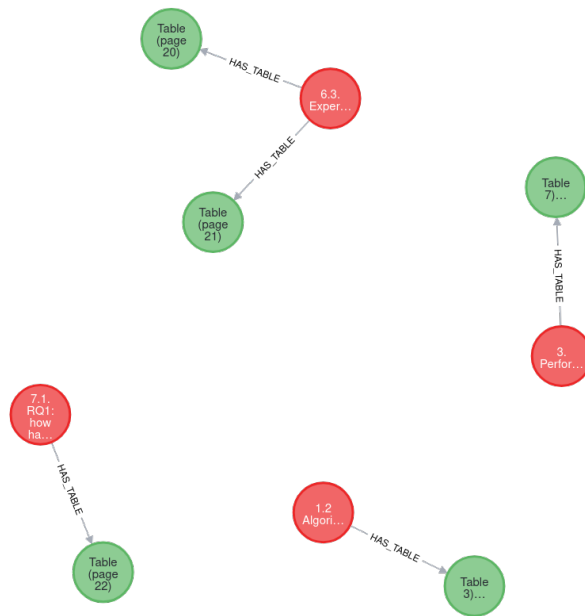


Figure C.1: The HAS_TABLE edge connecting table nodes and section nodes. This edge creates a relation for a table to the section it is located in.

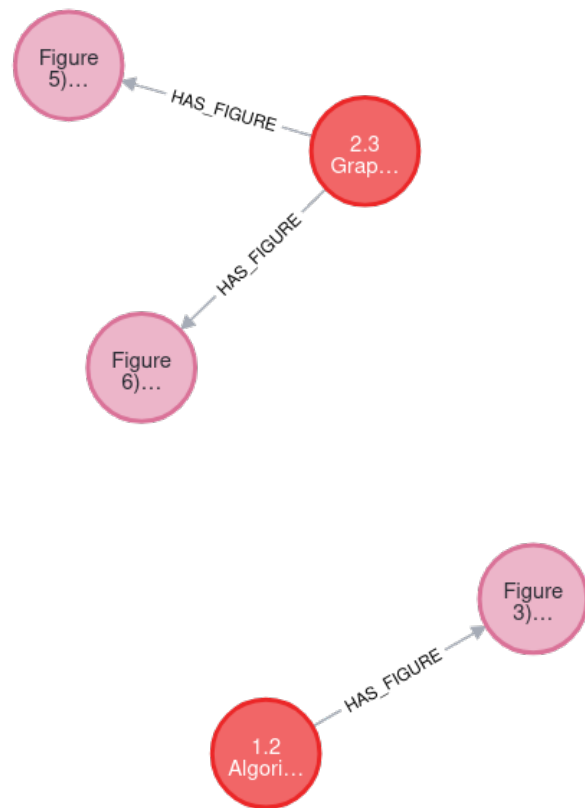


Figure C.2: The HAS_FIGURE edge works in a similar fashion. Connecting figures to their corresponding section

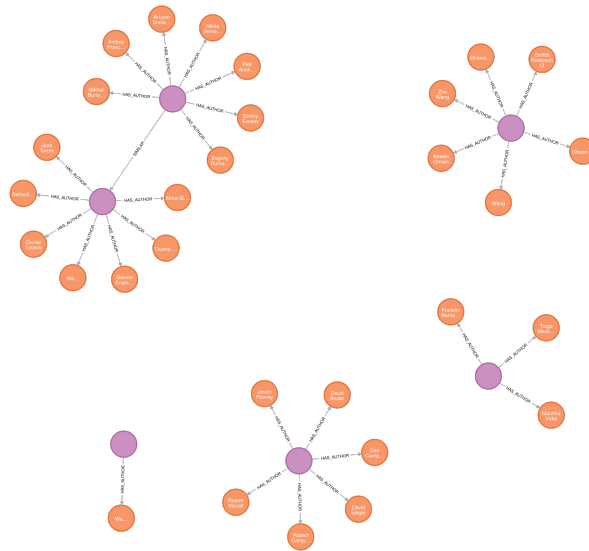


Figure C.3: The `HAS_AUTHOR` edge connects authors with papers. In this figure is also possible to see the `SIMILAR` connection between two paper nodes. Constructed by computing the semantic similarity of abstracts between papers.

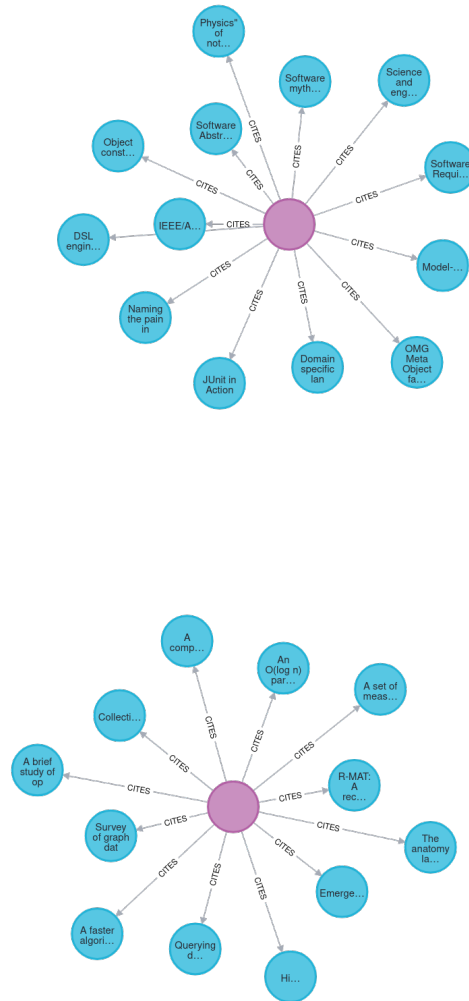


Figure C.4: The CITES relationship is a central edge type in the knowledge graph, connecting papers with reference nodes. The CITES relation also enables graph traversal and multi-hop retrieval in the case where two papers share reference nodes. In the example shown, the papers do not reference one another, and consequently no reference node is shared between them.

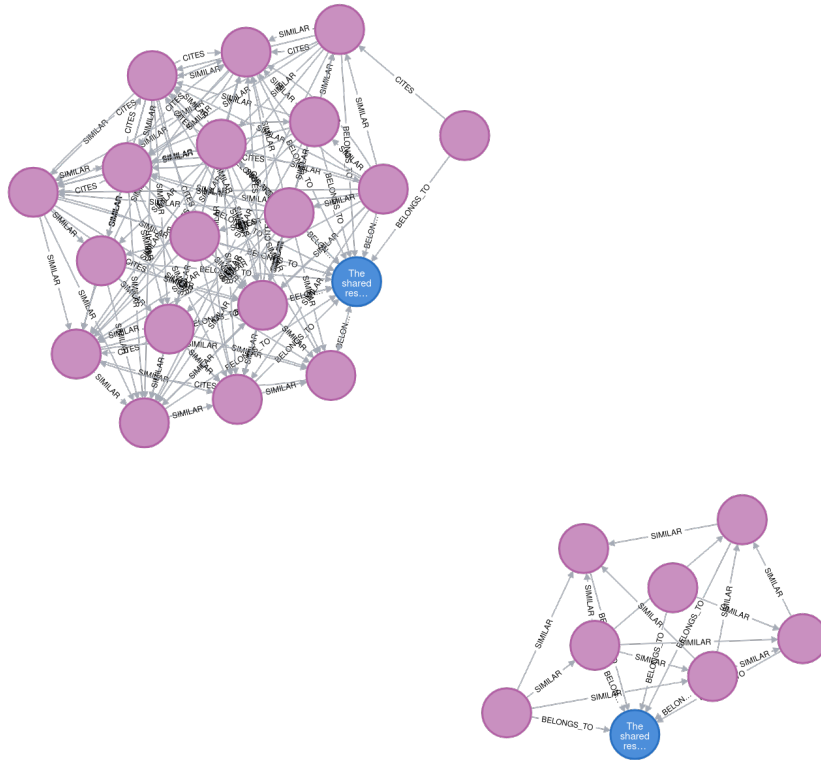


Figure C.5: The `BELONGS_TO` relationship links each paper node to a community node. Community clusters are computed using the Leiden algorithm on a combined graph consisting of weighted `CITES` and `SIMILAR` relationships.

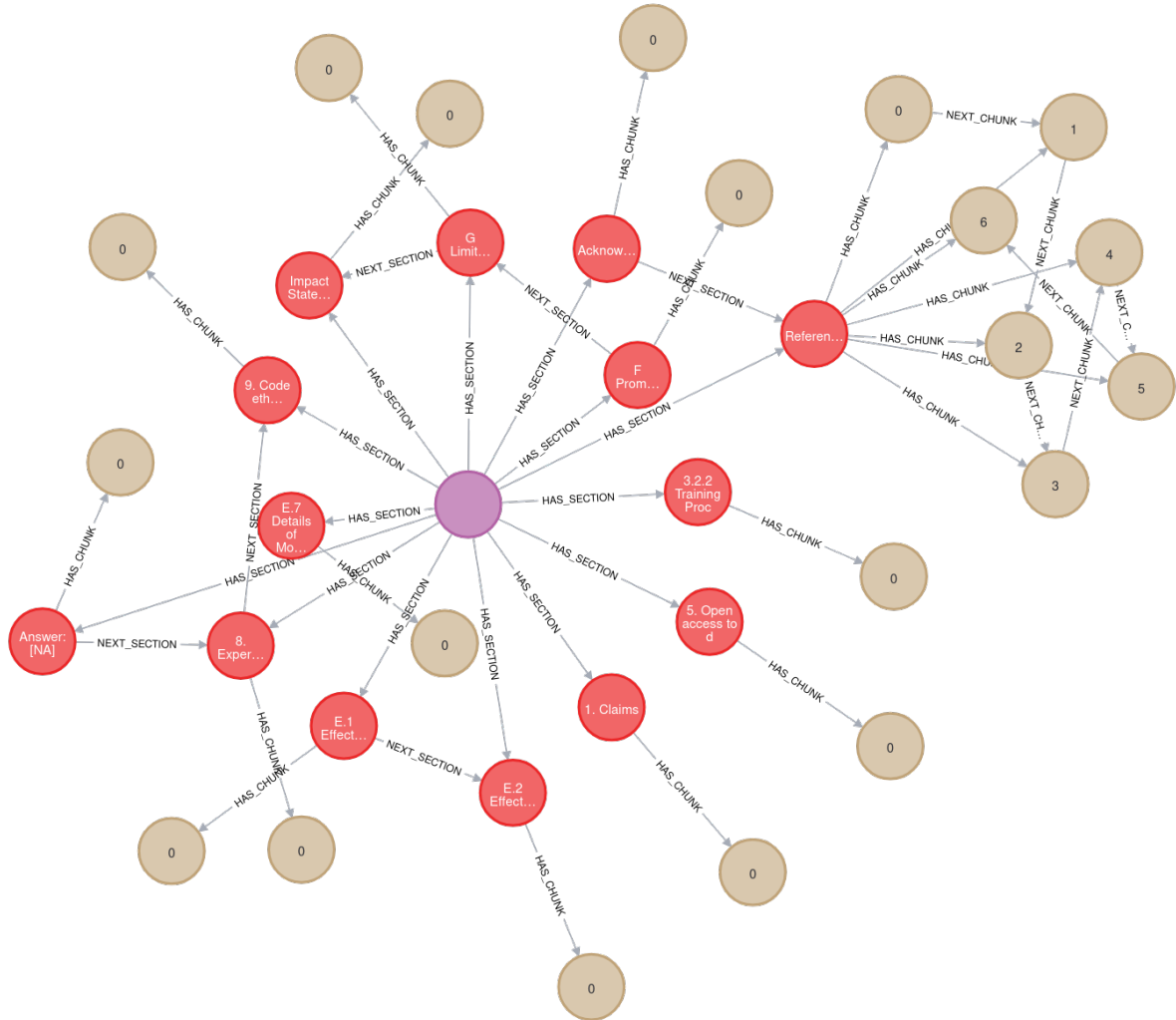


Figure C.6: Several edges are pictured in the figure. `HAS_CHUNK` connects section and chunk nodes, `HAS_SECTION` connects paper and section nodes, `NEXT_SECTION` and `NEXT_CHUNK` connects sequential section and chunk nodes respectively.

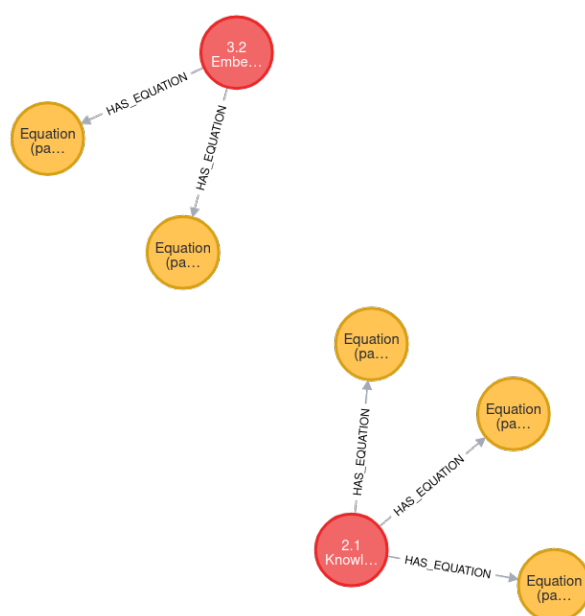


Figure C.7: This figure shows the HAS_EQUATION edge connecting equations with corresponding sections

Chapter D

Additional Results

Include extended result tables, per-query outputs, failure cases, and additional plots.

Chapter E

Algorithms

E.0.1 Community Creation

Algorithm 1 Community graph construction

Require: Paper nodes $P = \{p_1, p_2, \dots, p_n\}$

Require: Abstract embeddings $\mathbf{a}_i$ for each paper p_i

Require: Similarity threshold τ

- 1: Initialize graph G with all paper nodes P
- 2: Add existing citation relationships CITES to G
- 3: **for** each pair of papers (p_i, p_j) where $i < j$ **do**
- 4: Compute cosine similarity:

$$s_{ij} = \frac{\mathbf{a}_i \cdot \mathbf{a}_j}{\|\mathbf{a}_i\| \|\mathbf{a}_j\|}$$

- 5: **if** $s_{ij} \geq \tau$ **then**
 - 6: Add weighted SIMILAR relationship between p_i and p_j
 - 7: Set relationship weight to s_{ij}
 - 8: **end if**
 - 9: **end for**
 - 10: Run Leiden community detection on the graph containing CITES and weighted SIMILAR relationships
 - 11: **for** each detected community c_k **do**
 - 12: **if** $|c_k| \geq \text{minimum community size}$ **then**
 - 13: Create Community node
 - 14: Connect each paper in c_k to the community using BELONGS_TO
 - 15: **else**
 - 16: Treat papers in c_k as isolated papers during retrieval
 - 17: **end if**
 - 18: **end for**
 - return** Graph with SIMILAR, CITES, Community, and BELONGS_TO structure
-

E.0.2 Embedding Windows

Algorithm 2 Greedy construction of section-aware embedding windows

Require: Ordered chunks $C = \{c_1, c_2, \dots, c_n\}$ within a section

Require: Maximum window size $T_{\max}$, minimum window size $T_{\min}$

```

1: Initialize empty list of windows  $W$ 
2: Initialize empty current window  $w$ 
3: Initialize token count  $t \leftarrow 0$ 
4: for each chunk  $c_i \in C$  do
5:   if  $\text{tokens}(c_i) > T_{\max}$  then
6:     if  $w$  is not empty then
7:       Append  $w$  to  $W$ 
8:       Reset  $w \leftarrow \emptyset, t \leftarrow 0$ 
9:     end if
10:    Append  $\{c_i\}$  to  $W$ 
11:   else if  $t + \text{tokens}(c_i) > T_{\max}$  then
12:     Append  $w$  to  $W$ 
13:     Start new window  $w \leftarrow \{c_i\}$ 
14:      $t \leftarrow \text{tokens}(c_i)$ 
15:   else
16:     Add  $c_i$  to  $w$ 
17:      $t \leftarrow t + \text{tokens}(c_i)$ 
18:   end if
19: end for
20: if  $w$  is not empty then
21:   Append  $w$  to  $W$ 
22: end if
23: for each trailing window  $w_j \in W$  where  $\text{tokens}(w_j) < T_{\min}$  do
24:   Merge  $w_j$  into the previous window  $w_{j-1}$ 
25: end for
26: Assign one shared embedding_id and embedding vector to all chunks in each window
return  $W$ 

```

Chapter F

Json Schemas

F.0.1 Ingestion Schema

During ingestion, each processed paper is converted into an intermediate JSON representation before being loaded into Neo4j. This representation separates document extraction from database insertion and makes the graph construction process easier to inspect and debug.

Each JSON file contains four top-level fields:

- `paper_id`: a unique identifier for the processed paper.
- `source_file`: the path to the original PDF file.
- `nodes`: a list of extracted graph nodes.
- `edges`: a list of extracted graph relationships.

A simplified version of the schema is shown below:

```
{
  "paper_id": "paper_<normalized_title>",
  "source_file": "<path_to_pdf>",
  "nodes": [
    {
      "id": "<unique_node_id>",
      "type": "Paper | Author | Reference | Section | Chunk |
              Table | Figure | Equation",
      "properties": {
        "...": "node-specific properties"
      }
    }
  ]
}
```

```

],
"edges": [
  {
    "source": "<source_node_id>",
    "target": "<target_node_id>",
    "type": "HAS_AUTHOR | CITES | HAS_SECTION | HAS_CHUNK |
            HAS_TABLE | HAS_FIGURE | HAS_EQUATION |
            NEXT_SECTION | NEXT_CHUNK | DESCRIBED_IN"
  }
]
}

```

The node types correspond to the entities extracted during ingestion. For example, **Paper** nodes contain bibliographic metadata, **Section** and **Chunk** nodes represent the textual document hierarchy, and **Table**, **Figure**, and **Equation** nodes represent non-standard document elements. Relationship types encode document structure, sequential order, citation information, and links between visual elements and the text in which they are described.

An example extracted paper representation contains a **Paper** node with metadata such as title, abstract, publication date, and source, followed by nodes for authors, references, sections, chunks, tables, figures, and equations. Edges then connect the paper to its authors, references, sections, and extracted content elements.